US 20220027905A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2022/0027905 A1**

**Wright et al.** (43) **Pub. Date: Jan. 27, 2022**

(54) **COMPUTER IMPLEMENTED SYSTEM AND METHOD FOR ENABLING ACCESS TO DATA STORED ON A BLOCKCHAIN**

(71) Applicant: **nChain Holdings Limited**, St. John's (AG)

(72) Inventors: **Craig Steven Wright**, London (GB); **Owen Vaughan**, London (GB); **Jack Owen Davies**, London (GB); **Chloe Ceren Tartan**, London (GB)

(21) Appl. No.: **17/296,954**

(22) PCT Filed: **Nov. 14, 2019**

(86) PCT No.: **PCT/IB2019/059809**

§ 371 (c)(1),
(2) Date: **May 25, 2021**

(30) **Foreign Application Priority Data**

Nov. 27, 2018 (GB) ..... 1819284.9
Nov. 27, 2018 (GB) ..... 1819286.4
Nov. 27, 2018 (GB) ..... 1819290.6
Nov. 27, 2018 (GB) ..... 1819291.4
Nov. 27, 2018 (GB) ..... 1819293.0
Nov. 27, 2018 (GB) ..... 1819297.1
Nov. 27, 2018 (GB) ..... 1819299.7

**Publication Classification**

(51) **Int. Cl.**
***G06Q 20/38*** (2006.01)
***H04L 9/32*** (2006.01)

(52) **U.S. Cl.**
CPC ... ***G06Q 20/38215*** (2013.01); ***G06Q 20/3827*** (2013.01); *H04L 2209/38* (2013.01); ***H04L 9/3239*** (2013.01); *H04L 2209/56* (2013.01); ***G06Q 20/3829*** (2013.01)

(57) **ABSTRACT**

A method of enabling access to data stored on a blockchain is disclosed. The method comprises generating at least one first blockchain transaction of a blockchain, wherein the first data is stored in at least one first blockchain transaction and is inaccessible to a first participant in the absence of first access data. The method further comprises generating a second blockchain transaction of the blockchain, wherein the second blockchain transaction contains means for causing the first access data to be made available to the first participant.

| $TxID_{Bob}$ | | | |
|---|---|---|---|
| Inputs | | Outputs | |
| Value | Script | Value | Script |
| $x$ | $<Sig\ P_B> <P_B>$ | $x$ | [Private Key Puzzle $P_1, r_0$] [CheckSig $P_A$] |

| *TxID* | |
|---|---|
| Inputs | Outputs |
| <*Sig P*> <*P*> | OP_RETURN <Metanet Flag> <Attribute 1> <Attribute 2> |
| | <Content 1> OP_DROP <$H(P')$> [CheckSig $P'$] |

Fig. 1

| *TxID* | |
|---|---|
| Inputs | Outputs |
| <*Sig P*> <*P*> <Content 1> OP_DROP | OP_RETURN <Metanet Flag> <Attribute 1> <Attribute 2> |

Fig. 2

| $TxID_1$ | |
|---|---|
| Inputs | Outputs |
| < $Sig\ P$> <$P$> | OP_RETURN <Metanet Flag> <Attribute 1,1 = Content name><br><Attribute 1,2 = Recombination scheme><br><Attribute 1,3 = Content chunk index 1> |
| | <Content chunk 1> OP_DROP <$H(P')$> [CheckSig $P'$] |

| $TxID_2$ | |
|---|---|
| Inputs | Outputs |
| < $Sig\ P$> <$P$> | OP_RETURN <Metanet Flag> <Attribute 2,1 = Content name><br><Attribute 2,2 = Recombination scheme><br><Attribute 2,3 = Content chunk index 2> |
| | <Content chunk 2> OP_DROP <$H(P')$> [CheckSig $P'$] |

Fig. 3

| $TxID_{Bob}$ | | | |
|---|---|---|---|
| Inputs | | Outputs | |
| Value | Script | Value | Script |
| $x$ | < $Sig\ P_B$> <$P_B$> | $x$ | [Private Key Puzzle $P_1, r_0$] [CheckSig $P_A$] |

Fig. 4

| $TxID_{Alice}$ | | | |
|---|---|---|---|
| Inputs | | Outputs | |
| Value | Script | Value | Script |
| $x$ | $< Sig\ P_A > < P_A > < Sig\ P_1, r_0 > < P_1 >$ | $x$ | [CheckSig $P_A$] |

Fig. 5

| Use | | Alice | Bob |
|---|---|---|---|
| *Token Issuance* | Purchase of 10 tokens | $I_{Alice} = k$ | $I_{Bob} = H^{10}(Y)$ |
| *Token redemption* | Redeem token 1 | $X_1$ | $T_1 = H^9(Y)$ |
| | Redeem token 2 | $X_2$ | $T_2 = H^8(Y)$ |
| | ... | ... | ... |
| | Redeem token 10 | $X_{10}$ | $T_{10} = Y$ |

Fig. 6

| $TxID_{Alice}$ | | | |
|---|---|---|---|
| Inputs | | Outputs | |
| Value | Script | Value | Script |
| $x$ | $< Sig\ P_A > < P_A >$ | $x$ | [Hash Puzzle $H(I_{Alice})$] [Hash Puzzle $H(I_{Bob})$] [CheckSig $P_B$] |

**Phase 1.1 (Alice to Bob):**

Fig. 7

| $TxID_{Bob}$ | | | |
|---|---|---|---|
| Inputs | | Outputs | |
| Value | Script | Value | Script |
| $10 + x$ | $< Sig\ P_B > < P_B >$ | $10 + x$ | [Hash Puzzle $H(I_{Alice})$] [Hash Puzzle $H(I_{Bob})$] [CheckSig $P_A$] |

**Phase 1.2 (Bob to Alice):**

Fig. 8

**Phase 2.1 (Bob to Alice):**

| $TxID_{Bob}$ | | | |
|---|---|---|---|
| Inputs | | Outputs | |
| Value | Script | Value | Script |
| $x$ | $< Sig\ P_B > < P_B >$ | $x$ | [Hash Puzzle $H(X_1)$] [Hash Puzzle $H(T_1)$] [CheckSig $P_A$] |

Fig. 9

**Phase 2.2 (Alice to Bob):**

| $TxID_{Alice}$ | | | |
|---|---|---|---|
| Inputs | | Outputs | |
| Value | Script | Value | Script |
| $x$ | $< Sig\ P_A > < P_A >$ | $x$ | [Hash Puzzle $H(X_1)$] [Hash Puzzle $H(T_1)$] [CheckSig $P_B$] |

Fig. 10

**Phase 2.3 (Alice to Alice):**

| $TxID_{Alice}$ | | | |
|---|---|---|---|
| Inputs | | Outputs | |
| Value | Script | Value | Script |
| $x$ | $< Sig\ P_A > < P_A > < T_1 > < X_1 >$ | $x$ | [CheckSig $P_A$] |

Fig. 11

**Phase 2.4 (Bob to Bob):**

| $TxID_{Bob}$ | | | |
|---|---|---|---|
| Inputs | | Outputs | |
| Value | Script | Value | Script |
| $x$ | $< Sig\ P_B > < P_B > < T_1 > < X_1 >$ | $x$ | [CheckSig $P_B$] |

Fig. 12

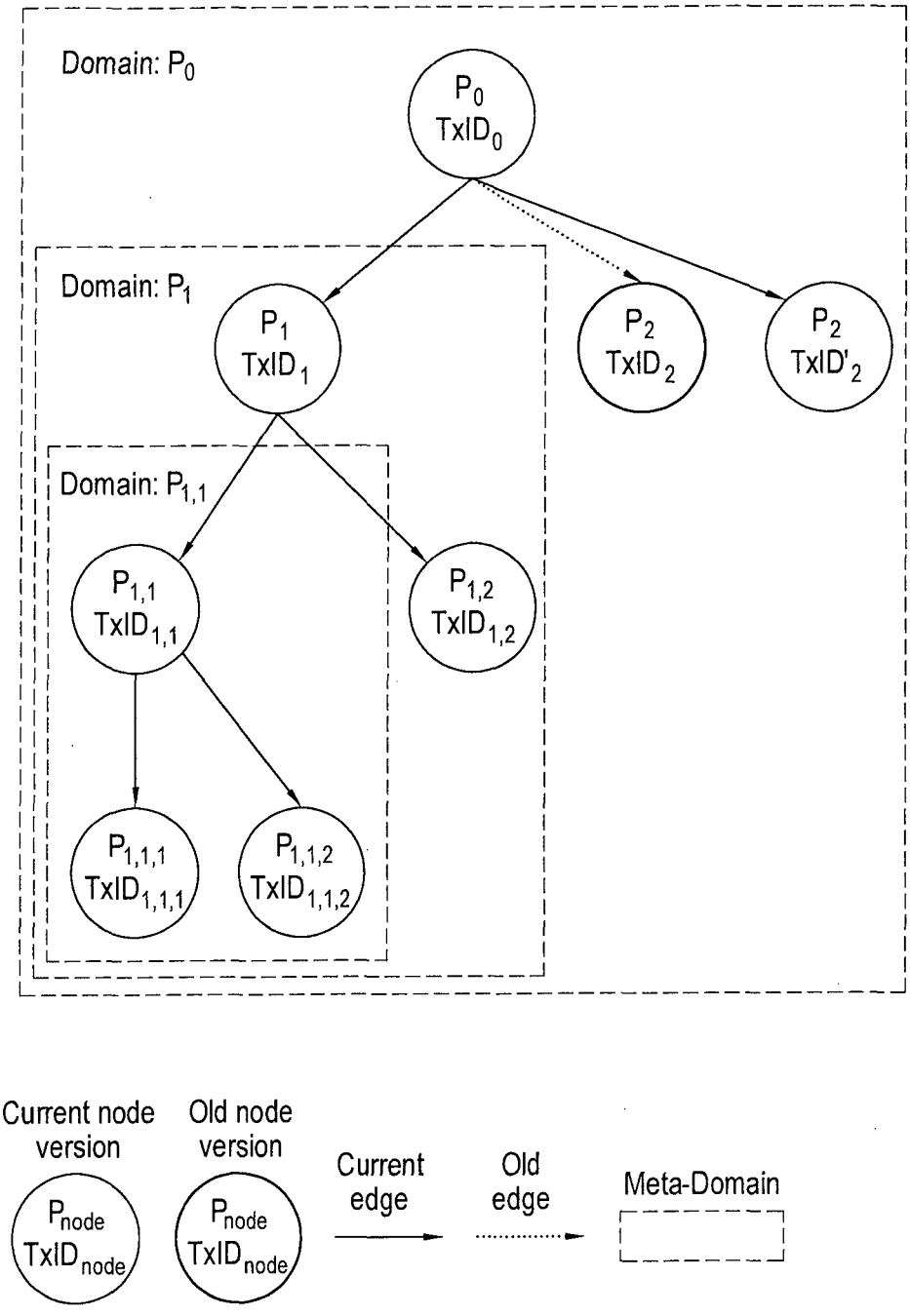
Domain: $P_0$
$P_0$ $TxID_0$
Domain: $P_1$
$P_1$ $TxID_1$
$P_2$ $TxID_2$
$P_2$ $TxID'_2$
Domain: $P_{1,1}$
$P_{1,1}$ $TxID_{1,1}$
$P_{1,2}$ $TxID_{1,2}$
$P_{1,1,1}$ $TxID_{1,1,1}$
$P_{1,1,2}$ $TxID_{1,1,2}$
Current node version
Old node version
$P_{node}$ $TxID_{node}$
$P_{node}$ $TxID_{node}$
Current edge
Old edge
Meta-Domain

Fig. 13

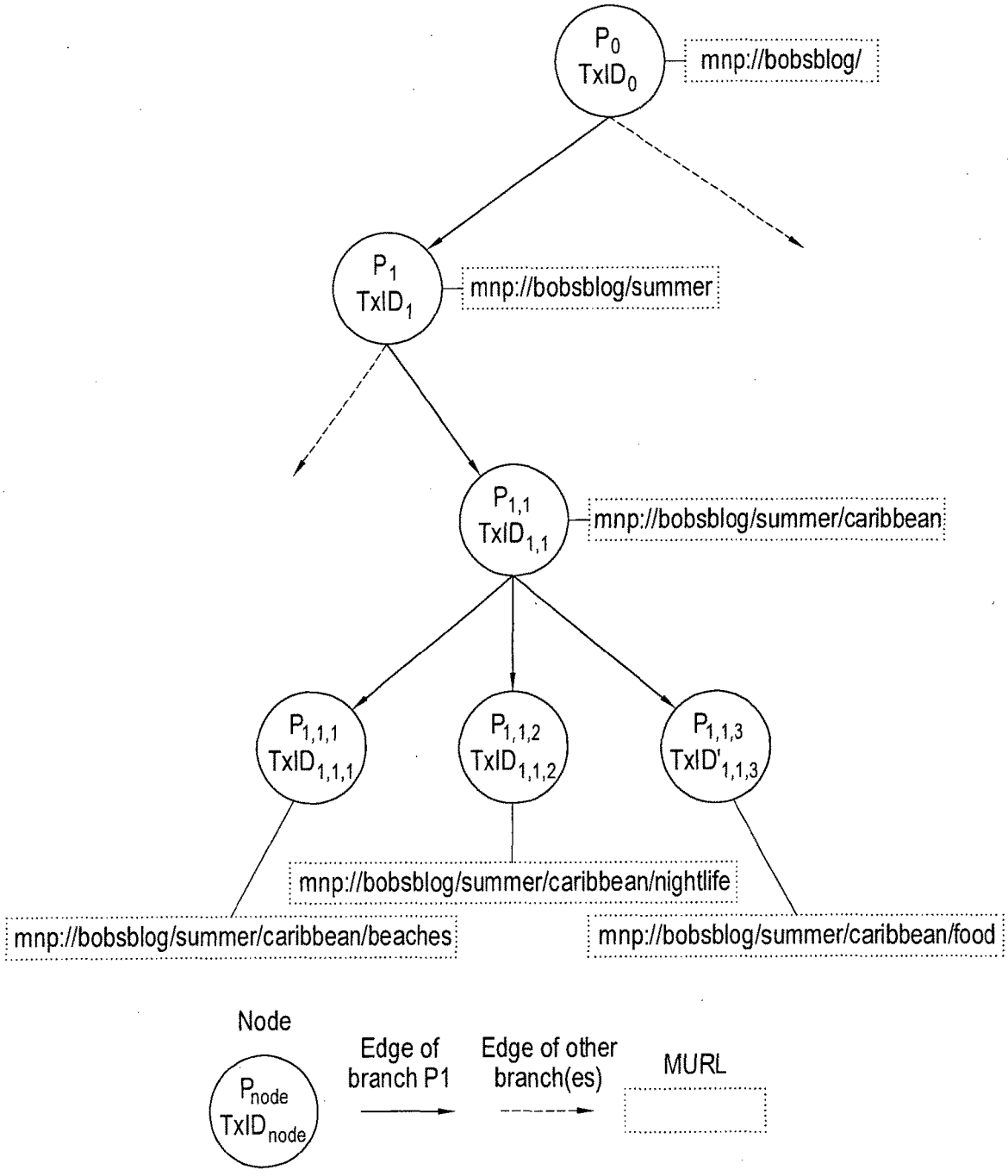
P0
TxID0
mnp://bobsblog/
P1
TxID1
mnp://bobsblog/summer
P1,1
TxID1,1
mnp://bobsblog/summer/caribbean
P1,1,1
TxID1,1,1
P1,1,2
TxID1,1,2
P1,1,3
TxID'1,1,3
mnp://bobsblog/summer/caribbean/nightlife
mnp://bobsblog/summer/caribbean/beaches
mnp://bobsblog/summer/caribbean/food
Node
Pnode
TxIDnode
Edge of branch P1
Edge of other branch(es)
MURL

Fig. 14

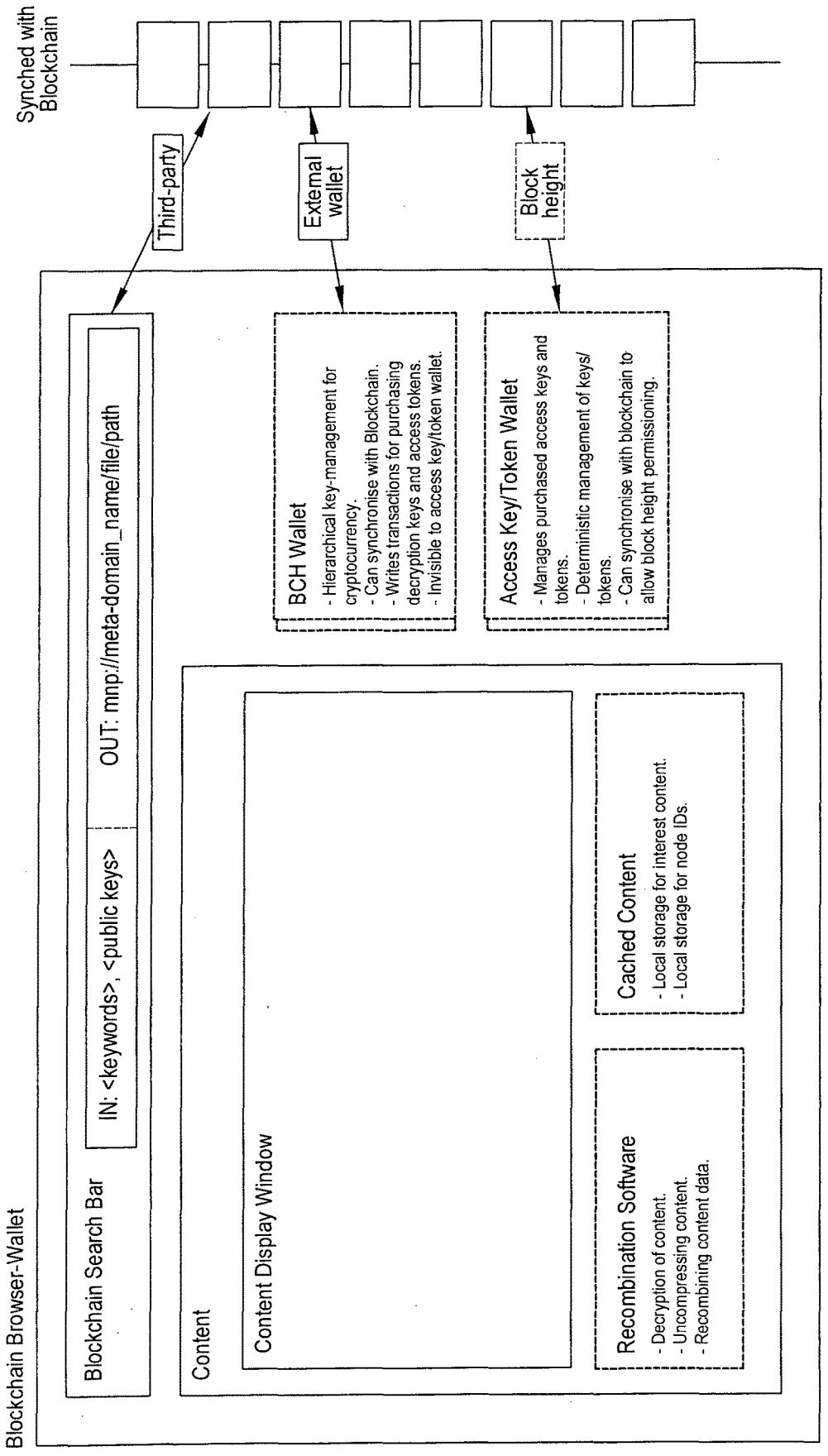
Synched with Blockchain
Third-party
External wallet
Block height
Blockchain Browser-Wallet
Blockchain Search Bar
IN: <keywords>, <public keys>
OUT: mnp://meta-domain_name/file/path
BCH Wallet
- Hierarchical key-management for cryptocurrency.
- Can synchronise with Blockchain.
- Writes transactions for purchasing decryption keys and access tokens.
- Invisible to access key/token wallet.
Access Key/Token Wallet
- Manages purchased access keys and tokens.
- Deterministic management of keys/tokens.
- Can synchronise with blockchain to allow block height permissioning.
Content
Content Display Window
Recombination Software
- Decryption of content.
- Uncompressing content.
- Recombining content data.
Cached Content
- Local storage for interest content.
- Local storage for node IDs.
Front-end component
Back-end component

Fig. 15

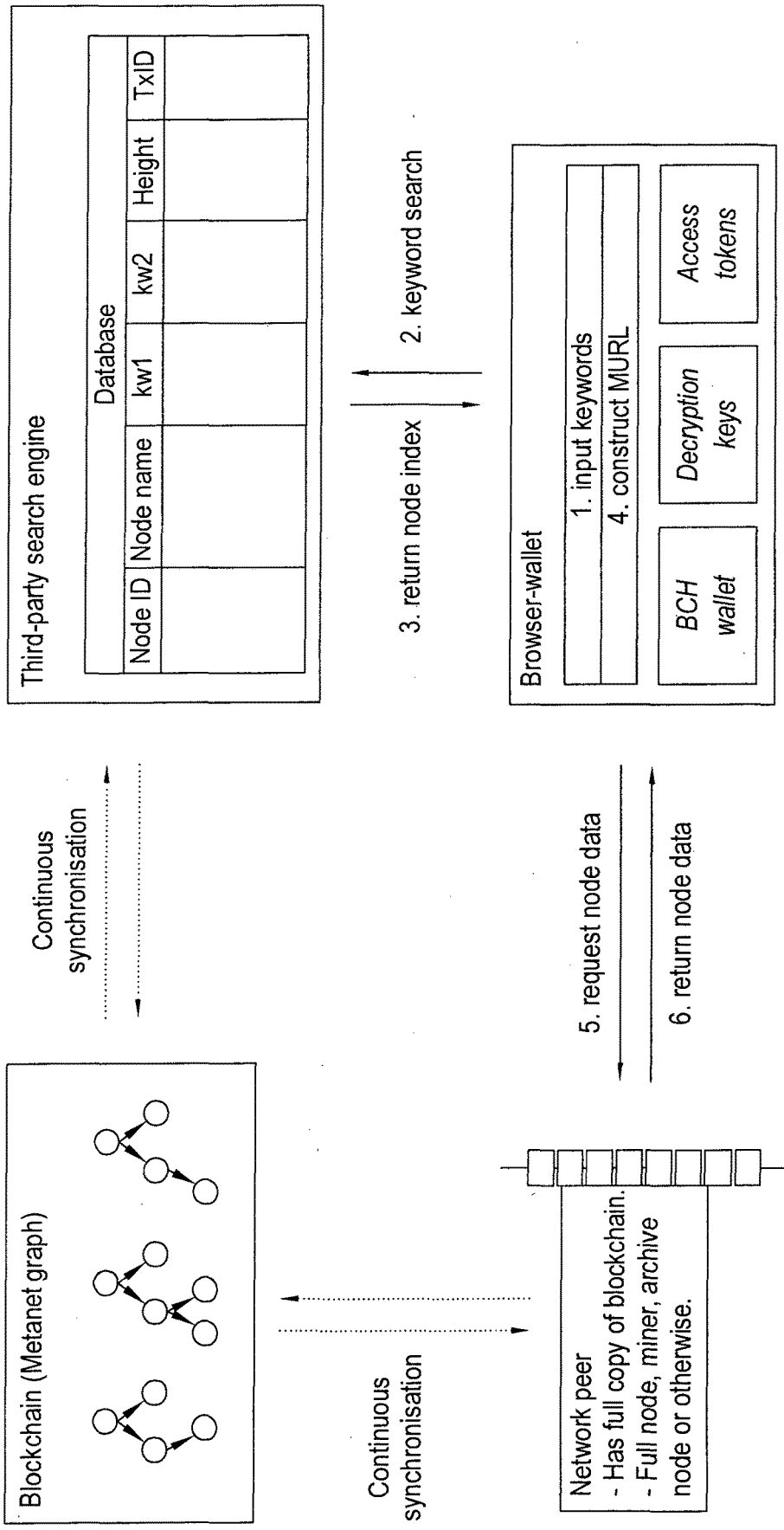
Third-party search engine
Database
Node ID
Node name
kw1
kw2
Height
TxID
2. keyword search
3. return node index
Browser-wallet
1. input keywords
4. construct MURL
BCH wallet
Decryption keys
Access tokens
Continuous synchronisation
5. request node data
6. return node data
Blockchain (Metanet graph)
Continuous synchronisation
Network peer
- Has full copy of blockchain.
- Full node, miner, archive node or otherwise.

Fig. 16

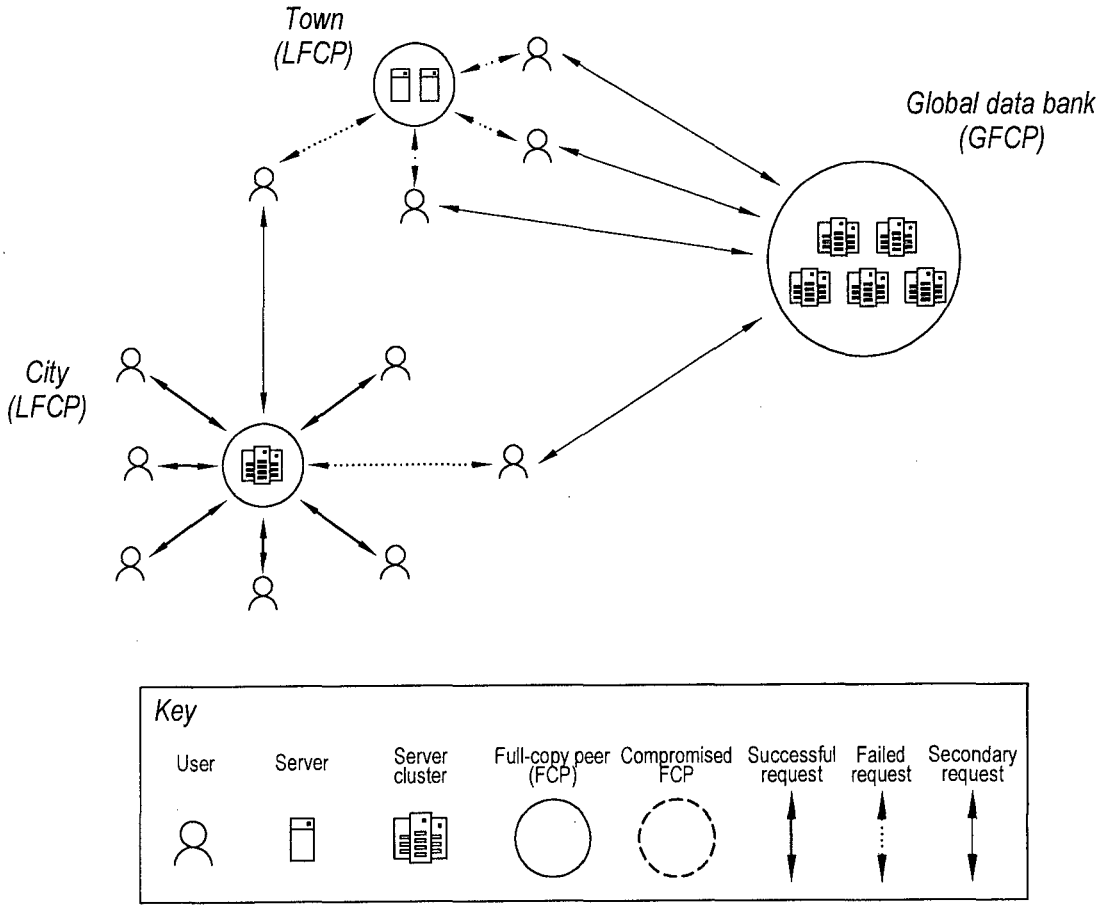
Town
(LFCP)
Global data bank
(GFCP)
City
(LFCP)
Key
User
Server
Server cluster
Full-copy peer (FCP)
Compromised FCP
Successful request
Failed request
Secondary request

Fig. 17

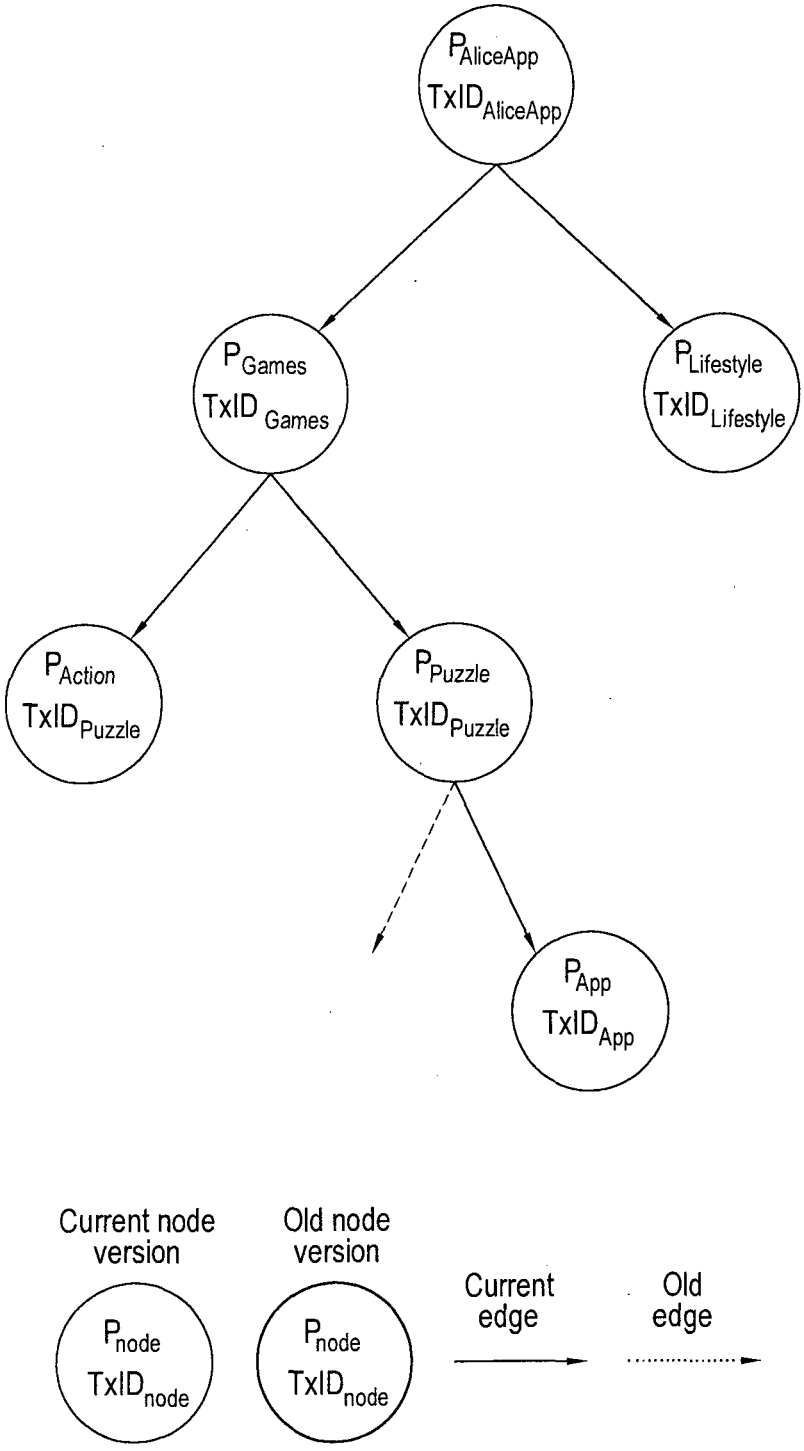
P AliceApp
TxID AliceApp
P Games
TxID Games
P Lifestyle
TxID Lifestyle
P Action
TxID Puzzle
P Puzzle
TxID Puzzle
P App
TxID App
Current node version
Old node version
P node
TxID node
P node
TxID node
Current edge
Old edge

Fig. 18

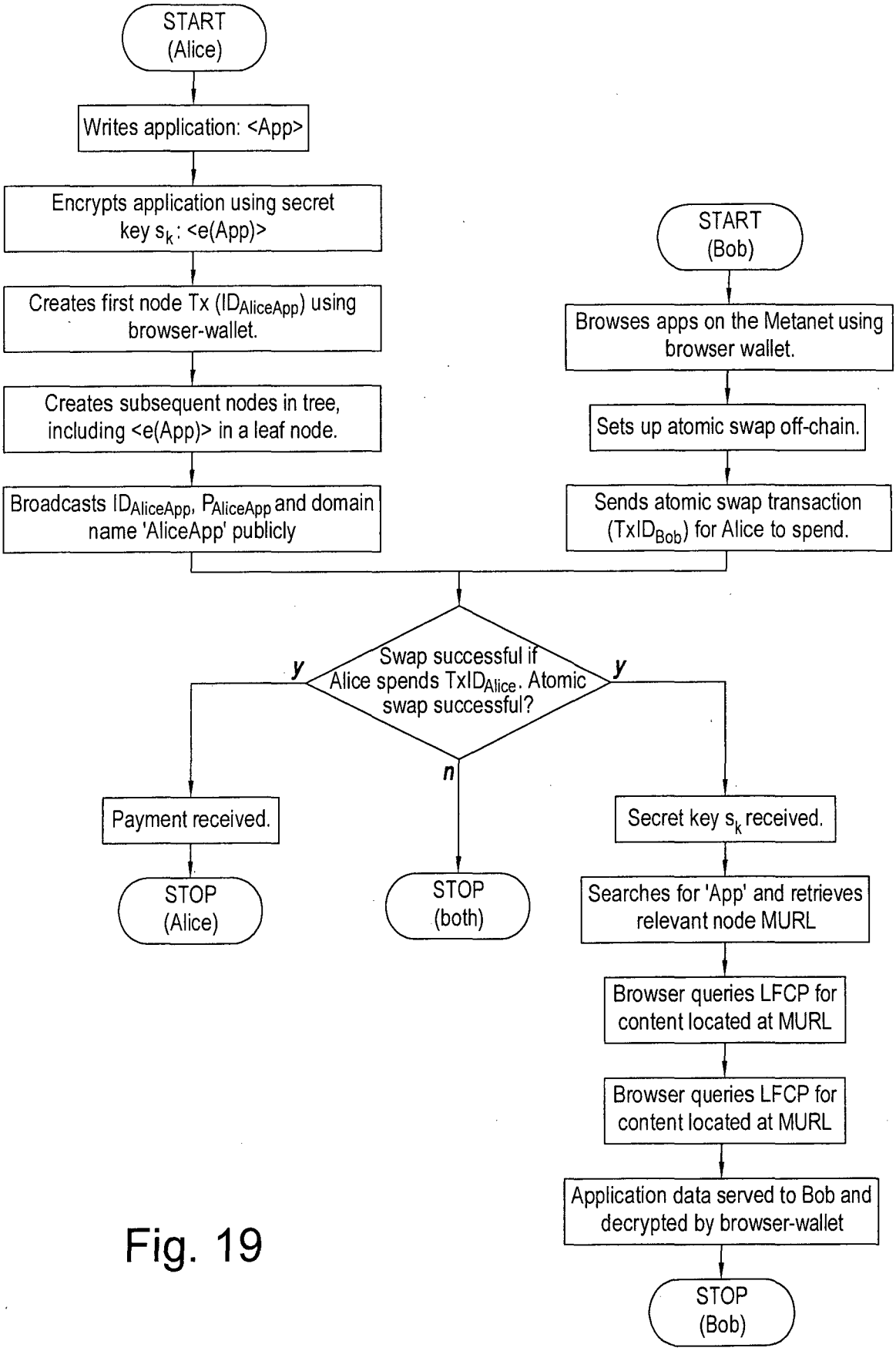
START (Alice)
Writes application: <App>
Encrypts application using secret key $s_k$: <e(App)>
Creates first node Tx ($ID_{AliceApp}$) using browser-wallet.
Creates subsequent nodes in tree, including <e(App)> in a leaf node.
Broadcasts $ID_{AliceApp}$, $P_{AliceApp}$ and domain name 'AliceApp' publicly
START (Bob)
Browses apps on the Metanet using browser wallet.
Sets up atomic swap off-chain.
Sends atomic swap transaction ($TxID_{Bob}$) for Alice to spend.
Swap successful if Alice spends $TxID_{Alice}$. Atomic swap successful?
y
n
y
Payment received.
STOP (Alice)
STOP (both)
Secret key $s_k$ received.
Searches for 'App' and retrieves relevant node MURL
Browser queries LFCP for content located at MURL
Browser queries LFCP for content located at MURL
Application data served to Bob and decrypted by browser-wallet
STOP (Bob)

Fig. 19

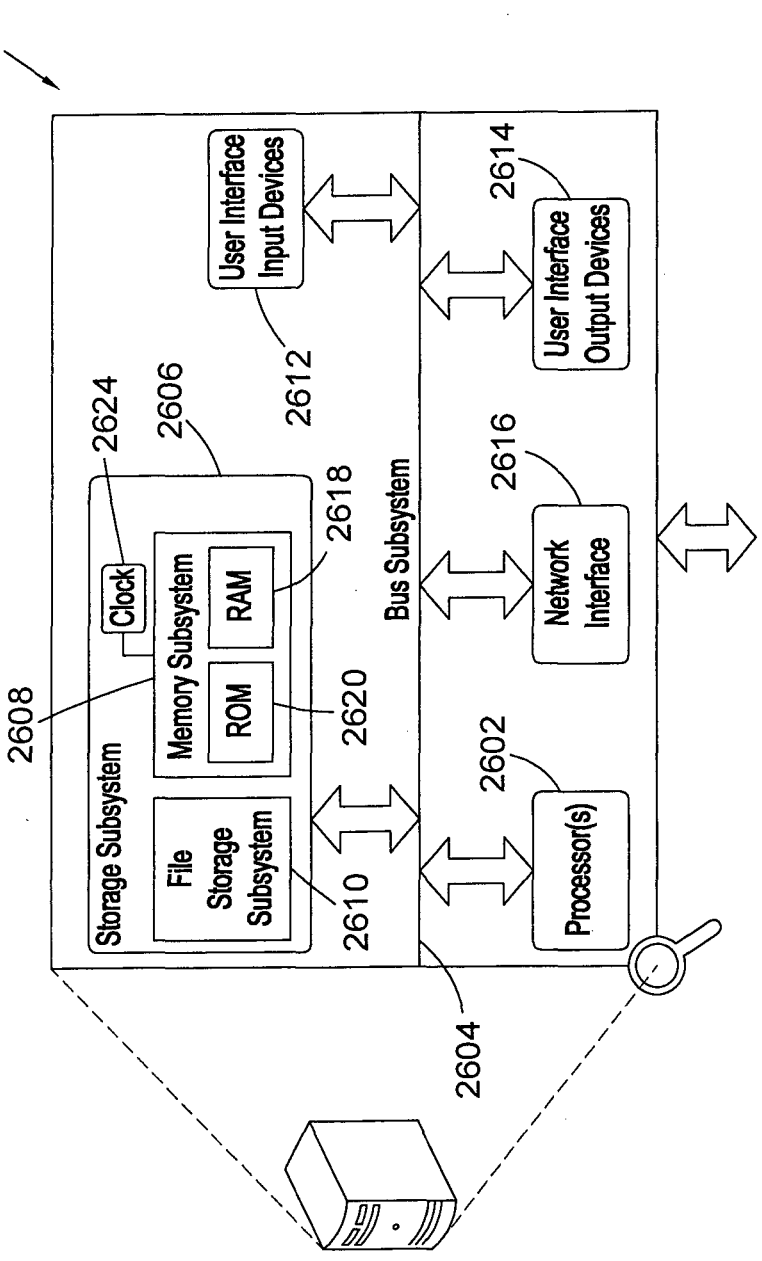
2600
2608
Storage Subsystem
File Storage Subsystem
2610
Memory Subsystem
ROM
2620
RAM
2618
Clock
2624
2606
2612
User Interface Input Devices
Bus Subsystem
2602
Processor(s)
2616
Network Interface
2614
User Interface Output Devices
2604

Fig. 20

# COMPUTER IMPLEMENTED SYSTEM AND METHOD FOR ENABLING ACCESS TO DATA STORED ON A BLOCKCHAIN

**[0001]** This invention relates generally to cryptographic techniques for improved data communication and exchange across an electronic network, and in particular a peer-to-peer network such as a blockchain network. It relates to data storage, access, retrieval and processing, and more particularly to such data-related activities on a blockchain. The invention is particularly suited, but not limited to, use in processing data in a manner similar to that provided by web sites and web pages but using the blockchain as an underlying mechanism or platform rather than web server(s). Thus, the invention provides a secure, efficient, cryptographically-enforced, alternative infrastructure for data processing and transfer.

**[0002]** In this document we use the term 'blockchain' to include all forms of electronic, computer-based, distributed ledgers. These include consensus-based blockchain and transaction-chain technologies, permissioned and un-permissioned ledgers, shared ledgers and variations thereof. The most widely known application of blockchain technology is the Bitcoin ledger, although other blockchain implementations have been proposed and developed. While Bitcoin may be referred to herein for the purpose of convenience and illustration, it should be noted that the invention is not limited to use with the Bitcoin blockchain and alternative blockchain implementations and protocols fall within the scope of the present invention. The term "user" may refer herein to a human or a processor-based resource. "Bitcoin" as used herein includes all versions and variations of protocols which derive from the Bitcoin protocol.

**[0003]** A blockchain is a peer-to-peer, electronic ledger which is implemented as a computer-based decentralised, distributed system made up of blocks which in turn are made up of transactions. Each transaction is a data structure that encodes the transfer of control of a 30 digital asset between participants in the blockchain system, and includes at least one input and at least one output. Each block contains a hash of the previous block so that blocks become chained together to create a permanent, unalterable record of all transactions which have been written to the blockchain since its inception. Transactions contain small programs known as scripts embedded into their inputs and outputs, which specify how and by whom the outputs of the transactions can be accessed. On the Bitcoin platform, these scripts are written using a stack-based scripting language.

**[0004]** In order for a transaction to be written to the blockchain, it must be "validated". Network nodes (miners) perform work to ensure that each transaction is valid, with invalid transactions rejected from the network. Software clients installed on the nodes perform this validation work on an unspent transaction (UTXO) by executing its locking and unlocking scripts. If execution of the locking and unlocking scripts evaluate to TRUE, the transaction is valid and the transaction is written to the blockchain. Thus, in order for a transaction to be written to the blockchain, it must be i) validated by the first node that receives the transaction—if the transaction is validated, the node relays it to the other nodes in the network; and ii) added to a new block built by a miner; and iii) mined, i.e. added to the public ledger of past transactions.

**[0005]** Although blockchain technology is most widely known for the use of cryptocurrency implementation, digital entrepreneurs have begun exploring the use of both the cryptographic security system Bitcoin is based on and the data that can be stored on the Blockchain to implement new systems. It would be highly advantageous if the blockchain could be used for tasks and processes which are not limited to the realm of cryptocurrency. Such solutions would be able to harness the benefits of the blockchain (e.g. a permanent, tamper proof records of events, distributed processing etc) while being more versatile in their applications.

**[0006]** One such area of interest is the use of the blockchain for the storage, sharing, accessing and controlling of data among users. Today, this is achieved via the Internet, with servers hosting web sites and pages which users visit in order to access the desired data, typically with a search engine.

**[0007]** However, some observers have begun to envisage the use of the blockchain to address some of the disadvantages of the Internet, such as control of large amounts of data and content by centralised parties. See, for example, "Life After Google: The Fall of Big Data and the Rise of the Blockchain Economy", George Gilder, Gateway Editions, July 2018, ISBN-10: 9781621575764 and ISBN-13: 978-1621575764.

**[0008]** Thus, it is desirable to provide an arrangement enabling such data to be stored, processed, retrieved, searched and/or shared on the blockchain, utilising the distributed, unalterable and permanent nature of the blockchain. Such an improved solution has now been devised. Embodiments of the present disclosure provide, at least, alternative, efficient and secure techniques for implementing a blockchain solution and for storing, processing, searching and/or retrieving data thereon or therefrom. Embodiments also provide, at least, an alternative, blockchain-implemented technical infrastructure for storing, processing, retrieving, transferring, searching and/or sharing data between computing nodes. Because the invention enables the blockchain network to be used in a new way and for the provision of an improved and technical result, the invention provides an improved blockchain-implemented network.

**[0009]** Embodiments also provide solutions for the secure control of access to digital resources over a technically different and improved computing platform which comprises a blockchain and a blockchain protocol.

**[0010]** The invention is defined in the appended claims

**[0011]** In accordance with the invention there may be provided a computer-implemented method. The method may be a method of enabling access to data stored on a blockchain, the method comprising:

- **[0012]** generating at least one first blockchain transaction of a blockchain, wherein said first data is stored in at least one said first blockchain transaction and is inaccessible to a first participant in the absence of first access data; and
- **[0013]** generating a second blockchain transaction of said blockchain, wherein said second blockchain transaction contains means for causing the first access data to be made available to the first participant.

**[0014]** This provides the advantage of enhancing the security of the process by ensuring that the first data becomes accessible if the terms of the unlocking script of the second transaction are fulfilled, thereby enabling an arrangement in which either two transactions can be executed or none.

**[0015]** At least one said first blockchain transaction may have at least one respective input being a respective output of a previous block chain transaction, wherein said output is redeemable by means of a respective private key of a cryptography system, and said first data may be encrypted by means of said cryptography system.

**[0016]** This provides the advantage of reducing the code base of the transactions, thereby reducing the amount of storage space required.

**[0017]** Access to said first data by said first participant may enable said first participant to store further data in at least one third blockchain transaction related to at least one said first blockchain transaction.

**[0018]** This provides the advantage of enabling efficient use of storage space for data on the blockchain.

**[0019]** The second blockchain transaction may include second data representing a quantity of crypto-currency.

**[0020]** The second blockchain transaction may include second data representing at least one token.

**[0021]** This provides the advantage of enabling conditions to be applied, for example timed access or expiry, to the token, thereby increasing flexibility of the method.

**[0022]** At least one said token may be at least one of a plurality of tokens generated by means of repeated application of a one-way function.

**[0023]** For example, tokens can be generated from an initial token and repeated application of a generation algorithm in the form of a one-way function, such as a hash function, enabling generation of subsequent tokens from the initial token. This provides the advantage that less data storage space is required.

**[0024]** The method may further comprise generating at least one fourth blockchain transaction for refunding at least part of said second data in the event of non-redemption of at least one output of the second blockchain transaction.

**[0025]** This provides the advantage of preventing the second data from becoming irretrievable in the event of non-redemption of the first blockchain transaction.

**[0026]** At least one said fourth blockchain transaction may have at least one output which is only redeemable after a predetermined period.

**[0027]** At least one said fourth blockchain transaction may have at least one output which is only redeemable after a predetermined number of blocks.

**[0028]** At least one said first blockchain transaction may has at least one output which is redeemable by means of a script containing data making said first access data available to said first participant.

**[0029]** The script may contain data enabling an input of a one-way function to be determined.

**[0030]** At least one said one-way function may be a Hash function.

**[0031]** The script may contain data enabling a private key of a public-private key pair to be determined by the first participant.

**[0032]** The method may further comprise securely sharing a first secret between said first participant and a second participant to enable data making said first access data available to said first participant to be determined.

**[0033]** For example, the shared secret may be used to mask an ephemeral key or a Hash pre-image.

**[0034]** The invention also provides a system, comprising:

**[0035]** a processor; and

**[0036]** memory including executable instructions that, as a result of execution by the processor, causes the system to perform any embodiment of the computer-implemented method described herein.

**[0037]** The invention also provides a non-transitory computer-readable storage medium having stored thereon executable instructions that, as a result of being executed by a processor of a computer system, cause the computer system to at least perform an embodiment of the computer-implemented method described herein.

**[0038]** These and other aspects of the present invention will be apparent from and elucidated with reference to, the embodiment described herein. An embodiment of the present invention will now be described, by way of example only, and with reference to the accompanying drawings, in which:

**[0039]** FIG. **1** shows a blockchain transaction embodying the present invention in which data is stored in a plurality of outputs;

**[0040]** FIG. **2** shows a blockchain transaction embodying the present invention in which data is stored in an input;

**[0041]** FIG. **3** shows a series of blockchain transactions embodying the present invention in which data is stored over outputs of a plurality of blockchain transactions;

**[0042]** FIG. **4** shows a blockchain transaction embodying the present invention transferring cryptocurrency payment to allow access to data by means of an atomic swap;

**[0043]** FIG. **5** shows a blockchain transaction embodying the present invention for redeeming the payment of the transaction of FIG. **4**;

**[0044]** FIG. **6** shows secret values kept by participants in a blockchain transaction embodying the present invention issuing tokens to allow access to data by means of an atomic swap;

**[0045]** FIGS. **7** and **8** show blockchain transactions embodying the present invention for issuing tokens to allow access to data by means of an atomic swap;

**[0046]** FIGS. **9** and **10** show blockchain transactions embodying the present invention for redeeming tokens issued by means of the transactions of FIGS. **7** and **8**;

**[0047]** FIGS. **11** and **12** show blockchain transactions for accessing the secrets exchanged by the transactions of FIGS. **9** and **10**;

**[0048]** FIG. **13** provides an illustration of the Metanet graph structure in accordance with an embodiment of the present invention.

**[0049]** FIG. **14** shows an illustration of a Metanet graph tree for the domain 'bobsblog' including MURL search paths in accordance with an embodiment of the present invention.

**[0050]** FIG. **15** shows a schematic for an illustrative embodiment of a browser-wallet in accordance with one example of the invention, and how its core functions can be split across different components of the application.

**[0051]** FIG. **16** provides a diagram illustrating how searching for content can be performed within the infrastructure of an embodiment of the invention.

**[0052]** FIG. **17** shows an illustrative interaction between local- and global-full copy peers in accordance with an embodiment of the invention.

**[0053]** FIG. **18** shows a Metanet a tree (or graph) for use in reference to the illustrative use case described below.

**[0054]** FIG. **19** shows a flow chart which illustrates the process embodied by the illustrative use case provided below.

**[0055]** FIG. **20** is a schematic diagram illustrates a computing environment in which various embodiments can be implemented.

**[0056]** The term "bitcoin" is used herein for convenience only, and is intended to include all cryptocurrency/blockchain protocols including, but not limited to, all variations that are derived from the Bitcoin protocol as well as any alternative protocols for other blockchains. In the remainder of this document, the protocol determining operation of the embodiments of the present invention will be referred to as the "Metanet protocol". The terms "content" and "data" may be used interchangeably herein to refer to data that is stored in a blockchain transaction in accordance with embodiments of the present invention. Herein, "sharing" may include providing to a node or user, sending, communicating, transmitting or providing access to the portion of data. The term "processing" may be interpreted as meaning any activity relating to the transaction or its associated data, including generating, transmitting, validating, accessing, searching for, sharing submitting to a blockchain network, and/or identifying.

## Overview

**[0057]** As stated above, there is a recognised need for an improved and/or alternative infrastructure for storing, writing, accessing and reviewing data between and by computing nodes. It would be advantageous to use the benefits which are inherent with blockchain technology (e.g. immutable records, cryptographically enforced control and access, built-in payment mechanism, ability to publicly inspect the ledger, distributed architecture etc). However, the construction of a "blockchain implemented internet" is challenging from a number of technical perspectives.

**[0058]** These challenges may include, but are not limited to: how to locate a particular portion of data within the network; how to secure and control access to the data so that only authorised parties may gain access; how to transfer the data from one party to another in a per-to-peer manner; how to arrange the data so that it can be logically associated yet stored in different locations within the network and how to subsequently combine it from different locations to provide a collective and augmented result; how to provide and/or store data in a hierarchical fashion; how to allow users and parties with different computing platforms access to the desired data; how to store, provide and share data across a (potentially global) computing network without reliance on or the need for large storage servers and centralised data controllers, and how to improve the efficiency of such data-related activities on the network.

**[0059]** The present invention provides such an improved solution in a manner which, in some ways, is analogous to the internet but achieves its results in an entirely different way, using an entirely different platform of hardware and software components from that known in the prior art. In accordance with embodiments of the present invention, the servers which store internet/web data and provide it to end users are replaced by blockchain transactions residing on the blockchain network. In order to achieve this, several innovations have had to be devised. These are described in the following sections.

**[0060]** Inserting Data into the Blockchain "Metanet" Referring to FIG. **1**, a blockchain transaction embodying the present invention is shown in which first data to be stored on the blockchain is stored in one or more first outputs of the transaction, and second data representing attributes of the first data is stored in one or more second outputs of the transaction. One or more first parts <Content 1> of the first data are stored in a spendable output of the transaction. Data <Attribute 1> and <Attribute 2>, representing respective attributes of the first data, together with a flag indicating that data is being stored according to the Metanet protocol, is stored in a second, unspendable output of the transaction. The term "unspendable" is used to indicate that at least one first and/or second output of the transaction may include a script opcode (OP RETURN) for marking an output as invalid for subsequent use as an input to a subsequent transaction.

**[0061]** It is advantageous to store the content and attributes parts of the data separately in separate outputs of the transaction.

**[0062]** FIG. **2** shows a blockchain transaction embodying the present invention in which first data <Content 1> to be stored on the blockchain is stored in an input of the transaction. The Metanet flag, and the attributes data <Attribute 1> and <Attribute 2> are stored in an unspendable output of the transaction, in a manner similar to the arrangement shown in FIG. **1**.

**[0063]** Data Insertion

**[0064]** Data Insertion Methods

**[0065]** It is desired to be able to insert the following data into the blockchain

**[0066]** a) Metanet Flag

**[0067]** b) Attributes

**[0068]** c) Content

**[0069]** The content is data to be stored on the blockchain, the Metanet Flag is a 4-byte prefix that acts as the identifier for any data pertaining to the Metanet protocol, while the attributes contain indexing, permissioning and encoding information about the content. This could include, but is not limited to, data type, encryption and/or compression schemes. Such attributes are also often referred to as metadata. Use of this term in the present document will be avoided in order to avoid confusion with transaction metadata.

**[0070]** The following techniques can be used to embed this data within a Bitcoin script:

**[0071]** 1. OP_RETURN—In this method all data (attributes and content) are placed after OP_RETURN in the locking script of a provably unspendable transaction output.

**[0072]** An example of an output script using this operator is:

**[0073]** UTXO0: OP_RETURN<Metanet Flag><attributes><content>

**[0074]** 2. OP_RETURN with OP_DROP—In this case the OP_RETURN contains the attributes, whilst the content is stored before an OP_DROP in a spendable transaction script (either locking or unlocking). The content can be split into multiple data packets within the transaction input and outputs. However, it is advantageous to insert data into transaction outputs as it is only output scripts that may be signed in the Bitcoin protocol. If the data is inserted into the transaction input, OP_MOD can be used as a checksum on the data

to ensure its validity in place of miner validation. For example, one could perform a 32-bit OP_MOD operation and check that it is equal to a pre-calculated value.

**[0075]** In this case, the attributes may contain information about how the content data packets are recombined. In addition, providing the hash of the recombined data packets H(content1+content2) as an attribute enables verification that the recommended recombination scheme has been used.

**[0076]** A transaction that implements the second data insertion method is shown in FIG. **1**. For simplicity this transaction only includes content inserted in its outputs, that is signed by its single input. Content inserted into additional inputs would also be possible using OP_DROP statements using this method as shown in FIG. **2**.

**[0077]** If the content is very large it may be advantageous to split it over multiple transactions. Such an arrangement is shown in FIG. **3**. FIG. **3** shows a pair of blockchain transactions embodying the present invention in which first data <Content> to be stored on the blockchain is split into two chunks <Content chunk 1> and <Content chunk 2> which can subsequently be recombined as <Content>=Content chunk 1>||<Content chunk 2>, where the operator '||' concatenates the two chunks of content data. This concatenation operator may be replaced by any desired bitwise or similar piecewise binary operator. The two chunks <Content chunk 1> and <Content chunk 2> are then stored in respective spendable outputs of the separate blockchain transactions, while data relating to the attributes of the content data is stored in respective unspendable outputs of the blockchain transactions. Again, the attributes can contain information about the recombination scheme. For example, the content may be raw data, an executable program or a HTML webpage. Additionally, content1 may include a pointer to the location on the blockchain of content2, which functions in the same way as an embedded HTML link within a webpage.

**[0078]** It should be noted that both transactions take the same public key P (and ECDSA signature) as input, such that <Content chunk 1> and <Content chunk 2> can be related by the same public key P despite being stored in different transactions with TxID1 and TxID2 respectively.

**[0079]** Exploiting the Role of Miner Validation

**[0080]** Here the transaction validation process performed by miners is used to gain an advantage when storing this data. This is because all data in transaction outputs will be signed by the owner of a public key P in at least one transaction input (if the SIGHASH|ALL flag is present) and this signature will be checked in the transaction validation process that all miners perform.

**[0081]** This ensures

**[0082]** Data integrity—If the data is corrupted the CHECKSIG operation will fail.

**[0083]** Data authenticity—The owner of P has provably witnessed and signed the data.

**[0084]** This is particularly advantageous for content that is split over multiple transactions as the input signature of P provides a provable link between the split components of the data, as described above with reference to the arrangement is shown in FIG. **3**.

**[0085]** Rabin Signature

**[0086]** Another way to ensure data authenticity is to use Rabin signatures, which can be used to sign the data itself rather than the whole message. This can be advantageous as the signer does not need to sign every individual transaction in which the data appears, and the signature can be re-used in multiple transactions.

**[0087]** Rabin signatures can be easily validated in script. These can be incorporated in case (2) above by inserting Rabin signature validation before the OP_DROP command, i.e.

**[0088]** <content1><Rabin Sig (content1)> FUNC_CHECKRABSIG OP_DROP<H($P_1$)>[CheckSig $P_1$]

**[0089]** It should be noted that this cannot be done in case (1) above since a script containing OP_RETURN fails anyway and so no validation can be achieved.

**[0090]** Specific Example of Use of Rabin Signature

INTRODUCTION

**[0091]** Digital signatures are a fundamental part of the Bitcoin protocol. They ensure that any Bitcoin transaction recorded on the blockchain has been authorised by the legitimate holder of the Bitcoin being sent. In a standard Bitcoin P2PKH transaction a transaction message is signed using the elliptic curve digital signature algorithm (ECDSA). However the ECDSA signature is generally applied to the whole transaction.

**[0092]** There are some use cases of the Bitcoin blockchain where a participant from outside the network may want to provide a signature for arbitrary data types which can then be used by network participants. By using Rabin digital signatures any piece of data can be signed —even if it originates from outside the Bitcoin blockchain and then placed in one or multiple transactions.

**[0093]** It will now be shown how data can be signed and verified directly in Bitcoin script by utilising the algebraic structures of the Rabin cryptosystem

**[0094]** Rabin Digital Signatures

**[0095]** The Rabin Digital Signature Algorithm

**[0096]** Background Mathematics

**[0097]** Definition—Integers mod p

**[0098]** The integers modulo p are defined as the set

$$\mathbb{Z}_p := \{1, 2, \ldots, p-1\}$$

**[0099]** Fermat's Little Theorem

**[0100]** Let p be a prime number. Then for any integer a the following applies

$$a^{p-1} \equiv 1 \bmod p$$

**[0101]** Euler's Criterion

**[0102]** Let p be a prime number. r is a quadratic residue mod p if and only if

$$r^{\frac{p-1}{2}} \equiv 1 \bmod p$$

**[0103]** Modular Square Roots (p=3 mod 4)

**[0104]** Let p be a prime number such that $p \equiv 3 \bmod 4$. Then for any integer r satisfying Euler's criterion, if a is an integer such that

$$a^2 \equiv r \bmod p$$

**[0105]** Then there exists a solution for a of the form

$$a \equiv \pm r^{\frac{p+1}{4}} \bmod p$$

**[0106]** Chinese Remainder Theorem

**[0107]** Given pairwise coprime positive integers $n_1, n_2, \ldots, n_k$ and arbitrary integers $a_1, a_2, \ldots, a_k$, the system of simultaneous congruences

$$x \equiv a_1 \bmod n_1$$

$$x \equiv a_2 \bmod n_2$$

$$\vdots$$

$$x \equiv a_k \bmod n_k$$

**[0108]** has a unique solution modulo $N=n_1 n_2 \ldots n_k$. As a special case of the Chinese Remainder Theorem, it can be shown that

$$x \equiv r \bmod n_1 \text{ and } x \equiv r \bmod n_2$$

if and only if

$$x \equiv r \bmod n_1 \cdot n_2$$

**[0109]** The Rabin Digital Signature Algorithm

**[0110]** The Rabin digital signature algorithm can be described as follows:

**[0111]** For any message m let H be a collision resistant hashing algorithm with k output bits. To generate the keys, choose prime numbers p and q, each with bit-length approximately k/2 such that $p \equiv 3 \bmod 4$, $q \equiv 3 \bmod 4$ and compute the product $n=p \cdot q$. The private keys are (p, q) and the public key is $n=p \cdot q$.

**[0112]** To sign the message, m, the signer chooses a padding U such that H(m||U) satisfies

$$H(m\|U)^{\frac{p-1}{2}} \equiv 1 \bmod p$$

$$H(m\|U)^{\frac{q-1}{2}} \equiv 1 \bmod q$$

**[0113]** The signature, S, is computed using the formula

$$S \equiv \left[\left(p^{q-2} \cdot H(m\|U)^{\frac{q+1}{4}} \bmod q\right) \cdot p + \left(q^{p-2} \cdot H(m\|U)^{\frac{p+1}{4}} \bmod p\right) \cdot q\right] \bmod n.$$

**[0114]** The signature for the message m is the pair (S, U). Verification can be simply done by checking that, for a given m, U and S

$$H(m\|U) \equiv S^2 \bmod n \qquad \text{(Equation 1).}$$

**[0115]** This is true if and only if there exists an integer λ in the range 0, . . . , n−1 such that

$$H(m\|U) + \lambda \cdot n = S^2 \qquad \text{(Equation 2).}$$

**[0116]** The factor λ may be safely included in the signature, to provide the combination (S,λ, U).

**[0117]** The advantageous features of the Rabin signature scheme are as follows:

**[0118]** a) Signature generation is computationally expensive, whereas verification of the signature is computationally easy.

**[0119]** b) The security of the signature relies only on the hardness of integer factorisation. As a result, Rabin signatures are existentially unforgeable (unlike RSA).

**[0120]** C) The hash function values H(m||U) must be of a similar magnitude to the public key n.

**[0121]** Verification in script is straightforward as it only requires squaring a given signature, performing a modular reduction and then checking that the result is equal to H(m||U).

**[0122]** Rabin Signature Proof

**[0123]** Let p, q be coprime and $n=p \cdot q$. By the Chinese Remainder Theorem it can be shown that

$$S^2 \equiv H(m\|U) \bmod n$$

If and only if

$$S^2 \equiv H(m\|U) \bmod p$$

$$S^2 \equiv H(m\|U) \bmod q$$

It can be shown that

$$S^2 \equiv H(m\|U) \bmod q$$

Using

$$S \equiv \left(\left(p^{q-2} H(m\|U)^{\frac{q+1}{4}} \bmod q\right) \cdot p + \left(q^{p-2} H(m\|U)^{\frac{p+1}{4}} \bmod p\right) \cdot q\right) \bmod q \equiv \left(\left(p^{q-2} H(m\|U)^{\frac{q+1}{4}} \bmod q\right) p\right) \bmod q \equiv p^{q-2} H(m\|U)^{\frac{q+1}{4}} \cdot p \bmod q \equiv (p^{q-1} \bmod q) H(m\|U)^{\frac{q+1}{4}} \bmod q \equiv H(m\|U)^{\frac{q+1}{4}} \bmod q$$

**[0124]** Therefore

$$S^2 \equiv H(m\|U)^{\frac{q+1}{2}} \equiv H(m\|U)^{\frac{q-1}{2}} \cdot H(m\|U) \equiv H(m\|U) \bmod q$$

**[0125]** Where it has been assumed that H(m||U) satisfies Euler's criterion. By a similar calculation one can also show that

$$S^2 \equiv H(m\|U) \bmod p$$

**[0126]** Rabin Signatures in Bitcoin

**[0127]** Signature Verification in Script

**[0128]** A small number of arithmetic and stack manipulation opcodes are required to verify a Rabin signature. Consider a redeem script of the form

**[0129]** OP_DUP OP_HASH160<$H_{160}$(n)> OP_EQUALVERIFY OP_MUL OP_SWAP OP_2 OP_ROLL OP_CAT FUNC_HASH3072 OP_ADD OP_SWAP OP_DUP OP_MUL OP_EQUAL

**[0130]** where n is the public key of the signer. This will evaluate to TRUE if and only if is provided with the input

**[0131]** <S><U><m><λ><n>

**[0132]** where m is an arbitrary message, and (S, λ, U) is a valid Rabin signature. Alternatively, if the Rabin signature is checked using equation 1 above the redeem script is given by

**[0133]** OP_DUP OP_HASH160<$H_{160}$(n)> OP_DUP OP_TOALTSTACK OP_SWAP<roll index> OP_ROLL

OP_CAT FUNC_HASH3072 OP_SWAP OP_MOD OP_SWAP OP_DUP OP_MUL OP_FROMALTSTACK OP_MOD OP_EQUAL

**[0134]** In this case the script will evaluate to TRUE if and only if is provided with the input

**[0135]** <S><U><m><n>

**[0136]** In both redeem scripts, use has been made of the 3072-bit hash projection function 'FUNC_HASH3072'. For a given message/padding concatenation the FUNC_HASH3072 hash projection is generated using the script

**[0137]** OP_SHA256 {OP_2 OP_SPLIT OP_SWAP OP_SHA256 OP_SWAP}(×11)

**[0138]** OP_SHA256 OP_SWAP OP_SHA256 {OP_CAT} (×11)

**[0139]** Compression of Data

**[0140]** Internet data consists of JavaScript and common file types such as text files (SML, HTML, etc.), video files (MPEG, M-JPEG etc.), image files (GIF, JPEG etc.) and audio files (AU, WAV, etc.), for example as described in greater detail at https://www.doc.c.ac.uk/~nd/surprise_97/oournal/vol1/mmp/#text. Using the above data insertion techniques, these different data types can also be embedded on the blockchain. Larger file sizes can be compressed using one of several existing coding schemes before embedding it on the blockchain. Lossless data compression algorithms such as Run-length and Huffman encoding can be used in several applications, including ZIP files, executable programs, text documents and source code.

**[0141]** Many different algorithms exist depending on the specific input data. Apple Lossless and Adaptive Transform Acoustic Coding can be used to compress audio files, PNG and TIFF for the compression of Graphics files, while movie files can be compressed using one of many lossless video codecs. Any compression of the data content can be indicated using a flag within the attributes. For example, the flag for the LZW lossless coding scheme in the attributes would be <LZW>.

**[0142]** Encryption and Paid Decryption

**[0143]** Encryption of Data

**[0144]** The owner of the content may choose to protect the content before embedding it on the Blockchain. This ensures that the content cannot be viewed without acquiring the necessary permissions.

**[0145]** There are many well-established techniques for the encryption of data (either plaintext or other data types). These can be categorised as asymmetric encryption or symmetric encryption.

**[0146]** Elliptic Curve Cryptography (ECC) is asymmetric as it relies on a public-private key pair. It is one of the most secure cryptosystems and is typically used in cryptocurrencies such as bitcoin. For ECC cryptography the Koblitz algorithm can be used to encrypt data.

**[0147]** In symmetric schemes a single key is used to both encrypt and decrypt the data. The Advanced Encryption Standard (AES) algorithm is considered one of the most secure symmetric algorithms that is seeded by such a secret, for example as described in greater detail in C. Paar and J. Pelzl, Chapter 4 in "Understanding Cryptography," Springer-Verlag Berlin Heidelberg $2^{nd}$ Ed., 2010, pp. 87-118.

**[0148]** When encrypting data stored on a blockchain there are advantages in using the same cryptosystems as the underlying blockchain. In bitcoin this is the secp256k1 conventions for ECC key pairs in asymmetric cryptography, and the SHA-256 hash function in symmetric cryptography. These advantages are:

**[0149]** The security level of the encryption is the same as the underlying system in which the data is stored on.

**[0150]** The software architecture needed for storing encrypted data will have a smaller codebase.

**[0151]** The management of keys in a wallet can be used both for transactions and encryption/decryption.

**[0152]** It is more efficient as the same keys can be used both for encryption and payment in the cryptocurrency, so fewer keys are required. This also reduces storage space.

**[0153]** Fewer communication channels may be needed to exchange/purchase the ability to decrypt data.

**[0154]** There is increased security as keys used for encryption and transactions are the same data structure and so targeted attacks for a specific type of key are mitigated.

**[0155]** Keys can be purchased using the underlying cryptocurrency.

**[0156]** For illustrative purposes, how the Koblitz algorithm can be used to encrypt data using ECC is described.

**[0157]** Koblitz Algorithm

**[0158]** Given a ECC key pair $P_1=S_1 \cdot G$ the Koblitz algorithm allows anyone to encrypt a message using the public key $P_1$ such that only someone who knows the corresponding private key $S_1$ may decrypt the message.

**[0159]** Suppose it is desired to encrypt the plaintext message 'hello world' using the Koblitz method. This is done character-by-character. The first character 'h' is encrypted and decrypted as follows.

**[0160]** 1. The character 'h' is mapped to a point on the secp256k1 curve. This is achieved by using the ASCII conventions to map a plaintext character to an 8-bit number. A point on the curve is then calculated by multiplying the base point G by this number. In the present example 'h' is mapped to 104 in ASCII, and the elliptic curve point is given by $P_m=104\ G$.

**[0161]** 2. The point $P_m$ is then encrypted using the public key $P_1$. This is achieved by choosing a random ephemeral key $k_0$ and calculating the pair of points $C_m=\{k_0 \cdot G, Q\}$ where $Q:=P_m+k_0 \cdot P_1$, which may then be broadcast.

**[0162]** 3. The owner of the private key $S_1$ may decrypt the original point by calculating $P_m=Q-S_1 \cdot k_0 \cdot G$. They may then recover the original ASCII number by trial-and-error or by means of a look up table to establish which number x corresponds to $P_m=x \cdot G$.

**[0163]** Using the Blockchain to Purchase Permissions

**[0164]** Storing data on a blockchain has the obvious advantage that a payment mechanism is built into the system. Payments can be used to purchase

**[0165]** Decryption data in order to view/use

**[0166]** Permission to insert data at a particular address

**[0167]** In both cases the buyer is using a cryptocurrency, for example Bitcoin, to purchase a secret that grants them permission to do something. This secret may be either a hash preimage or a private key.

**[0168]** An efficient and secure way to make such a purchase is to use an atomic swap. This keeps secure communication channels to a minimum and ensures that either the seller gets paid and the secret is revealed to the buyer, or neither event occurs.

**[0169]** Besides payments in cryptocurrencies, it can also be convenient to purchase a permission using an access token. This is a secret value (typically a hash preimage) that the buyer owns that they can use in order to make a purchase. Such tokens may be bought in bulk by the buyer ahead of time, and then activated at the time they would actually like to use the permission.

**[0170]** How atomic swaps are executed will now be described with reference to FIGS. **4** and **5**.

**[0171]** Atomic Swaps Using Hash Puzzles or Private Key Puzzles

**[0172]** Suppose Alice is the owner of the secret. This secret may be either a hash preimage of a known hash digest, or the private key of a known public key. Suppose Bob wishes to use Bitcoin to buy this secret from Alice. A mechanism known as an atomic swap is described that enables this transaction to occur. It is atomic in the sense that either Alice gets paid Bitcoin and the secret is revealed to Bob, or neither event occurs.

**[0173]** The method is as follows:

**[0174]** Alice owns a private key $S_A$ of a public/private key pair $P_A=S_A \cdot G$ and Bob owns a private key $S_B$ of a public/private key pair $P_B=S_B \cdot G$.

**[0175]** Alice owns a secret which is either the preimage X of a known hash digest H(X), or the private key $S_1$ of a known public key $P_1=S_1 \cdot G$.

**[0176]** They agree on a price in Bitcoin for Alice to sell the secret to Bob.

**[0177]** Prior to these, Bob must set up a transaction to send Alice the ephemeral key $k_0$ off-block so that she can calculate $r_0$, a component of a digital signature.

**[0178]** Referring now to FIG. **4**,

**[0179]** 1. Bob transfers Bitcoin to Alice that are locked with the following redeem script

**[0180]** R (written schematically):

**[0181]** For a hash preimage:

**[0182]** R=[Hash Puzzle H(X)] [CheckSig $P_A$]

**[0183]** This forces the preimage X to be exposed in the input of the redeem script.

**[0184]** For a private key:

**[0185]** R=[Private Key Puzzle $P_1$, $r_0$] [CheckSig $P_A$]

**[0186]** This forces the private key $S_1$ to be able to be calculated from the input to the redeem script. In this case Bob and Alice must agree on an ephemeral key $k_0$ that is used to construct $r_0$, where $(r_0, R_y)=k_0 \cdot G$.

**[0187]** 2. Since Alice knows her secret (either X or $S_1$) she can spend her funds on the Bitcoin blockchain by means of the transaction shown in FIG. **5**. This allows Bob to determine her secret.

**[0188]** As an optional security feature, Alice and Bob may use their public keys $P_A$, $P_B$ to establish a shared secret S known only to both parties. This could be achieved in a manner outlined in International Patent Publication no. WO 2017/145016. In this case S may be added to the preimage X in the hash puzzle in order for X not to be revealed publicly on the blockchain. Similarly, in the private key puzzle S may be used as the ephemeral key $k_0$ to ensure that only Alice or Bob is able to calculate the private key.

**[0189]** A time-locked refund can be introduced to the procedure to prevent Bob's funds being locked by Alice, should Alice decide not to spend her funds.

**[0190]** Purchase Using Tokens

**[0191]** Suppose that the same situation exists as described above, but instead paying in cryptocurrency for Alice's secret at its point of use Bob would like to redeem an access token—that he has bought ahead of time—in exchange for the secret.

**[0192]** The procedure that Alice and Bob must follow is similar to the case described in the previous section, but instead uses a sequence of similar atomic swaps. There are two phases of the process; token issuance and token redemption.

**[0193]** Phase 1: Token Issuance

**[0194]** The token issuance phase is effectively a one-time purchase of tokens by Bob. For example, we will consider the scenario where Alice has 10 distinct secrets $X_1, X_2, \ldots, X_{10}$ and Bob wishes to make a single purchase for 10 tokens $T_1, T_2, \ldots, T_{10}$ which each grant him access to a respective secret.

**[0195]** First Bob generates a set of 10 tokens from a secret seed value Y known only to him. These tokens are created by sequential hashing of the seed to form a hash chain, where each token is calculated as

$$T_i=H^{10-i}(Y) \text{ for } i\in\{1,2,\ldots,10\}.$$

**[0196]** Alice and Bob now have 10 secret values each, which can be revealed in hash puzzles for example, for the redemption of tokens. In order to issue these tokens however, they must also generate a secret initialising value $I_{Alice}$ and $I_{Bob}$ respectively. These are given as

$$I_{Alice}=k, k\in\mathbb{Z}_{256},$$

$$I_{Bob}=H^{10}(Y).$$

**[0197]** It should be noted that Alice's initialiser is simply a random integer with no specific meaning, but Bob's initialiser should be the hash of his first token $T_1=H^9(Y)$. Extending the chain of tokens to the initialiser in this way allows the token issuance to also define the token to be used later for successive redemptions. In total the secret values kept by each participant are shown in FIG. **6**.

**[0198]** Now Alice and Bob can agree to a price of 10 cryptocurrency units for the purchase of 10 tokens. The purchase of these tokens can happen in a number of ways, this is illustrated here using an atomic swap. The atomic swap is initiated by Alice and Bob broadcasting the transactions shown in FIGS. **7** and **8** respectively, in both of which the outputs require the solution to two hash puzzles and a valid signature.

**[0199]** Once both transactions appear in the blockchain Alice and Bob can share their share initialiser values $I_{Alice}$ and $I_{Bob}$ and complete the atomic swap for token issuance. As a result of this atomic swap Alice receives payment for the purchase of 10 tokens and both initialiser secrets are revealed. It should be noted that only Bob's secret $I_{Bob}=H^{10}(Y)$ is meaningful here because it will define the first hash puzzle to be solved [Hash Puzzle ($T_1$)], whose solution is the preimage $H^9(Y)$ of the initialiser $H^{10}(Y)$.

**[0200]** Phase 2: Token Redemption

**[0201]** At some point in the future Bob wants to redeem his first token $T_1=H^9(Y)$ and receive his first secret $X_1$, but it will be recalled that he has already paid for this secret by purchasing a valid token. The process of redeeming a token will take the form of another atomic swap, where the solutions to the locking hash puzzles are a token $T_i$ and corresponding secret $X_i$.

**[0202]** To redeem his token, Bob should broadcast the transaction shown in FIG. **9**, whose output is locked with the two hash puzzles. When Alice sees this transaction she

broadcasts her own similar transaction as shown in FIG. **10**, whose output is locked with the same two hash puzzles. The two participants can now exchange their secrets $T_1$ and $X_1$ and unlock the outputs to these transactions. Both parties can now redeem the nominal fee x by providing the correct unlocking script that also exposes both secrets. The transactions with these unlocking scripts are shown in FIGS. **11** and **12**.

**[0203]** The completion of this atomic swap for redeeming a token reveals Alice's first secret $X_1$ to Bob, reveals Bob's first token $T_1$ to Alice and has a net-zero exchange of cryptocurrency funds, given that the amount x is suitably large to encourage both parties to spend the locked outputs. Crucially, this also establishes that the next token Bob can use must be the solution $T_2$ to the hash puzzle [Hash Puzzle $H(T_2)$], where the target hash $H(T_2)=T_1$ has just been revealed to Alice. This process can be repeated recursively until Bob has used his final token $T_{10}=Y$.

**[0204]** Naming and Addressing

**[0205]** Node and Edge Structure

**[0206]** It has been explained above how data can be inserted into the blockchain by providing it within transactions. We now present a protocol for structuring these transactions in logical way that allows for addressing of nodes, permissions, and content version control. The structure of this distributed peer metanet is analogous to the existing internet.

**[0207]** It should be noted that this is a "tier-2" protocol that does not modify the protocol or consensus rules of the underlying blockchain.

**[0208]** The aim of the structure described here is to

**[0209]** (i) Associate related content in different transactions to enable searching, identification and access to the data

**[0210]** (ii) Allow identification of content using human-readable keyword searches, to improve speed, accuracy and efficiency of searching

**[0211]** (iii) Build and emulate server-like structures within a blockchain

**[0212]** Our approach is to structure data associated with the Metanet as a directed graph. The nodes and edges of this graph correspond to:

**[0213]** Node—A transaction associated with the Metanet protocol. A node stores content. (The terms "content" and "data" may be used interchangeably within this document).

**[0214]** A node is created by including an OP_RETURN that is immediately followed by <Metanet Flag>. Each node is assigned a public key $P_{node}$ The combination of public key and transaction ID uniquely specify the index of the node $ID_{node}$: $=H(P_{node}TxID_{node})$.

**[0215]** The hash function used should be consistent with the underlying blockchain protocol that the invention is to be used with e.g. SHA-256 or RIPEMD-160 for Bitcoin.

**[0216]** Edge—An association of a child node with a parent node.

**[0217]** An edge is created when a signature Sig $P_{parent}$ appears in the input of a Metanet transaction, and therefore only a parent can give permission to create an edge. All nodes may have at most one parent, and a parent node may have an arbitrary number of children. In the language of graph theory the indegree of each node is at most 1, and the outdegree of each node is arbitrary.

**[0218]** It should be noted that an edge is an aspect of the Metanet protocol and is not itself a transaction associated with the underlying blockchain.

**[0219]** A valid Metanet node (with parent) is given by a transaction of the following form:

| $TxID_{node}$ | |
|---|---|
| Input | Output |
| <Sig $P_{parent}$> <$P_{parent}$> | OP_RETURN <Metanet Flag> <$P_{node}$> <$TxID_{parent}$> |

**[0220]** This transaction contains all the information needed to specify the index of the node and its parent

$ID_{node}=H(P_{node}||TxID_{node})$,$ID_{parent}=H(P_{parent}||TxID\text{-}_{parent})$.

**[0221]** Moreover, since the signature of the parent node is required, only a parent can create an edge to a child. If the <$TxID_{parent}$> field is not present, or it does not point to a valid Metanet transaction, then the node is an orphan. It has no higher-level node by which it can be reached.

**[0222]** Additional attributes may be added to each node. These may include flags, names and keywords. These are discussed later in the this document.

**[0223]** As shown, the index of a node (transaction) can be broken down into

**[0224]** a) Public Key $P_{node}$, which we interpret as the address of the node

**[0225]** b) Transaction ID $TxID_{node}$, which we interpret as the version of the node

**[0226]** Two advantageous features arise from this structuring:

**[0227]** 1. Version control—If there are two nodes with the same public key, then we interpret the node with transaction ID with greatest proof of work as the latest version of that node. If the nodes are in different blocks, then this can be checked with the block height. For transactions in the same block, this is determined by the Topological Transaction Ordering Rule (TTOR).

**[0228]** 2. Permissioning—A child of a node can only be created if the owner of the public key $P_{node}$ signs the transaction input in the creation of a child node. Therefore $P_{node}$ not only represents the address of a node but also the permission to create a child node. This is intentionally analogous to a standard bitcoin transaction —a public key in not only an address but also the permission associated with that address.

**[0229]** Note that since the signature of the parent node appears in a UXTO unlocking script it is validated through the standard miner validation process at the point when the transaction is accepted to the network. This means that the permission to create a child node is validated by the bitcoin network itself.

**[0230]** It is worth noting that standard Internet Protocol (IP) addresses are unique only within a network at a certain point in time. On the other hand, the index of a node in the Metanet is unique for all time and there is no notion of

separate networks, which allows data to be permanently anchored to a single object $ID_{node}$.

**[0231]** The node and edge structure allow us to visualise the Metanet as a graph, as shown in FIG. **13**.

**[0232]** Domains, Naming and Locating Content within the Metanet

**[0233]** The hierarchy of the Metanet graph allows rich domain-like structure to emerge. We interpret an orphan node as a top-level domain (TLD), a child of an orphan node as a sub-domain, a grandchild as a sub-sub-domain etc., and a childless node as an end-point. See FIG. **13**.

**[0234]** The domain name is interpreted as $ID_{node}$. Each top-level domain in the Metanet may be thought of as a tree with the root being the orphan node and the leaves being the childless nodes. The Metanet itself is a global collection of trees which form a graph.

**[0235]** The Metanet protocol does not stipulate that any node contains content data, but leaf (childless) nodes represent the end of a directed path on the data tree, and thus will be used generally to store content data. However, content may be stored at any node in the tree. Protocol-specific flags, included in nodes as attributes, may be used to specify the role of nodes in a data tree (disk space, folders, files or permissioning changes).

**[0236]** Recall that the internet uses the Domain Name System (DNS) to associate human-readable names to Internet Protocol (IP) addresses. The DNS is in a sense decentralised, although in practice it is controlled by a small number of key players, such as governments and large corporations. Depending on your DNS provider the same name may take you to different addresses. This issue is inherent when mapping short human-readable names to computer generated numbers.

**[0237]** We assume that an equivalent distributed system exists that maps a human-readable top-level domain name to the decentralised index $ID_{root}$ of a root node. In other words, there exists a 1-1 function κ that maps human-readable names to Metanet root node indexes, for example

$$\kappa(\text{`bobsblog'})=ID_{bobsblog}(=H(P_{bobsblog}\|TxID_{bobsblog}))$$

**[0238]** The input to the left-hand-side is human-readable word, whereas the output on the right-hand-side is a hash digest, which will typically be a 256-bit data structure. Note that $P_{bobsblog}$ and $TxID_{bobsblog}$ are also not human readable in general. In the standard IP protocol this would be a map from www.bobsblog.com to the IP address of the corresponding domain within the network.

**[0239]** The map κ should be interpreted as a measure to ensure backwards-compatibility of the Metanet with the internet in replicating the human-readability of DNS-issued domain names, but the naming and addressing scheme that provides the structure of the Metanet is not explicitly dependent on this map.

**[0240]** Possible existing forms of the mapping function κ include the DNSLink system employed by Interplanetary File System (IPFS) or the OpenNIC service (https://www.openic.org). This mapping can be stored in an existing TXT record as part of the DNS. This is similar to a DNSLink in the IPFS—see https://docs.ipfs.io/guides/concepts/dnslink/. However, in general these sacrifice some element of decentralisation in order to provide a map that is 1-1—see https://hackernoon.com/ten-terrible-attempts-to-make-the-inter-planetary-file-system-human-friendly-e4e95df0c6fa

**[0241]** Vanity Addresses

**[0242]** The public key used as the address of a Metanet node is not a human-readable object. This can make searching, referencing and inputting activities error prone and slow for human users. However, it is possible to create human-recognisable public key addresses—vanity addresses $P_{vanity}$—which include a plaintext prefix that can be interpreted directly by a user. Vanity addresses are known in the prior art.

**[0243]** The difficulty in creating such an address depends on the character length of the desired prefix. This means that human-recognisable vanity addresses may be used as node addresses that rely only on the effort of the owner to create rather than on central issuance. For a given prefix there exist many distinct vanity addresses, due to the characters remaining in the suffix, and hence many node addresses can share a common prefix whilst still retaining uniqueness.

**[0244]** An example of a vanity address with a desirable prefix is

**[0245]** $P_{bobsblog}$: bobsblogHtKNngkdXEeobR76b53LETtpyT

**[0246]** Prefix: bobsblog

**[0247]** Suffix: HtKNngkdXEeobR76b53LETtpyT

**[0248]** The vanity address above may be used to sense check the map from the name 'bobsblog' to the node index $ID_{bobsblog}$ and to aid the searchability of Metanet nodes by address. Note that the prefix is not unique here but the entire address itself is a unique entity.

**[0249]** The combination of a chosen address $P_{vanity}$ with a TxID that together form $ID_{node}$ is also beneficial as it means there is no central issuer of domain names (TxIDs are generated by decentralised proof-of-work) and the names are recoverable from the blockchain itself. Advantageously, there are no longer the points of failure that exist within the internet DNS.

**[0250]** Since Metanet domains already provide a permissions system (the public key) there is no need to issue a certificate to prove ownership. The use of a blockchain for this purpose has already been explored in namecoin (https://namecoin.org/) for example. In accordance with the present invention, however, there is no need to use a separate blockchain for this function as everything is achieved within one blockchain.

**[0251]** This significantly reduces the amount of resources (hardware, processing resources and energy) required by the invention in comparison to the prior art. It also provides an entirely different architecture in terms of apparatus and arrangement of system components.

**[0252]** An advantage of this naming system is that a user is able to identify a top-level domain in the Metanet by a memorable word (for example a company name) rather than a hash digest. This also makes searching for the domain faster as it is quicker to search for a keyword rather than a hash digest. It also reduces input errors, thus providing an improved searching tool for blockchain-stored data.

**[0253]** Given that we have a map from a domain name to a node index we can build up a resource locator similar to that of a Uniform Resource Locator (URL) for the internet. We will call this a Metanet URL (MURL), and takes the form

MURL='mnp:'+'//domain name'+'/path'+'/file'.

**[0254]** Each of the components of a URL—protocol, domain name, path and file—have been mapped to the

structure of a MURL, making the object more user-intuitive and enabling it to be integrated with the existing structure of the internet.

**[0255]** This assumes that each node has a name associated with its public key (address) that is unique at the level within the domain tree. This name is always the right-most component of the MURL for a given node. If two nodes at the same level in the tree have the same name then they will have the same public key and so the latest version is taken.

**[0256]** The following table gives an analogy between the Metanet protocol and the Internet Protocol:

| Internet | Metanet |
|---|---|
| Website/file | Node |
| Owner | Public Key $P_{node}$ |
| IP Address (non-unique) | Node index (unique) $ID_{node} = H(P_{node}\|\|TxID_{node})$ |
| Domain structure | Node tree structure |
| Domain Name System | Map from root node name to node index |
| URL | MURL |
| http://www.bobsblog.com/path/file | mnp://bobsblog/path/file |

**[0257]** Searching the Metanet

**[0258]** We have defined an illustrative embodiment the Metanet graph structure such that each node has a unique index and may have a name attributed to it. This allows for content to be located using a MURL. In order to also enable quick search functionality, we allow for additional keywords to be attributed to a node.

**[0259]** The fixed attributes of a node are the index and index of parent node, and the optional attributes are the name and keywords.

```
Node attributes
{
    index:              H(P_node||TxID_node);
    index of parent:    H(P_parent||TxID_parent); (NULL if orphan)
    name:               'bobsblog';
    kwd1:               'travel';
    kwd2:               'barbados';
                .
                .
                .
}
```

**[0260]** In one example, a practical method for searching the Metanet may be to first use a block explorer to trawl through the blockchain and identify all transactions with the Metanet flag, check that they are valid Metanet nodes, and if so record their indexes and keyword in a database or other storage resource. This database can then be used to efficiently search for nodes with desired keywords. Once the index of the node(s) with the desired keywords is found its content can be retrieved from the block explorer and viewed.

**[0261]** By way of example, consider the $P_1$ branch of FIG. **14**, where the nodes corresponding to public keys $P_0$, $P_1$ and $P_{1,1}$ represent a home page, topic page and sub-topic page respectively. These nodes are given the names 'bobsblog', 'summer' and 'caribbean', and their attributes are shown below

**[0262]** Home page node $P_0$

```
MURL: mnp://bobsblog
{
        index:              H(P0||TxID0);
        index of parent:    NULL
        name:               'bobsblog';
        kwd1:               'travel';
        kwd2:               'barbados';
                .
                .
                .
}
Topic page node P1
MURL: mnp://bobsblog/summer
{
        index:              H(P1||TxID1);
        index of parent:    H(P0||TxID0);
        name:               'summer';
        kwd1:               'travel';
        kwd2:               'barbados';
                .
                .
                .
}
Sub-topic page node P1,1
MURL: mnp://bobsblog/summer/caribbean
{
        index:              H(P1,1||TxID1,1);
        index of parent:    H(P1||TxID1);
        name:               'caribbean';
        kwd1:               'travel';
        kwd2:               'barbados';
                .
                .
                .
}
```

**[0263]** In this example, the leaf nodes $P_{1,1,1}$ $P_{1,1,2}$ and $P_{1,1,3}$ are given the names 'beaches', 'nightlife' and 'food' respectively and are used to store separate blog posts. The full domain structure is shown on the diagram overleaf, including the MURL search path pertaining to each node in the tree.

**[0264]** We note that the Metanet can also incorporate a content addressable network (CAN) by storing a hash of the content stored by a node transaction as an additional attribute. This means Metanet nodes may also be indexed and searched for by content hash.

**[0265]** The naming and addressing methods described above provide numerous technical advantages over the prior art, including:

**[0266]** 1. Public key addresses—The system uses the same public-private key pairs as the blockchain to assign node addresses. This means that the same set of keys are used for both management of cryptocurrency funds and permissioning of content data. This provides an efficient and secure solution.

**[0267]** 2. Decentralised domains—The issuance of domain names is fully decentralised through the inclusion of the $TxID_{node}$, which can only be generated by proof of work. The domain names can also incorporate human-recognisable public keys $P_{vanity}$ (vanity addresses) which enable the fair distribution of desired domain public keys. Again, this solution provides enhanced efficiency and security.

[0268] 3. Graph structure—The naming and addressing architecture specifies a graph that can be constructed from the subset of blockchain data comprising Metanet nodes. This design maps the complexity of the internet to the blockchain using an ordered structure such that it fully replicates its functionality and scalability whilst remaining secure.

[0269] Browser-Wallet Application

[0270] Recall that in the Metanet protocol all data lives directly on the blockchain itself. In this section we present embodiments of an illustrative computer application, which we will refer to herein for convenience only as a "browser-wallet", that can efficiently access, display and interact with Metanet data stored on the blockchain.

[0271] We will begin with a discussion of the core components and functionalities of how the browser-wallet interfaces with the distributed peer internet, before providing a more detailed description in the remainder of this section.

Overview

[0272] Components

[0273] The browser-wallet is an application intended to allow an end-user to interact with the Metanet infrastructure on the blockchain. This application should allow explorative searches of the Metanet graph for specific content embedded in trees. Additionally, the browser-wallet will handle retrieval, decryption, recombination and caching (optional) of content.

[0274] The browser-wallet application will combine these elements with cryptocurrency payment mechanisms by supporting a native (or external) wallet. The browser-wallet will comprise the following core elements, combined into a single computer application.

[0275] Blockchain search engine—Support for a third-party search engine to query Metanet nodes by a variety of indexes including $ID_{node}$, node name, keywords, block height and TxID.

[0276] Display window—Software that unpacks content returned to the browser by a full-copy blockchain peer. This covers decryption, recombination, caching and redemption ofaccess tokens.

[0277] Cryptocurrency wallet—Dedicated key-management for currency of the blockchain. Can be native to the application or authorised to communicate and synchronise with external wallet (software or hardware). Able to write standard blockchain transactions as well as new Metanet node transactions. Can mediate on-chain purchase of access keys and access tokens.

[0278] Hierarchical, deterministic key management is leveraged for both cryptocurrency public keys and Metanet node addresses.

[0279] Access key/token wallet—Dedicated key-management for purchased access keys or tokens. Can receive purchased keys or tokens using cryptocurrency wallet but has no permissions over them. They may be hidden to the user to allow later expiry. This may be achieved through the use of a trusted execution environment. Timed-access can be secured by synchronising with the blockchain and querying current block height.

[0280] Functionality

[0281] The specification for the Metanet browser-wallet ensures the following functionalities of the application.

[0282] 1. Hierarchical key-management—The keys used for controlling funds and managing Metanet trees (graphs) utilise the same hierarchical deterministic key infrastructure, reducing the burden on users to maintain key records for their Metanet content.

[0283] 2. Pointing to an external cryptocurrency wallet—Ability to authorise and synchronise with an external (non-native to application) wallet allows for additional security by removing the browser-wallet as a point of failure.

[0284] The application can write blockchain transactions and require the signature of an external wallet that houses keys, delegating this responsibility to separate software or hardware.

[0285] 3. Searching of Metanet content—The browser-wallet can support and query a third-party search engine whose functions may comprise crawling, indexing, servicing and ranking Metanet node transaction data in a global database. A database of OP_RETURN transactions containing the Metanet protocol flag may be constructed. See BitDB 2.0—https://bitdb.network/.

[0286] The search engine can serve the browser-wallet with a node index, which allows data to be found.

[0287] 4. Reading and writing data to blockchain—In addition to using search engines and full-nodes to serve the browser with content, the support for a cryptocurrency wallet also allows for content to be written into the Metanet directly from the browser-wallet.

[0288] 5. Decompression and decryption of data—The browser-wallet handles decryption keys and can perform decompression on Metanet content in-situ.

[0289] 6. Caching node identities ($ID_{node}$)—Unique node identities can be cached locally for more efficient lookup and querying.

[0290] 7. Bypassing web-servers—Given a node index, the browser-wallet can query any full-copy member of the peer-to-peer (P2P) blockchain network for the content located at a node. Because the Metanet lives on-chain, any full-copy peer must have a local copy of the node and its content.

[0291] This means that the user's browser-wallet need only query a single peer, which can be done directly and without the need for an intermediary web-server. FIG. **15** shows the schematic for the browser-wallet and how its core functions are split across different components of the application.

[0292] Blockchain Search Engine

[0293] Search Engines—Existing Technologies

[0294] Search engines (SEs) as known in the prior art rely on powerful web crawlers to locate, index and rank web content according to the user queries. (The same underlying principles can be extended to a third-party Blockchain SE that crawls the Metanet).

[0295] SEs identify relevant HTML metatags and content through a search of the keywords in the query. The results of the crawl are subsequently indexed where any embedded images/videos/media files are analysed and catalogued. The most relevant results from the index are then ranked programmatically, taking into consideration the user's location, language and device.

**[0296]** A typical SE should have the following functionality:

**[0297]** 1. Crawling—Identify internet data and crawl through the relevant metadata, such as domain names, linked pages and related keywords. New internet content is discovered through existing content and also crawled for any relevant information.

**[0298]** 2. Indexing—Content data are analysed and catalogued. This information is stored in the database.

**[0299]** 3. Servicing and ranking—Content indexes are ranked in order of relevance to the users' queries.

**[0300]** Block Explorers

**[0301]** The closest blockchain analogue to an internet search engine (SE) is a blockchain explorer, sometimes referred to as a 'block explorer' or 'blockchain browser'. Blockchain explorers are web applications which enable user-friendly queries of a blockchain, at a high level, and function similarly to web browsers but are connected to a blockchain rather than the internet. See https://en.bitcoin.it/wiki/Bilock_chain_browser.

**[0302]** In most cases, these explorers allow blocks (indexed by hash of the block header), transactions (indexed by TxID), addresses and unspent transaction outputs (UTXOs) to be taken as inputs and searched for. Many explorers also offer their own application programming interfaces (APIs) for retrieving raw transaction and block data. See https://biockexplorer.com/api-ref.

**[0303]** Block explorers, while varying in their capabilities, are generally useful for cataloguing transactions and displaying their basic information—such as transacted currency values, confirmations and histories of coins and addresses—in a form that is easy for users to digest. Many explorers, such as Bitcoin.com https://explorer.bitcoin.com/bch and Blockchain.com https://www.blockchain.com/explorer also allow the individual input and locking scripts for transactions to be viewed, although there are inconsistencies between how these and more advanced sites like Blockchair https://blockchair.com/choose to provide this information.

**[0304]** Recently there have been many extensions of basic blockchain explorers used to run web applications based on blockchain data. These applications, such as Memo.cash https://memo.cash/protocol and Matter https://www.mttr.app/home act like block explorers that catalogue and organise blockchain transactions that contain specific protocol identifiers, as well as displaying data encoded within those particular transactions.

**[0305]** However, there are two important issues with using blockchain explorers that are addressed by embodiments of the present invention:

**[0306]** 1. Universality—There is currently no industry-wide standard for browsing content data that is stored in transactions. Content data refers to any data that does not pertain to the protocol used for creating and securing the underlying blockchain.

**[0307]** 2. Keyword searching—Content data stored in transactions needs to be retrievable by human-readable keywords. This is not generally a function of current block explorers as they are used to query the protocol-based properties of transactions, such as block height, TxID and addresses rather than taking keywords as search inputs. (However, some e.g. Blockchair can search for words if they are directly included in the script of the transaction).

**[0308]** Importantly, the powerful naming and addressing structure of the present invention, as discussed above, facilitates and enables the construction of a more sophisticated blockchain explorer than known in the art.

**[0309]** Proposed Metanet Search Engine

**[0310]** The browser-wallet application communicates with a third-party search engine for discovery of node identities ($ID_{node}$). It is envisaged that such a third-party may provide a powerful and versatile service that replicates the capabilities of existing internet search engines.

**[0311]** A Metanet search engine third-party maintains a database of all Metanet transactions mined into the blockchain identifiable by the Metanet protocol flag. This database can catalogue all Metanet nodes by a range indexes including $ID_{node}$, node names, key words, TxID and block height.

**[0312]** There already exist services, such as Bit DB https://bitdb.network/, which continuously synchronise with the blockchain and maintain transaction data in a standard database format. The browser-wallet offloads the responsibilities of crawling, indexing, servicing and ranking Metanet transactions to such a third-party and makes a connection to their services when locating content stored on the Metanet graph.

**[0313]** Efficiency savings may be made by having a database that is dedicated to Metanet data only. Unlike Bit DB this would not store the data associated with all transaction, only those containing the Metanet flag. Certain databases, such as non-relational databases like MongoDB, may be more efficient at storing the graph structure of the Metanet. This would allow for faster querying, lower storage space, and more efficiently associate related content within Metanet domains.

**[0314]** FIG. **16** shows how the browser-wallet interacts with a third-party search engine when a user searches for content within the Metanet infrastructure. Importantly, it should be noted that, in contrast with the Internet, no routing is necessary and thus the invention provides important advantages in respect of efficiency, speed, processing and required resources.

**[0315]** The process is as follows

**[0316]** 1. The end user inputs keywords into the browser-wallet search bar.

**[0317]** 2. The browser-wallet sends the keyword query to a third-party SE.

**[0318]** 3. The SE checks the keywords against its database and returns $ID_{node}$ for any Metanet nodes containing relevant content. The third-party can also return other indexes on each node to the user, as well as providing suggestions for related content.

**[0319]** 4. The browser-wallet uses the node identity and the domain name associated with it to construct a MURL.

**[0320]** 5. The browser-wallet requests the content belonging to the specified node from any network peer with a full copy of the blockchain.

**[0321]** 6. The network peer serves the browser-wallet with the requested content. Because the peer has a copy of the blockchain, they must also have a copy of the content and so only one request is made, and it is never forwarded to other network peers.

**[0322]** It is emphasised that the third-party SE only has the responsibility of indexing and maintaining records of the attributes of Metanet nodes, while the raw content data

stored on the nodes is instead stored by network peers (e.g. full-copy peers, miners, archives) with a full copy of the blockchain.

**[0323]** Content Display—Metanet Browser

**[0324]** The browser-wallet application emulates the same front-end capabilities that any typical web-browser should provide. These functions include, but are not limited to:

**[0325]** 1. Searching—Provide access to a search engine (SE) for locating content.

**[0326]** 2. Retrieval—Communicate with a server to facilitate the transfer of content using a known protocol e.g. Hypertext Transfer Protocol (HTTP).

**[0327]** 3. Interpreting—Parsing raw code (e.g. in JavaScript) and executing.

**[0328]** 4. Rendering—Efficient display of parsed content to be viewed by the end user.

**[0329]** 5. User interface (UI)—Provide an intuitive interface for a user to interact with content, including action buttons and mechanism for user-input.

**[0330]** 6. Storage—Local temporary storage capacity for caching internet content, cookies etc., to improve repeated access to content.

**[0331]** In certain embodiments, the software component of the browser-wallet application responsible for acting as a web-browser is able to perform the above functions on Metanet content embedded in the blockchain that is both searchable (using SEs) and retrievable (from peers) using their attributes.

**[0332]** Recombination, De-Compression and Decryption

**[0333]** In accordance with certain embodiments of the invention, the web-browser software component of the browser-wallet application is able to handle all operations that need to be performed on given Metanet content. There are many such operations that need to be performed in general, but we assume that at least the following are executed by the application using the Metanet protocol and infrastructure.

**[0334]** Recombination—In the case that Metanet content needs to be split up and inserted into multiple separate node transactions, the application will request the content from all relevant nodes and reconstitute the original content. The ordering and structure of the splintered content can be encoded using additional flags in each node's attributes.

**[0335]** Decompression—Where content data is stored on the blockchain in a compressed form, it should include a flag to indicate to the browser-wallet which standard compression scheme has been used. The application will decompress content according to this flag.

**[0336]** Decryption—Where content is encrypted a flag should be used to signify the encryption scheme. The application will locate a key from its decryption key wallet (as discussed below) and decrypt content data for use according to the encryption scheme used.

**[0337]** In performing these operations on content data, flags can be used to signify to the browser-wallet that a given operation needs to be performed. This generalises to any other operation, for which a suitable Coperation_flag> can be included as part of the attributes of nodes to which the operation applies.

**[0338]** Caching

**[0339]** The caching of local files and cookies is a common and important function of typical web-browsers. The browser-wallet application also uses local storage in a similar way in order to optionally keep a record of $ID_{node}$ and other node attributes that pertain to content of interest. This allows more efficient lookup and retrieval of content from frequently-visited Metanet nodes.

**[0340]** The Metanet solves the problem inherent with caching internet data that it is mutable and can be changed or censored by web-browsing software depending on the provider. When caching Metanet data a user can always easily verify that it is in the same state as when originally included as an immutable record on the blockchain.

**[0341]** Cryptocurrency Wallet

**[0342]** Hierarchical Deterministic Key Management

**[0343]** Deterministic keys Dk are private keys initialized from a single "seed" key (See Andreas M. Antonopoulos, Chapter 5 in "Mastering Bitcoin" O'Reilly 2$^{nd}$ Ed., 2017, pp. 93-98). The seed is a randomly generated number that acts as a master key. A hash function can be used to combine the seed with other data, such as an index number or "chain code" (see HD Wallets—BIP-32/BIP-44), to derive deterministic keys. These keys are related to one another and are fully recoverable with the seed key. The seed also permits the easy import/export of a wallet between different wallet implementations, giving an additional degree of freedom if the user wishes to use an external wallet in conjunction with the Metanet browser-wallet.

**[0344]** A hierarchical deterministic (HD) wallet is a well known derivation method of deterministic keys. In HD wallets, a parent key generates a sequence of children keys, which in turn derive a sequence of grandchildren keys, and so on. This tree-like structure is a powerful mechanism for managing several keys.

**[0345]** In a preferred embodiment, a HD wallet can be incorporated into the Metanet architecture illustrated in FIG. **16**. Advantages of using HD wallets include:

**[0346]** 1. Structure Additional organisational meaning can be expressed using different branches of subkeys for different purposes. For example, a user could dedicate different branches (and their corresponding subkeys) to different types of data.

**[0347]** 2. Security Users can create a sequence of public keys without the corresponding private keys, making HD wallets functional in a receive-only capacity and suitable for use on insecure servers. Also, since fewer secrets need to be stored there is a lower risk of exposure.

**[0348]** 3. Recovery If keys are lost/corrupted then they can be recovered from the seed key.

**[0349]** Native (Internal) and External Wallet Support

**[0350]** Advantageously, embodiments of the invention can directly merge the functionality of traditional web-browsers with one or more cryptocurrency wallets. This is fundamentally how the Metanet combines the payment for "internet" content with its delivery to the end user.

**[0351]** To achieve this, embodiments of the browser-wallet may have a dedicated, in-built software component that operates as a cryptocurrency wallet. This wallet is native the application itself and can be used to manage cryptocurrency private keys and authorise transactions as payment for Metanet content within the browser-wallet itself.

**[0352]** This means that the browser component of the application can prompt the wallet component to authorise a payment that is required—by purchasing a decryption key, access token or otherwise—to view Metanet content. The

application does not need to invoke an external third party to process the payment, and thus the Metanet content of interest is consumed by the application and paid for in-situ.

**[0353]** External Wallets

**[0354]** The same advantages and functionality can be achieved by embodiments of the application if a user wishes to instead manage or keep their cryptocurrency private keys on an external wallet (software or hardware) or even use multiple wallets. This may be performed in lieu of, or in conjunction with, the application's native wallet.

**[0355]** In such embodiments, the application establishes a link or pairing with an external wallet(s), and synchronises with it, but does not store private keys in the browser-wallet itself. Instead, when the browser component prompts a payment to be made for content, the application requests an authorisation by digital signature from the external wallet of choice. This authorisation is made by the user and the browser-wallet can broadcast the transaction and view the paid content.

**[0356]** Reading and Writing Metanet Transactions

**[0357]** An intrinsic advantage of the Metanet is that it uses the same data-structure—the blockchain—to record both payments and content data. This means that software wallets can be used to write content data to the Metanet infrastructure in addition to creating transactions that are purely based on the exchange of cryptocurrency.

**[0358]** The native wallet built-in to the application is able to write transactions to the blockchain that are more complex than typical simplified payment verification (SPV) clients—see https://bitcoin.org/en/glossary/simplified-payment-verification. The wallet allows users to choose to write a Metanet node transaction to the blockchain directly from the application by selecting content data, from their computer, to be embedded in the blockchain.

**[0359]** As the browser-wallet application has a user interface (UI) it allows for the wallet component to create and broadcast transactions that include content data that has be constructed either in the browser-component or on the user's computer beforehand. This capability would be far more difficult to achieve for a purpose-built wallet to handle on its own.

**[0360]** Access Key/Token Wallet

**[0361]** Recall from above that, built in to the Metanet protocol, is the ability to encrypt content using an ECC keypair or AES symmetric key, and the ability purchase the corresponding decryption key or token. We will refer to these as access keys or access tokens.

**[0362]** Such keys/tokens grant the user permission to view or edit content (either single use or multi-instance use) and play a distinct role from keys that control the users cryptocurrency wallet (although the same key may be used for both purposes if desired). For this reason, it is advantageous to introduce a new wallet, separate from the application's native cryptocurrency wallet, that is used for storing and managing access keys and tokens.

**[0363]** One can also introduce the notion of timed access to Metanet content by allowing access keys/tokens to be burned after a certain time period. This can be achieved by requiring that access keys/tokens are stored in a Trusted Execution Environment (TEE) and are not directly accessible to the user.

**[0364]** The fact that access keys/tokens may be "burned" is also a motivating factor for not storing them in the cryptocurrency wallet to ensure there is no risk of cryptocurrency private keys being burned.

**[0365]** In a similar way to the cryptocurrency wallet, decryption keys and access tokens can be stored and managed deterministically to facilitate efficient handling and deployment. Decryption keys (e.g. ECC private keys) can be generated and recovered by subsequent additions to a master key, while access tokens can be reconstructed using a hash chain that is seeded by some initial token.

**[0366]** It is important to make the distinction here that the cryptocurrency wallet handles the deterministic key generation for key pairs that are used in transacting with other users and creating new Metanet nodes, whereas a key/token wallet(s) handles keys and tokens that have been purchased by the cryptocurrency wallet.

**[0367]** Block Height Permissioning

**[0368]** Timelocks can be included in the bitcoin script language to enable block height permissioning. The op_code OP_CHECKLOCKTIMEVERIFY (CLTV) sets the block height at which a transaction output (UTXO) is permissible for spending.

**[0369]** The advantages of block height permissioning are twofold:

**[0370]** 1. Version control—In the Metanet Protocol, the newest version of a node can be identified from the node at the greatest block height. The browser-wallet can be set up to only display the most recent version of a file by block height, allowing proof-of-work version control.

**[0371]** 2. Timed access—The Browser-wallet application can periodically burn the decryption keys atomically purchased by the user. This ensures that viewers can only access content data during the time period for which they have paid. Cloning of the decryption keys can be prevented by storing them in a trusted execution environment (TEE). Moreover, the atomic swap involves the purchase of a deterministic key Dk (for decryption of thecontent data). Although this deterministic key is publicly visible, the TEE can be used to sign for the combination of Dk and a securely enclaved private key.

**[0372]** The browser-wallet can be arranged to synchronise with the current state of the blockchain in order to use block height as its own proxy for time, rather than relying on any external clock or third-party time oracle.

**[0373]** Bypassing Web-Servers

**[0374]** The invention allows a new mechanism for a browser (client) and a web server to communicate and exchange information over the distributed peer internet that bypasses domain name system (DNS) servers and typical network routing procedures. See http://www.theshulers.com/whitepapers/internet_whitepaper/. The invention provides anew network architecture, from which the browser-wallet application can be served content, comprising peers that maintain a full-copy of the blockchain.

**[0375]** Local Full-Copy Peers

**[0376]** Consider a system of local peers in each geographical area e.g. postcode, town, city. We assume that within this local network at least one peer maintains a full-copy of the blockchain, which we will refer to as a Local Full-Copy Peer (LFCP). For our purposes, an LFCP need only store the blockchain transactions that include the Metanet flag, but we do not limit them to this.

**[0377]** All users default to sending 'get' requests to LFCPs. As the peers maintain a complete and up-to-date copy of the entire blockchain, all requests can be served because any node ID that is queried will be available to the LFCP. Note that a Metanet search engine may also act as an LFCP if the SE is powerful and large enough to both store Metanet content and perform the main functions of a typical SE.

**[0378]** In the simplest case, every LFCP will have the same storage and disk space overheads as they will all need to be able to store the full blockchain (around 200 GB at the time of writing). The distinction between each LFCP is that they should scale their capability to respond to the demands of local requests from Metanet users. So, if each Metanet user in the world queries their closest LFCP by default, each LCFP should aim to scale in their operational capacity to meet their local demand. Densely-populated areas like cities will require LFCP operations comprising of many clustered servers, while sparser areas like small towns will require smaller LCFP operations.

**[0379]** It is important to note that the disk space requirement is universal, while the CPU requirement for each LFCP adapts to local network demand. This is an example of an adaptable network, such as Freenet—see https://blockstack.org/papers/.

**[0380]** One advantage of such a system is that users only need to make a single (local) connection to their LFCP when retrieving the content associated with a given $ID_{node}$. There is no need for the LFCP to forward the request to other peers as they are guaranteed to be able to serve the required content themselves.

**[0381]** The Metanet provides many advantages over the internet—such as decentralisation and deduplication—that are similar to other peer-to-peer (P2P) file-sharing services like IFPS. However, the Metanet improves on these existing P2P models by ensuring immutability and, crucially, removing the need to flood the network with requests for given content.

**[0382]** The Metanet infrastructure is also robust to the compromise of any one LFCP by employing a network of these peers. This means that if a LFCP is disabled then the end user simply defaults to using their next nearest LFCP. This can be made more efficient if LFCPs communicate with each other to indicate which nearby peers are below or above capacity in terms of requests at any given time. This can allow users to send their request to the most appropriate peer and establish dynamic equilibria in the request distribution between nearby LFCPs.

**[0383]** Global Full-Copy Peers

**[0384]** We now consider a scenario when the universal disk space requirement becomes too great for smaller peers, which may happen as the Metanet portion of the blockchain scales and grows with adoption.

**[0385]** In this case smaller LFCPs should use their disk space capacity to store Metanet node transactions based on a popularity system (there are existing techniques for ranking content by request volume and nature). This means that LFCPs now tailor both their CPU (for request handling ability) and their storage allocation (for content-serving ability) to suit their local geographic demands both in volume and nature of content.

**[0386]** In order to address the fact that LFCP are now unable to store all Metanet transaction content the concept of a Global Full-Copy Peer (GFCP) can be utilised. A GFCP is a full-copy peer that has the following properties:

**[0387]** 1. A GFCP grows its disk space capacity in order to maintain a full-copy of the blockchain at all times.

**[0388]** 2. A GFCP has substantial CPU resources such that it can handle significantly more requests than an LFCP. A global full-copy peer should be able to handle a sudden increase in demand should many LFCPs become compromised.

**[0389]** There are two main functions of a GFCP. First, to act as a failsafe for user requests of Metanet content in times of overflow requests from LFCPs. Second, GFCPs act as archive peers to store all Metanet content mined historically, which ensures that any Metanet node content can be accessed even if many LFCPs omit some content from their local storage provisions.

**[0390]** Global Data Banks

**[0391]** The concept of a GFCP is a powerful one and illustrates how the overall architecture of the Metanet provides a solution to an existing problem; that of creating all-encompassing, global data banks.

**[0392]** Before now, it has not been possible to safely construct a universal and globally-accessible data bank because it would need to be maintained by a central authority. This central authority injects both a point of failure and of trust into the system. Crucially, if one organisation is relied upon to store and maintain all internet data then we need to trust the facts that they are doing so correctly and legitimately, without corrupting the information.

**[0393]** With the Metanet infrastructure, we have effectively removed both of these problems, those of trust and of centrality, from the concept of a global data centre. Now, such a GFCP can be created because they are only relied upon to provide the required disk space for storage and not to verify and authenticate the information to be stored.

**[0394]** With the Metanet, the process of verifying what is stored is done by miners and hence universal, global data banks can be trusted because they cannot corrupt blockchain information. The GFCP does not need to be trusted and need only provide storage. The fact that all GFCPs can store the same information, that is always verifiable and provable against the blockchain itself, means that it can be replicated across many such GFCPs. This means me have also solved the issue of having a single point of failure by enabling many global data banks to exist in parallel and provably store the same information.

**[0395]** FIG. **17** shows a system of two LFCPs and one GFCP and illustrates how each peer can support the other in a network that is robust to the compromise of individual peers.

**[0396]** The aspects of the invention which can be implemented in embodiments of the browser-wallet application described above provide numerous distinguishing features and advantages over the prior art, including but not limited to:

**[0397]** 1. Deterministic keys—Hierarchical deterministic key management for both cryptocurrency and Metanet addresses is performed in the same wallet component of the application. This allows organisation of keys with multiple functions that reduces their storage requirements and enables key recovery.

**[0398]** 2. Payment mechanism—The application allows consumers to pay merchants directly without needing to point to another application or third-party payment

service that would conventionally authenticate and provide trust. This allows the purchase and delivery of digital content to take place via the same blockchain platform. The application inherits the advantages of bitcoin payments including low value exchanges or more complex transactions involving multiple parties.

**[0399]** 3. Bypassing web-servers—The application facilitates the bypassing of traditional web-servers, which would conventionally process a high volume of traffic, requests and routing. This is because the application need only request content from a single LFCP, which is guaranteed to serve the user without needing to forward the request to other LFCPs. This reduces the overall traffic volume as well as the completion time of each request.

**[0400]** 4. Timed access—The application facilitates timed access to content by synchronising with the blockchain and using it to enforce access permissions based on its current state. This removes the need for third party services to monitor a users' privileges over time while protecting the rights of the original owner.

**[0401]** Use Case—Decentralised App Store (Swapp Store)

**[0402]** The first use case presented here (for illustrative purposes only) for the Metanet architecture is for the decentralised payment for and distribution of applications (apps).

**[0403]** We consider a scenario in which an app developer Alice and a consumer Bob wish to transact with each other. This transaction will take the form of an atomic swap in which money is exchanged for a secret key that grants Bob access to the application data. The encrypted application data is already public as part of a Metanet node transaction.

**[0404]** An atomically-swapped application is known as a Swapp. A third-party platform (Swapp store) may be used for cataloguing and advertising applications that exist on the Metanet, but the payment for and transfer of the access key to a user such as Bob does not need to involve any third party and can be done directly between merchant and consumer.

**[0405]** The following section details a process that may be used for buying and selling Swapps from the creation of an app by Alice to its deployment by Bob. Throughout the process, Alice and Bob will use their respective browser-wallets to interact with the Metanet.

**[0406]** Publishing

**[0407]** 1. Alice writes an application. The data that constitutes this application is the content denoted by <App>. She also encrypts it sa<e(App)> using a secret key $S_k$.

**[0408]** 2. Alice creates a node transaction, $ID_{AliceApp}$ to set up her first Metanet domain (tree). She generates 1AliceAppHtKNngkdXEeobR76b53LETtpy ($P_{Alice-App}$) to be used as the node address.

**[0409]** 3. Alice then creates children of the first node to form a tree that corresponds to a Metanet library of her applications. Alice's tree domain is shown in FIG. **18**.

**[0410]** One of the leaf nodes on this tree is the node corresponding to her application <App> with index $ID_{App}$. In this node, Alice inserts the encrypted application data <e(App)> into the input script (scriptSig) of the node. The app data is encrypted using the Koblitz method using a secret key $s_k$.

**[0411]** This node transaction is shown below.

| $TxID_{App}$ | |
|---|---|
| Input | Output |
| <Sig $P_{Puzzle}$> <$P_{Puzzle}$> <e(App)> OP_DROP | OP_RETURN <Metanet Flag> <$P_{App}$> <$TxID_{Puzzle}$> |

**[0412]** 4. Alice broadcasts $ID_{AliceApp}$, $P_{AliceApp}$ and domain name 'AliceApp' publicly. This can be via social media, an internet website or by using a third-party Metanet website.

**[0413]** Purchasing

**[0414]** 1. Bob wants to download a puzzle game and sees Alice's app listed on a Metanet website (Swapp store) that he views on his browser-wallet.

**[0415]** 2. Bob then communicates with Alice, using information from the website, and sets up an atomic swap. The swap is designed such that Bob will pay an agreed price in Bitcoin to Alice and Alice will reveal the secret key $s_k$ or neither event occurs.

**[0416]** 3. The atomic swap is completed and Bob's browser-wallet stores the secret key $s_k$ in its access key/token wallet.

**[0417]** Deployment

**[0418]** Bob now has the key $s_k$ that will allow him to decrypt the application data Alice published previously. In order to download the app and deploy it Bob does the following.

**[0419]** 1. Bob uses a Metanet search engine (SE) to find the MURL associated with the encrypted app data <e(App)>. He uses the keywords 'AliceApp' and 'App' as input to the search bar in his browser-wallet. The third-party SE resolves the query and returns the following MURL:

**[0420]** mnp://aliceapp/games/puzzle/app

**[0421]** This locator corresponds to the unique Metanet node $ID_{App}$ which includes the encrypted app data in its input script.

**[0422]** 2. Bob's browser wallet receives this MURL and sends a request to the nearest appropriate LFCP. This peer serves Bob with the requested data <e(App)>.

**[0423]** 3. The browser-wallet processes the data according to the attributes of $ID_{App}$. This includes using the secret key $s_k$ to decrypt the application data and process <App>.

**[0424]** 4. Bob downloads the application <App> from his browser to his computer. Bob can now deploy the application locally without having to re-purchase access.

**[0425]** FIG. **19** illustrates the entire process outlined in the above illustrative use case. The flow chart shows two action branches, Alice's (starting on the left hand side) and Bob's (starting on the right hand side). The branch corresponding to Alice shows the initial publishing phase and Bob's shows the phase of setting up the purchase via atomic swap.

**[0426]** On Bob's branch, he broadcasts the following transaction $TxID_{Bob}$ as the atomic swap set-up phase:

| $TxID_{Bob}$ | | | |
|---|---|---|---|
| Inputs | | Outputs | |
| Value | Script | Value | Script |
| x BCH | <Sig $P_B$> <$P_B$> | x BCH | [Private Key Puzzle $P_k$, $r_0$] [CheckSig $P_A$] |

**[0427]** In this transaction, the output is locked with a private key puzzle that requires the secret decryption key $s_k$ to be revealed to Bob in order for Alice to spend.

**[0428]** Alice and Bob's branches in this diagram converge at the point where Alice successfully completes the atomic swap transaction. This is achieved when Alice broadcasts the transaction $TxID_{Alice}$:

| $TxID_{Alice}$ | | | |
|---|---|---|---|
| Inputs | | Outputs | |
| Value | Script | Value | Script |
| x Bitcoin | <Sig $P_A$> <$P_A$> <Sig $P_1$, $r_0$> <$P_1$> | x Bitcoin | [CheckSig $P_A$] |

**[0429]** As soon as this transaction is broadcast, Alice and Bob's action branches diverge once more. Alice receives payment of x Bitcoin while Bob receives the secret decryption key $s_k$ and is able to retrieve and decrypt Alice's application from the Metanet.

**[0430]** Turning now to FIG. **20**, there is provided an illustrative, simplified block diagram of a computing device **2600** that may be used to practice at least one embodiment of the present disclosure. In various embodiments, the computing device **2600** may be used to implement any of the systems illustrated and described above. For example, the computing device **2600** may be configured for use as a data server, a web server, a portable computing device, a personal computer, or any electronic computing device. As shown in FIG. **20**, the computing device **2600** may include one or more processors with one or more levels of cache memory and a memory controller (collectively labelled **2602**) that can be configured to communicate with a storage subsystem **2606** that includes main memory **2608** and persistent storage **2610**. The main memory **2608** can include dynamic random-access memory (DRAM) **2618** and read-only memory (ROM) **2620** as shown. The storage subsystem **2606** and the cache memory **2602** and may be used for storage of information, such as details associated with transactions and blocks as described in the present disclosure. The processor(s) **2602** may be utilized to provide the steps or functionality of any embodiment as described in the present disclosure.

**[0431]** The processor(s) **2602** can also communicate with one or more user interface input devices **2612**, one or more user interface output devices **2614**, and a network interface subsystem **2616**.

**[0432]** A bus subsystem **2604** may provide a mechanism for enabling the various components and subsystems of computing device **2600** to communicate with each other as intended. Although the bus subsystem **2604** is shown schematically as a single bus, alternative embodiments of the bus subsystem may utilize multiple busses.

**[0433]** The network interface subsystem **2616** may provide an interface to other computing devices and networks. The network interface subsystem **2616** may serve as an interface for receiving data from, and transmitting data to, other systems from the computing device **2600**. For example, the network interface subsystem **2616** may enable a data technician to connect the device to a network such that the data technician may be able to transmit data to the device and receive data from the device while in a remote location, such as a data centre.

**[0434]** The user interface input devices **2612** may include one or more user input devices such as a keyboard; pointing devices such as an integrated mouse, trackball, touchpad, or graphics tablet; a scanner; a barcode scanner; a touch screen incorporated into the display; audio input devices such as voice recognition systems, microphones; and other types of input devices. In general, use of the term "input device" is intended to include all possible types of devices and mechanisms for inputting information to the computing device **2600**.

**[0435]** The one or more user interface output devices **2614** may include a display subsystem, a printer, or non-visual displays such as audio output devices, etc. The display subsystem may be a cathode ray tube (CRT), a flat-panel device such as a liquid crystal display (LCD), light emitting diode (LED) display, or a projection or other display device. In general, use of the term "output device" is intended to include all possible types of devices and mechanisms for outputting information from the computing device **2600**. The one or more user interface output devices **2614** may be used, for example, to present user interfaces to facilitate user interaction with applications performing processes described and variations therein, when such interaction may be appropriate.

**[0436]** The storage subsystem **2606** may provide a computer-readable storage medium for storing the basic programming and data constructs that may provide the functionality of at least one embodiment of the present disclosure. The applications (programs, code modules, instructions), when executed by one or more processors, may provide the functionality of one or more embodiments of the present disclosure, and may be stored in the storage subsystem **2606**. These application modules or instructions may be executed by the one or more processors **2602**. The storage subsystem **2606** may additionally provide a repository for storing data used in accordance with the present disclosure. For example, the main memory **2608** and cache memory **2602** can provide volatile storage for program and data. The persistent storage **2610** can provide persistent (non-volatile) storage for program and data and may include flash memory, one or more solid state drives, one or more magnetic hard disk drives, one or more floppy disk drives with associated removable media, one or more optical drives (e.g. CD-ROM or DVD or Blue-Ray) drive with associated removable media, and other like storage media. Such program and data can include programs for carrying out the steps of one or more embodiments as described in the present disclosure as well as data associated with transactions and blocks as described in the present disclosure.

**[0437]** The computing device **2600** may be of various types, including a portable computer device, tablet computer, a workstation, or any other device described below.

Additionally, the computing device **2600** may include another device that may be connected to the computing device **2600** through one or more ports (e.g., USB, a headphone jack, Lightning connector, etc.). The device that may be connected to the computing device **2600** may include a plurality of ports configured to accept fibre-optic connectors. Accordingly, this device may be configured to convert optical signals to electrical signals that may be transmitted through the port connecting the device to the computing device **2600** for processing. Due to the ever-changing nature of computers and networks, the description of the computing device **2600** depicted in FIG. **20** is intended only as a specific example for purposes of illustrating the preferred embodiment of the device. Many other configurations having more or fewer components than the system depicted in FIG. **20** are possible.

**[0438]** It should be noted that the above-mentioned embodiments illustrate rather than limit the invention, and that those skilled in the art will be capable of designing many alternative embodiments without departing from the scope of the invention as defined by the appended claims. In the claims, any reference signs placed in parentheses shall not be construed as limiting the claims. The word "comprising" and "comprises", and the like, does not exclude the presence of elements or steps other than those listed in any claim or the specification as a whole. In the present specification, "comprises" means "includes or consists of" and "comprising" means "including or consisting of". The singular reference of an element does not exclude the plural reference of such elements and vice-versa. The invention may be implemented by means of hardware comprising several distinct elements, and by means of a suitably programmed computer. In a device claim enumerating several means, several of these means may be embodied by one and the same item of hardware. The mere fact that certain measures are recited in mutually different dependent claims does not indicate that a combination of these measures cannot be used to advantage.

**1**. A method of enabling access to data stored on a blockchain, the method comprising:

generating at least one first blockchain transaction of a blockchain, wherein first data is stored in at least one said first blockchain transaction and is inaccessible to a first participant in the absence of first access data; and

generating a second blockchain transaction of said blockchain, wherein said second blockchain transaction contains means for causing the first access data to be made available to the first participant.

**2**. A method according to claim **1**, wherein at least one said first blockchain transaction has at least one respective input being a respective output of a previous blockchain transaction, wherein said output is redeemable by means of a respective private key of a cryptography system, and said first data is encrypted by means of said cryptography system.

**3**. A method according to claim **1**, wherein access to said first data by said first participant enables said first participant to store further data in at least one third blockchain transaction related to at least one said first blockchain transaction.

**4**. A method according to claim **1**, wherein the second blockchain transaction includes second data representing a quantity of crypto-currency.

**5**. A method according to claim **1**, wherein the second blockchain transaction includes second data representing at least one token.

**6**. A method according to claim **5**, wherein at least one said token is at least one of a plurality of tokens generated by means of repeated application of a one-way function.

**7**. A method according to claim **4**, further comprising generating at least one fourth blockchain transaction for refunding at least part of said second data in the event of non-redemption of at least one output of the second blockchain transaction.

**8**. A method according to claim **7**, wherein at least one said fourth blockchain transaction has at least one output which is only redeemable after a predetermined period.

**9**. A method according to claim **7**, wherein at least one said fourth blockchain transaction has at least one output which is only redeemable after a predetermined number of blocks.

**10**. A method according to claim **1**, wherein at least one said first blockchain transaction has at least one output which is redeemable by means of a script containing data making said first access data available to said first participant.

**11**. A method according to claim **10**, wherein the script contains data enabling an input of a one-way function to be determined.

**12**. A method according to claim **11**, wherein at least one said one-way function is a Hash function.

**13**. A method according to claim **10**, wherein the script contains data enabling a private key of a public-private key pair to be determined by the first participant.

**14**. A method according to claim **1**, further comprising securely sharing a first secret between said first participant and a second participant to enable data making said first access data available to said first participant to be determined.

**15**. A computer-implemented system comprising:

a processor; and

memory including executable instructions that, as a result of execution by the processor, causes the system to perform any embodiment of the method as claimed in claim **1**.

**16**. A non-transitory computer-readable storage medium having stored thereon executable instructions that, as a result of being executed by a processor of a computer system, cause the computer system to at least perform an embodiment of the method as claimed in claim **1**.

**17**. A computer-implemented system comprising:

a processor; and

memory including executable instructions that, as a result of execution by the processor, causes the system to perform any embodiment of the method as claimed in claim **2**.

**18**. A computer-implemented system comprising:

a processor; and

memory including executable instructions that, as a result of execution by the processor, causes the system to perform any embodiment of the method as claimed in claim **3**.

**19**. A non-transitory computer-readable storage medium having stored thereon executable instructions that, as a result of being executed by a processor of a computer system, cause the computer system to at least perform an embodiment of the method as claimed in claim **2**.

**20**. A non-transitory computer-readable storage medium having stored thereon executable instructions that, as a result of being executed by a processor of a computer system, cause the computer system to at least perform an embodiment of the method as claimed in claim **3**.

* * * * *