

(21) Application No: 2012873.2

(22) Date of Filing: 18.08.2020

(71) Applicant(s):
nChain Holdings Limited
Fitzgerald House, 44 Church Street, St.John's,
Antigua and Barbuda

(72) Inventor(s):
Michaela Pettit
Craig Steven Wright

(74) Agent and/or Address for Service:
Page White & Farrer
Bedford House, John Street, London, WC1N 2BF,
United Kingdom

(51) INT CL:
H04L 9/32 (2006.01) H04L 9/08 (2006.01)

(56) Documents Cited:
WO 2017/145048 A1 US 20110213982 A1
US 20110208970 A1

(58) Field of Search:
INT CL H04L

(54) Title of the Invention: Digital signatures
Abstract Title: Shared digital signatures

(57) A digital signature is shared amongst participants sharing a private key by a signing party obtaining a message (eg. part of a blockchain transaction) and hashing a first external data item (via eg. a random salt) in order to generate a first ephemeral private key share (eg. a Joint Verifiable Secret Share JVSS) from the hashed item. An ephemeral public key is generated from this private key, and a digital signature share generated from the original message, the shares of the private key, and the ephemeral public key. This signature share is then made available for generating a signature when a threshold of available signature shares is reached.

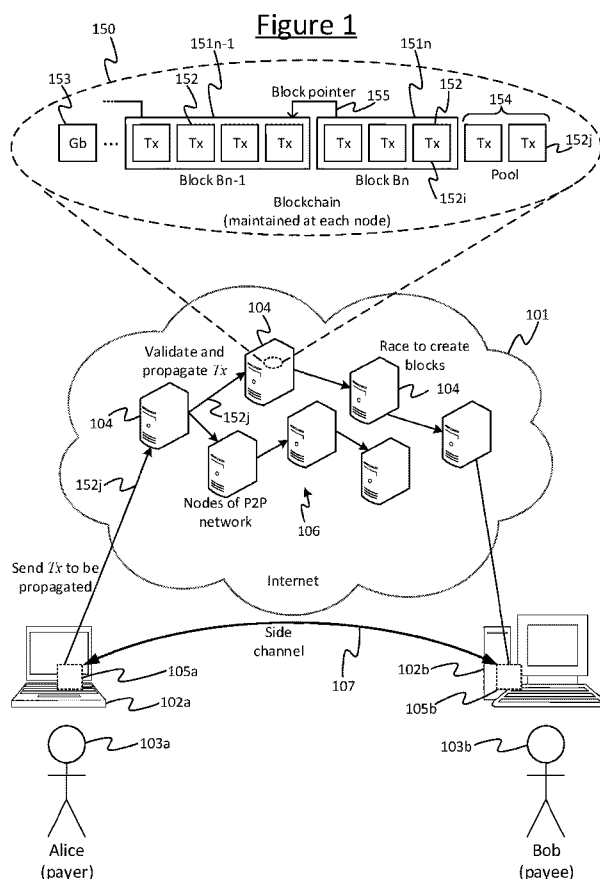


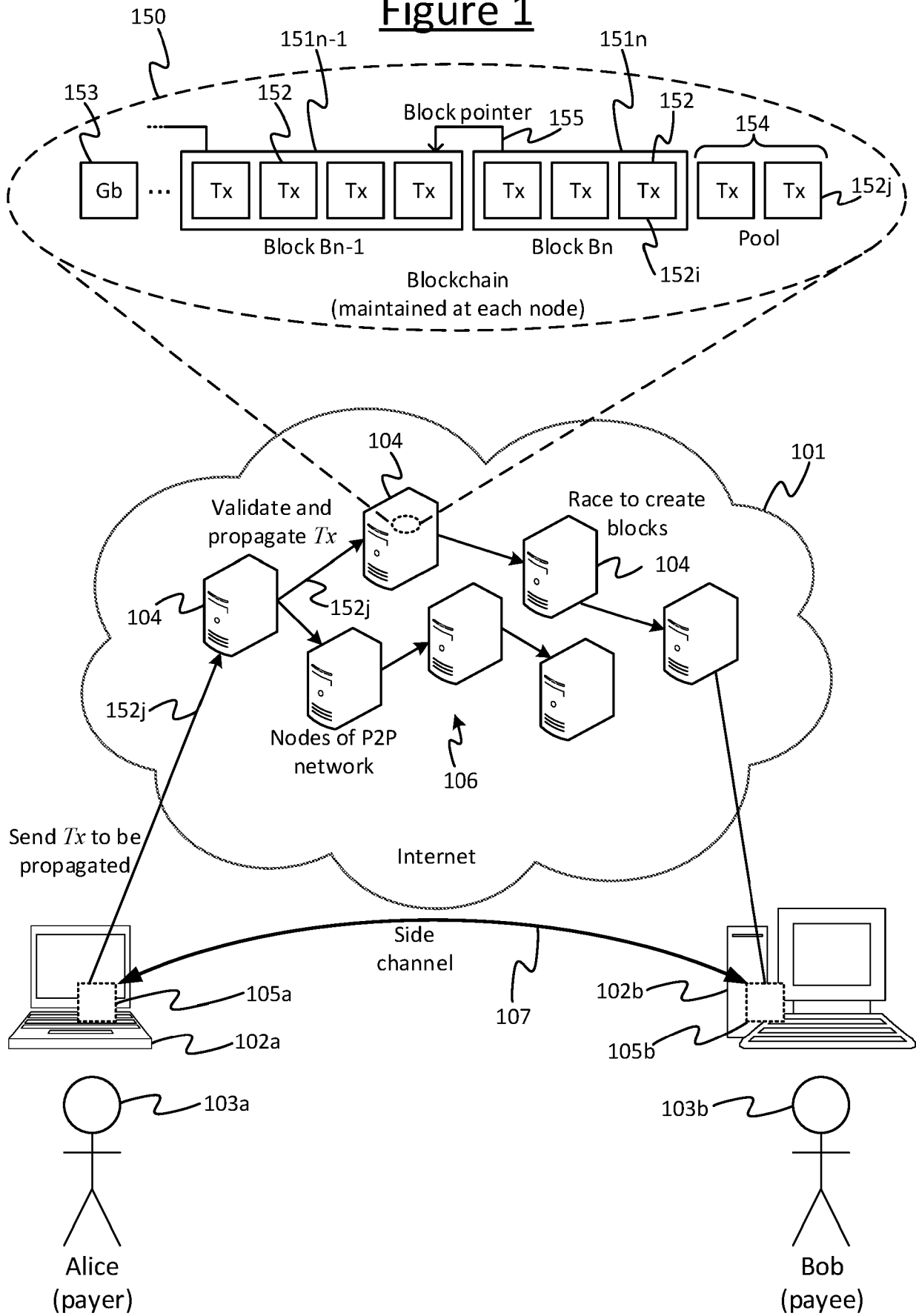
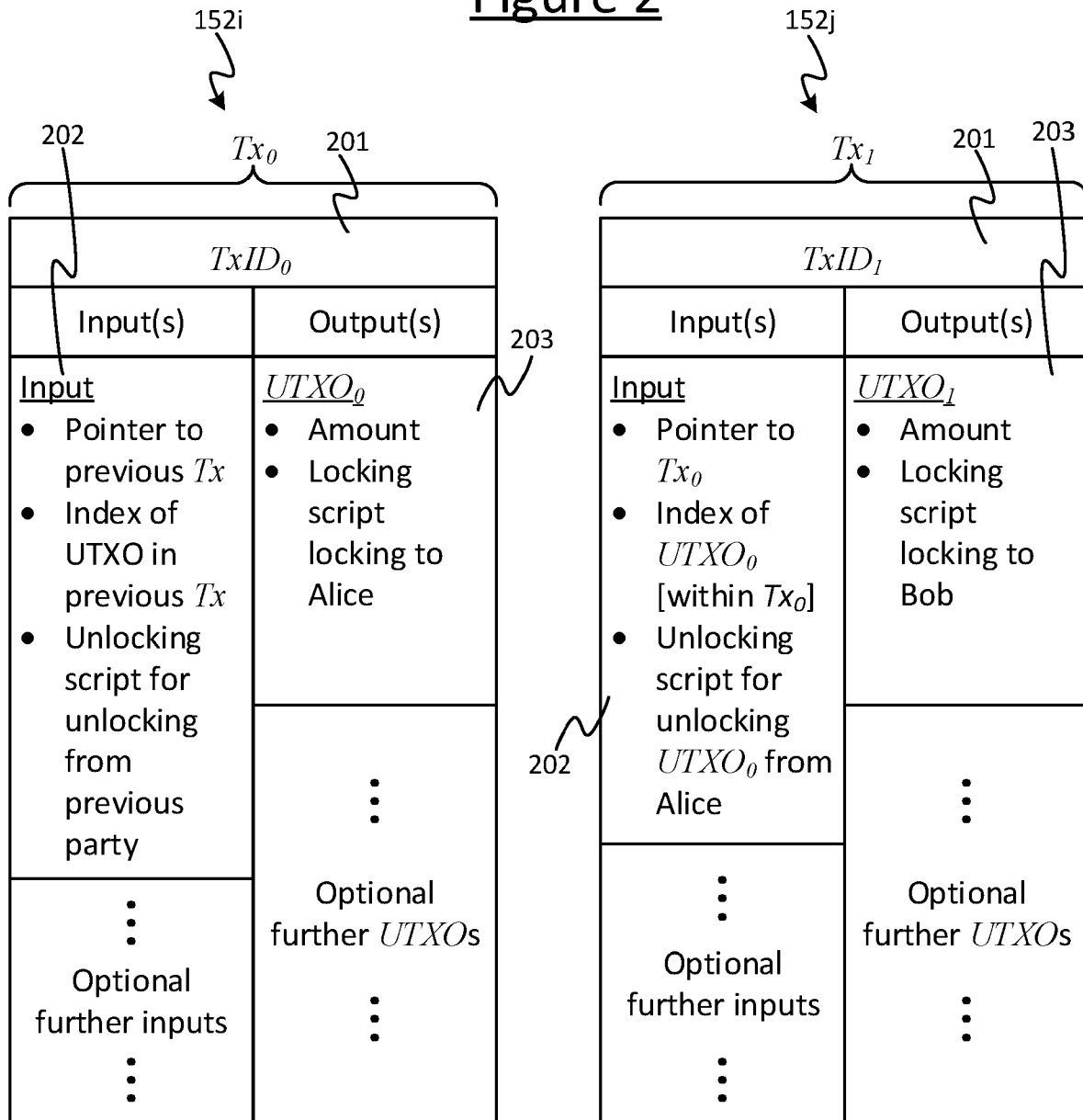
Figure 1

Figure 2

Transaction
from Alice to Bob

↓

Validated by running: Locking script from output 203 of Tx_0 , together with Alice's unlocking script from input 202 of Tx_1 . This checks that Alice's unlocking script in Tx_1 meets the condition(s) defined in the locking script of previous transaction Tx_0 .

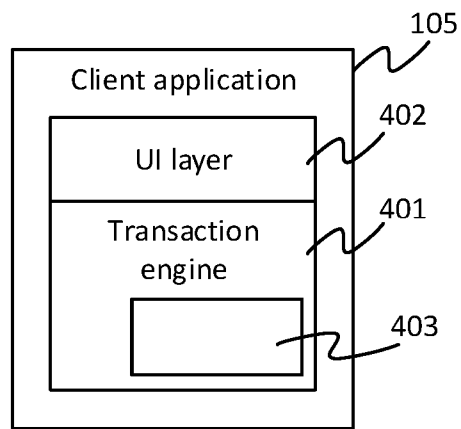
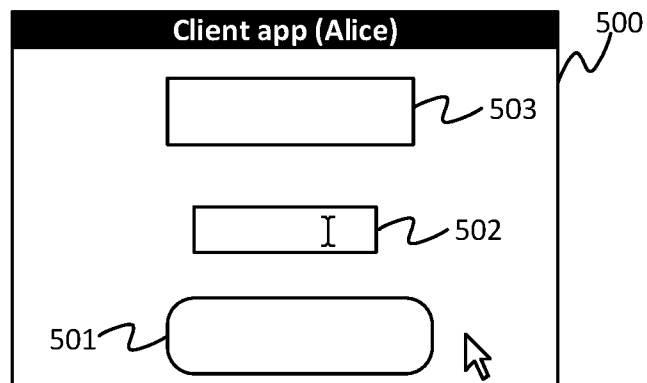
Figure 3AFigure 3B

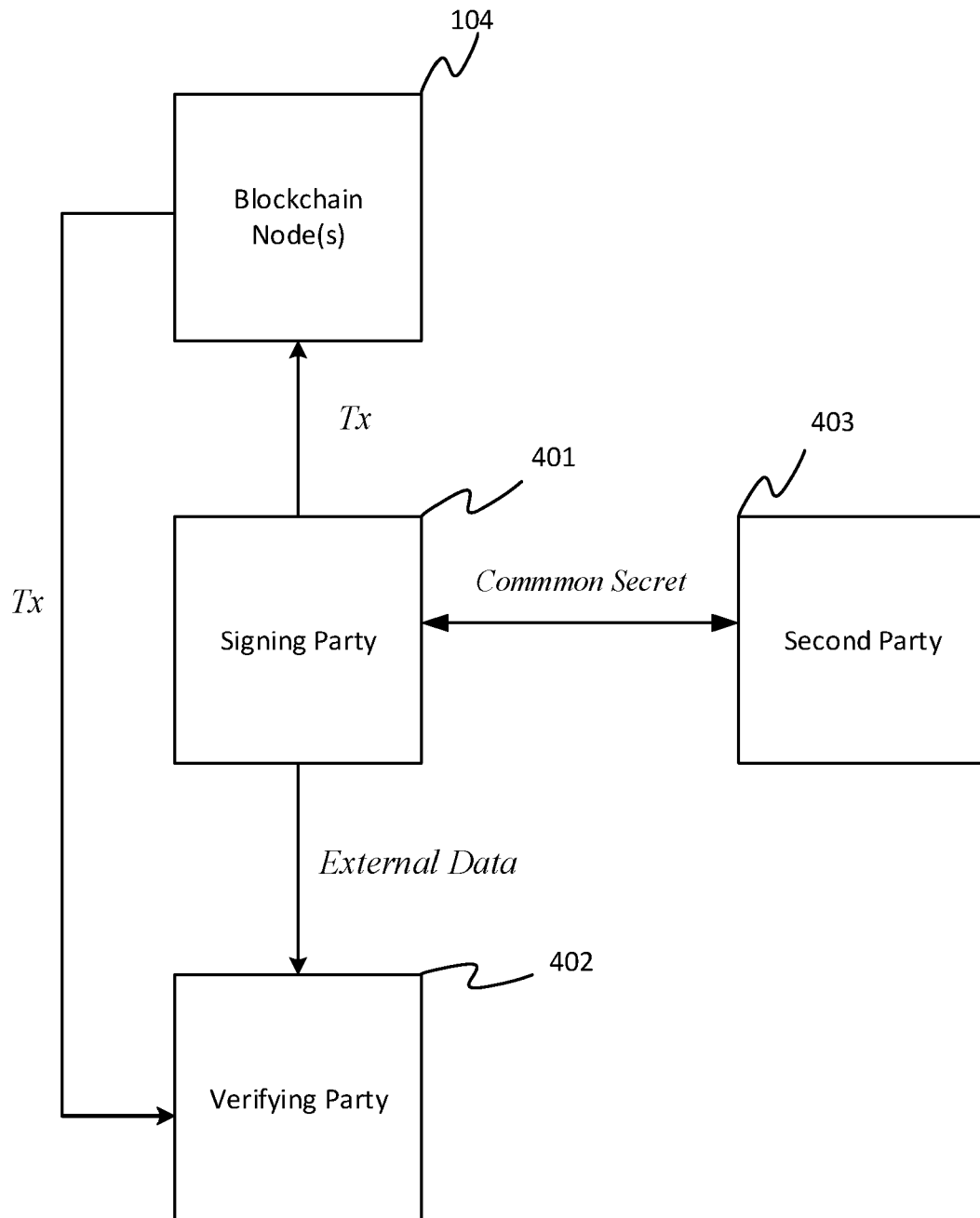
Figure 4

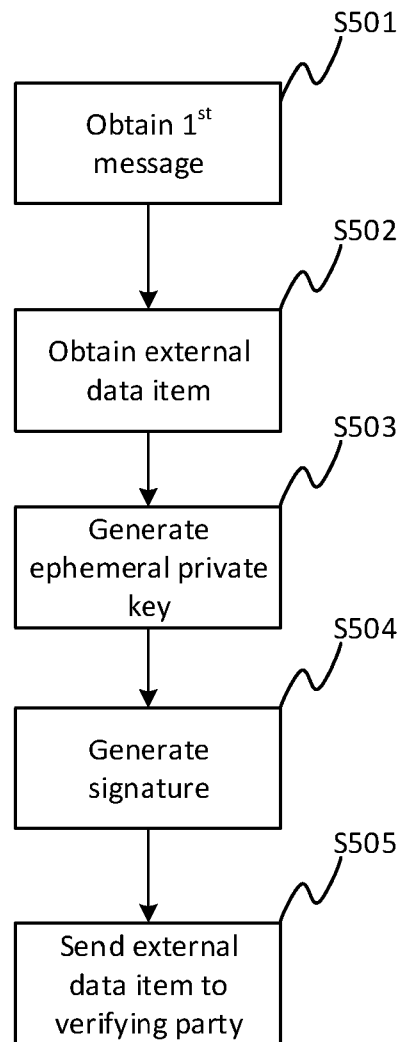
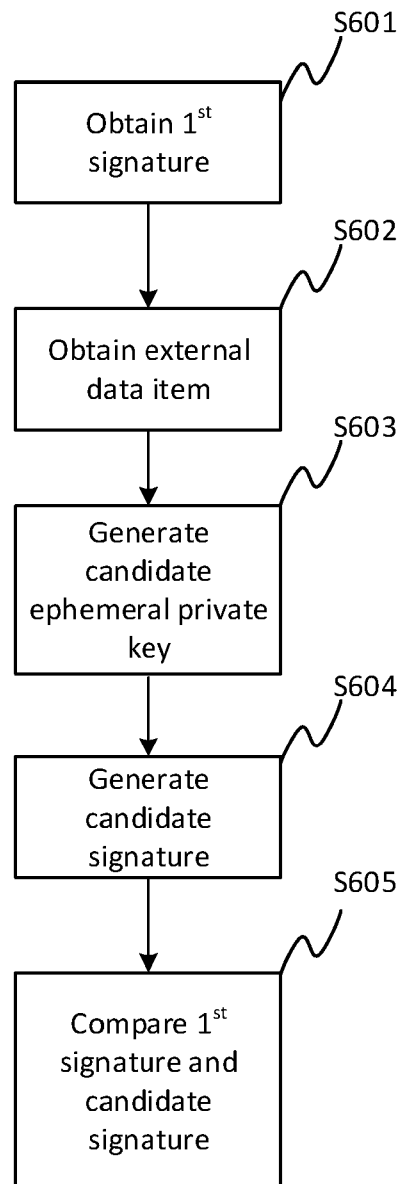
Figure 5

Figure 6

DIGITAL SIGNATURES

TECHNICAL FIELD

- 5 The present disclosure relates to a method of generating a digital signature. In particular, the method enables external data to be embedded within a digital signature.

BACKGROUND

- 10 Public-key cryptography is a type of cryptographic system that uses pairs of keys: private keys which are known only to the owner of the private key, and public keys which are generated based on the corresponding private key and which may be disseminated without compromising the security of the private key.

- 15 Public-key cryptography enables a sender to encrypt a message using a recipient's public key (i.e. the public key corresponding to a private key known only to the recipient). The encrypted message can then only be decrypted using the recipient's private key.

- 20 Similarly, a sender can use their own private key to sign a message, e.g. to prove that the message is being sent by the sender, and/or to indicate that the sender agrees with the message. The signer (i.e. the party generating the signature) uses their private key to create a digital signature based on the message. Creating a digital signature based on a message means supplying the message and private key to a function that generate the signature based on both the message and private key. The signature is added to (e.g. tagged onto) the
- 25 message or otherwise associated with the message. Anyone with the signer's corresponding public key can use the same message and the digital signature on the message to verify whether the signature was validly created, i.e. whether the signature was indeed made using the signer's private key. As well as ensuring the authenticity of a message, digital signatures also ensure the integrity and non-repudiation of the message. That is, a digital
- 30 signature can be used to prove that a message has not been changed since it was signed with the signature, and that the creator of a signature cannot deny in the future that they created the signature.

A digital signature scheme typically involves three procedures, i.e. algorithms. A key generation algorithm is used to generate a random private key and a corresponding public key. A signing algorithm is used to generate a signature based on a message and the private key. A verification algorithm is used to verify, given a public key and the message, whether the signature has been generated using the corresponding private key and according to the signing algorithm.

SUMMARY

Signatures enable the owner of a private key to attest to a given message, with the signature being verified using a public key corresponding to the private key. For instance, according to most blockchain protocols, bitcoin or other types of digital assets may be locked by requiring a valid signature created with a private key corresponding to a specified public key. To prevent the linking of blockchain transactions for security reasons, it is recommended that the same private key is not used more than once. This means that users will likely have multiple different keys that they will use to create multiple different signatures. The use of many different keys makes it difficult for a user to prove that they have generated any one signature. It would therefore be desirable for a user to be able to prove that they generated a signature, and to be able to do so in such a way that does not require the same private key to be used more than once.

According to one aspect disclosed herein, there is provided a computer-implemented method of generating a digital signature, wherein the method is performed by a signing party and comprises: obtaining a first message; generating an ephemeral private key based on at least a hash of an external data item; and generating a first signature comprising first and second signature components, wherein the first signature component is generated based on an ephemeral public key corresponding to the ephemeral private key, and wherein the second signature component is generated based on the first message, the ephemeral private key, the first signature component and a first private key.

According to another aspect disclosed herein, there is provided a computer-implemented method of verifying that a digital signature has been generated by a signing party, wherein the method is performed by a verifying party and comprises: obtaining a first signature
5 comprising first and second signature components; obtaining a candidate external data item from the signing party; generating a candidate ephemeral private key based on a hash of the candidate external data item; generating a candidate first signature component based on at least a public key corresponding to the candidate ephemeral private key; and verifying that
10 the first signature has been generated by the signing party based on whether the candidate first signature component corresponds to the first signature component.

A signing party generates a digital signature for signing a message. In the context of the blockchain, the message may be a transaction, e.g. the signing party generates a signature for unlocking an output of a previous transaction. In general the message may be any form
15 of message, e.g. a document, and does not necessarily need to be related to the blockchain. The signature is generated at least in part based on an ephemeral private key, e.g. a one-time use private key. The ephemeral private key is generated at least in part based on external information, i.e. the “external data item”. The external data item may comprise and/or be generated based on an identifier of the signing party, e.g. a name, address, phone
20 number, national insurance number, passport number, public key, etc. In some examples the external data item is another digital signature.

Once generated, the signature can be verified using a public key. However that is not enough to prove that the signing party generated the signature. Rather, since only the
25 signing party knows the external data item, the signing party can reveal the external data item and enable a verifying party to reconstruct at least part (i.e. the “first signature component”) of the signature. That is, the verifying party generates a candidate first signature component based on the external data item. If the reconstructed first signature component (i.e. the candidate first signature component) matches the first signature
30 component, then the verifying party can be sure that the signing party did indeed generate the signature.

The present invention enables external information to be incorporated into, i.e. embedded within, a signature. The external information is not known by a verifying party unless it is provided by the signing party. Specifically, the external information is used to derive the signature. For instance, the signing party may embed a public key that is linked to the signer's identity into a signature that is created using a different private key (i.e. a private key that does not correspond to the embedded public key). This allows the signing party to prove that they have generated the signature without having to re-use the private key corresponding to the identity-linked public key.

10 BRIEF DESCRIPTION OF THE DRAWINGS

To assist understanding of embodiments of the present disclosure and to show how such embodiments may be put into effect, reference is made, by way of example only, to the accompanying drawings in which:

15

Figure 1 is a schematic block diagram of a system for implementing a blockchain,

Figure 2 schematically illustrates some examples of transactions which may be recorded in a blockchain,

20

Figure 3A is a schematic block diagram of a client application,

Figure 3B is a schematic mock-up of an example user interface that may be presented by the client application of Figure 3A,

25

Figure 4 is a schematic block diagram of an example system for implementing embodiments of the present invention,

Figure 5 is a flow chart showing an example method for generating a digital signature according to some embodiments of the present invention, and

30

Figure 6 is a flow chart showing an example method for verifying that a party has generated a digital signature according to some embodiments of the present invention.

DETAILED DESCRIPTION OF EMBODIMENTS

5

EXAMPLE SYSTEM OVERVIEW

Figure 1 shows an example system 100 for implementing a blockchain 150. The system 100 may comprise a packet-switched network 101, typically a wide-area internetwork such as the Internet. The packet-switched network 101 comprises a plurality of blockchain nodes 104 that may be arranged to form a peer-to-peer (P2P) network 106 within the packet-switched network 101. Whilst not illustrated, the blockchain nodes 104 may be arranged as a near-complete graph. Each blockchain node 104 is therefore highly connected to other blockchain nodes 104.

15

Each blockchain node 104 comprises computer equipment of a peer, with different ones of the nodes 104 belonging to different peers. Each blockchain node 104 comprises processing apparatus comprising one or more processors, e.g. one or more central processing units (CPUs), accelerator processors, application specific processors and/or field programmable gate arrays (FPGAs), and other equipment such as application specific integrated circuits (ASICs). Each node also comprises memory, i.e. computer-readable storage in the form of a non-transitory computer-readable medium or media. The memory may comprise one or more memory units employing one or more memory media, e.g. a magnetic medium such as a hard disk; an electronic medium such as a solid-state drive (SSD), flash memory or EEPROM; and/or an optical medium such as an optical disk drive.

25

The blockchain 150 comprises a chain of blocks of data 151, wherein a respective copy of the blockchain 150 is maintained at each of a plurality of blockchain nodes 104 in the distributed or blockchain network 106. As mentioned above, maintaining a copy of the blockchain 150 does not necessarily mean storing the blockchain 150 in full. Instead, the blockchain 150 may be pruned of data so long as each blockchain node 150 stores the block header (discussed below) of each block 151. Each block 151 in the chain comprises one or

30

more transactions 152, wherein a transaction in this context refers to a kind of data structure. The nature of the data structure will depend on the type of transaction protocol used as part of a transaction model or scheme. A given blockchain will use one particular transaction protocol throughout. In one common type of transaction protocol, the data structure of each transaction 152 comprises at least one input and at least one output. Each output specifies an amount representing a quantity of a digital asset as property, an example of which is a user 103 to whom the output is cryptographically locked (requiring a signature or other solution of that user in order to be unlocked and thereby redeemed or spent). Each input points back to the output of a preceding transaction 152, thereby linking the transactions.

Each block 151 also comprises a block pointer 155 pointing back to the previously created block 151 in the chain so as to define a sequential order to the blocks 151. Each transaction 152 (other than a coinbase transaction) comprises a pointer back to a previous transaction so as to define an order to sequences of transactions (N.B. sequences of transactions 152 are allowed to branch). The chain of blocks 151 goes all the way back to a genesis block (Gb) 153 which was the first block in the chain. One or more original transactions 152 early on in the chain 150 pointed to the genesis block 153 rather than a preceding transaction.

Each of the blockchain nodes 104 is configured to forward transactions 152 to other blockchain nodes 104, and thereby cause transactions 152 to be propagated throughout the network 106. Each blockchain node 104 is configured to create blocks 151 and to store a respective copy of the same blockchain 150 in their respective memory. Each blockchain node 104 also maintains an ordered set (or “pool”) 154 of transactions 152 waiting to be incorporated into blocks 151. The ordered pool 154 is often referred to as a “mempool”. This term herein is not intended to limit to any particular blockchain, protocol or model. It refers to the ordered set of transactions which a node 104 has accepted as valid and for which the node 104 is obliged not to accept any other transactions attempting to spend the same output.

In a given present transaction 152_j, the (or each) input comprises a pointer referencing the output of a preceding transaction 152_i in the sequence of transactions, specifying that this

output is to be redeemed or “spent” in the present transaction 152j. In general, the preceding transaction could be any transaction in the ordered set 154 or any block 151. The preceding transaction 152i need not necessarily exist at the time the present transaction 152j is created or even sent to the network 106, though the preceding transaction 152i will need to exist and be validated in order for the present transaction to be valid. Hence “preceding” herein refers to a predecessor in a logical sequence linked by pointers, not necessarily the time of creation or sending in a temporal sequence, and hence it does not necessarily exclude that the transactions 152i, 152j be created or sent out-of-order (see discussion below on orphan transactions). The preceding transaction 152i could equally be called the antecedent or predecessor transaction.

The input of the present transaction 152j also comprises the input authorisation, for example the signature of the user 103a to whom the output of the preceding transaction 152i is locked. In turn, the output of the present transaction 152j can be cryptographically locked to a new user or entity 103b. The present transaction 152j can thus transfer the amount defined in the input of the preceding transaction 152i to the new user or entity 103b as defined in the output of the present transaction 152j. In some cases a transaction 152 may have multiple outputs to split the input amount between multiple users or entities (one of whom could be the original user or entity 103a in order to give change). In some cases a transaction can also have multiple inputs to gather together the amounts from multiple outputs of one or more preceding transactions, and redistribute to one or more outputs of the current transaction.

According to an output-based transaction protocol such as bitcoin, when a party 103, such as an individual user or an organization, wishes to enact a new transaction 152j (either manually or by an automated process employed by the party), then the enacting party sends the new transaction from its computer terminal 102 to a recipient. The enacting party or the recipient will eventually send this transaction to one or more of the blockchain nodes 104 of the network 106 (which nowadays are typically servers or data centres, but could in principle be other user terminals). It is also not excluded that the party 103 enacting the new transaction 152j could send the transaction directly to one or more of the blockchain nodes 104 and, in some examples, not to the recipient. A blockchain node 104 that receives

a transaction checks whether the transaction is valid according to a blockchain node protocol which is applied at each of the blockchain nodes 104. The blockchain node protocol typically requires the blockchain node 104 to check that a cryptographic signature in the new transaction 152j matches the expected signature, which depends on the previous transaction 152i in an ordered sequence of transactions 152. In such an output-based transaction protocol, this may comprise checking that the cryptographic signature or other authorisation of the party 103 included in the input of the new transaction 152j matches a condition defined in the output of the preceding transaction 152i which the new transaction assigns, wherein this condition typically comprises at least checking that the cryptographic signature or other authorisation in the input of the new transaction 152j unlocks the output of the previous transaction 152i to which the input of the new transaction is linked to. The condition may be at least partially defined by a script included in the output of the preceding transaction 152i. Alternatively it could simply be fixed by the blockchain node protocol alone, or it could be due to a combination of these. Either way, if the new transaction 152j is valid, the blockchain node 104 forwards it to one or more other blockchain nodes 104 in the blockchain network 106. These other blockchain nodes 104 apply the same test according to the same blockchain node protocol, and so forward the new transaction 152j on to one or more further nodes 104, and so forth. In this way the new transaction is propagated throughout the network of blockchain nodes 104.

In an output-based model, the definition of whether a given output (e.g. UTXO) is assigned (e.g. spent) is whether it has yet been validly redeemed by the input of another, onward transaction 152j according to the blockchain node protocol. Another condition for a transaction to be valid is that the output of the preceding transaction 152i which it attempts to redeem has not already been redeemed by another transaction. Again if not valid, the transaction 152j will not be propagated (unless flagged as invalid and propagated for alerting) or recorded in the blockchain 150. This guards against double-spending whereby the transactor tries to assign the output of the same transaction more than once. An account-based model on the other hand guards against double-spending by maintaining an account balance. Because again there is a defined order of transactions, the account balance has a single defined state at any one time.

In addition to validating transactions, blockchain nodes 104 also race to be the first to create blocks of transactions in a process commonly referred to as mining, which is supported by “proof-of-work”. At a blockchain node 104, new transactions are added to an ordered pool 154 of valid transactions that have not yet appeared in a block 151 recorded on the blockchain 150. The blockchain nodes then race to assemble a new valid block 151 of transactions 152 from the ordered set of transactions 154 by attempting to solve a cryptographic puzzle. Typically this comprises searching for a “nonce” value such that when the nonce is concatenated with a representation of the ordered pool of pending transactions 154 and hashed, then the output of the hash meets a predetermined condition. E.g. the predetermined condition may be that the output of the hash has a certain predefined number of leading zeros. Note that this is just one particular type of proof-of-work puzzle, and other types are not excluded. A property of a hash function is that it has an unpredictable output with respect to its input. Therefore this search can only be performed by brute force, thus consuming a substantive amount of processing resource at each blockchain node 104 that is trying to solve the puzzle.

The first blockchain node 104 to solve the puzzle announces this to the network 106, providing the solution as proof which can then be easily checked by the other blockchain nodes 104 in the network (once given the solution to a hash it is straightforward to check that it causes the output of the hash to meet the condition). The first blockchain node 104 propagates a block to a threshold consensus of other nodes that accept the block and thus enforce the protocol rules. The ordered set of transactions 154 then becomes recorded as a new block 151 in the blockchain 150 by each of the blockchain nodes 104. A block pointer 155 is also assigned to the new block 151n pointing back to the previously created block 151n-1 in the chain. The significant amount of effort, for example in the form of hash, required to create a proof-of-work solution signals the intent of the first node 104 to follow the rules of the blockchain protocol. Such rules include not accepting a transaction as valid if it assigns the same output as a previously validated transaction, otherwise known as double-spending. Once created, the block 151 cannot be modified since it is recognized and maintained at each of the blockchain nodes 104 in the blockchain network 106. The block pointer 155 also imposes a sequential order to the blocks 151. Since the transactions 152

are recorded in the ordered blocks at each blockchain node 104 in a network 106, this therefore provides an immutable public ledger of the transactions.

Note that different blockchain nodes 104 racing to solve the puzzle at any given time may be doing so based on different snapshots of the pool of yet-to-be published transactions 154 at any given time, depending on when they started searching for a solution or the order in which the transactions were received. Whoever solves their respective puzzle first defines which transactions 152 are included in the next new block 151n and in which order, and the current pool 154 of unpublished transactions is updated. The blockchain nodes 104 then continue to race to create a block from the newly-defined ordered pool of unpublished transactions 154, and so forth. A protocol also exists for resolving any “fork” that may arise, which is where two blockchain nodes 104 solve their puzzle within a very short time of one another such that a conflicting view of the blockchain gets propagated between nodes 104. In short, whichever prong of the fork grows the longest becomes the definitive blockchain 150. Note this should not affect the users or agents of the network as the same transactions will appear in both forks.

According to the bitcoin blockchain (and most other blockchains) a node that successfully constructs a new block 104 is granted the ability to newly assign an additional, accepted amount of the digital asset in a new special kind of transaction which distributes an additional defined quantity of the digital asset (as opposed to an inter-agent, or inter-user transaction which transfers an amount of the digital asset from one agent or user to another). This special type of transaction is usually referred to as a “coinbase transaction”, but may also be termed an “initiation transaction” or “generation transaction”. It typically forms the first transaction of the new block 151n. The proof-of-work signals the intent of the node that constructs the new block to follow the protocol rules allowing this special transaction to be redeemed later. The blockchain protocol rules may require a maturity period, for example 100 blocks, before this special transaction may be redeemed. Often a regular (non-generation) transaction 152 will also specify an additional transaction fee in one of its outputs, to further reward the blockchain node 104 that created the block 151n in which that transaction was published. This fee is normally referred to as the “transaction fee”, and is discussed below.

Due to the resources involved in transaction validation and publication, typically at least each of the blockchain nodes 104 takes the form of a server comprising one or more physical server units, or even whole a data centre. However in principle any given blockchain node 104 could take the form of a user terminal or a group of user terminals networked together.

The memory of each blockchain node 104 stores software configured to run on the processing apparatus of the blockchain node 104 in order to perform its respective role or roles and handle transactions 152 in accordance with the blockchain node protocol. It will be understood that any action attributed herein to a blockchain node 104 may be performed by the software run on the processing apparatus of the respective computer equipment. The node software may be implemented in one or more applications at the application layer, or a lower layer such as the operating system layer or a protocol layer, or any combination of these.

Also connected to the network 101 is the computer equipment 102 of each of a plurality of parties 103 in the role of consuming users. These users may interact with the blockchain network 106 but do not participate in validating transactions or constructing blocks. Some of these users or agents 103 may act as senders and recipients in transactions. Other users may interact with the blockchain 150 without necessarily acting as senders or recipients. For instance, some parties may act as storage entities that store a copy of the blockchain 150 (e.g. having obtained a copy of the blockchain from a blockchain node 104).

Some or all of the parties 103 may be connected as part of a different network, e.g. a network overlaid on top of the blockchain network 106. Users of the blockchain network (often referred to as “clients”) may be said to be part of a system that includes the blockchain network 106; however, these users are not blockchain nodes 104 as they do not perform the roles required of the blockchain nodes. Instead, each party 103 may interact with the blockchain network 106 and thereby utilize the blockchain 150 by connecting to (i.e. communicating with) a blockchain node 106. Two parties 103 and their respective equipment 102 are shown for illustrative purposes: a first party 103a and his/her respective

computer equipment 102a, and a second party 103b and his/her respective computer equipment 102b. It will be understood that many more such parties 103 and their respective computer equipment 102 may be present and participating in the system 100, but for convenience they are not illustrated. Each party 103 may be an individual or an organization. Purely by way of illustration the first party 103a is referred to herein as Alice and the second party 103b is referred to as Bob, but it will be appreciated that this is not limiting and any reference herein to Alice or Bob may be replaced with “first party” and “second “party” respectively.

The computer equipment 102 of each party 103 comprises respective processing apparatus comprising one or more processors, e.g. one or more CPUs, GPUs, other accelerator processors, application specific processors, and/or FPGAs. The computer equipment 102 of each party 103 further comprises memory, i.e. computer-readable storage in the form of a non-transitory computer-readable medium or media. This memory may comprise one or more memory units employing one or more memory media, e.g. a magnetic medium such as hard disk; an electronic medium such as an SSD, flash memory or EEPROM; and/or an optical medium such as an optical disc drive. The memory on the computer equipment 102 of each party 103 stores software comprising a respective instance of at least one client application 105 arranged to run on the processing apparatus. It will be understood that any action attributed herein to a given party 103 may be performed using the software run on the processing apparatus of the respective computer equipment 102. The computer equipment 102 of each party 103 comprises at least one user terminal, e.g. a desktop or laptop computer, a tablet, a smartphone, or a wearable device such as a smartwatch. The computer equipment 102 of a given party 103 may also comprise one or more other networked resources, such as cloud computing resources accessed via the user terminal.

The client application 105 may be initially provided to the computer equipment 102 of any given party 103 on suitable computer-readable storage medium or media, e.g. downloaded from a server, or provided on a removable storage device such as a removable SSD, flash memory key, removable EEPROM, removable magnetic disk drive, magnetic floppy disk or tape, optical disk such as a CD or DVD ROM, or a removable optical drive, etc.

The client application 105 comprises at least a “wallet” function. This has two main functionalities. One of these is to enable the respective party 103 to create, authorise (for example sign) and send transactions 152 to one or more bitcoin nodes 104 to then be propagated throughout the network of blockchain nodes 104 and thereby included in the blockchain 150. The other is to report back to the respective party the amount of the digital asset that he or she currently owns. In an output-based system, this second functionality comprises collating the amounts defined in the outputs of the various 152 transactions scattered throughout the blockchain 150 that belong to the party in question.

10 Note: whilst the various client functionality may be described as being integrated into a given client application 105, this is not necessarily limiting and instead any client functionality described herein may instead be implemented in a suite of two or more distinct applications, e.g. interfacing via an API, or one being a plug-in to the other. More generally the client functionality could be implemented at the application layer or a lower
15 layer such as the operating system, or any combination of these. The following will be described in terms of a client application 105 but it will be appreciated that this is not limiting.

The instance of the client application or software 105 on each computer equipment 102 is
20 operatively coupled to at least one of the blockchain nodes 104 of the network 106. This enables the wallet function of the client 105 to send transactions 152 to the network 106. The client 105 is also able to contact blockchain nodes 104 in order to query the blockchain 150 for any transactions of which the respective party 103 is the recipient (or indeed inspect other parties’ transactions in the blockchain 150, since in embodiments the blockchain 150
25 is a public facility which provides trust in transactions in part through its public visibility). The wallet function on each computer equipment 102 is configured to formulate and send transactions 152 according to a transaction protocol. As set out above, each blockchain node 104 runs software configured to validate transactions 152 according to the blockchain node protocol, and to forward transactions 152 in order to propagate them throughout the
30 blockchain network 106. The transaction protocol and the node protocol correspond to one another, and a given transaction protocol goes with a given node protocol, together implementing a given transaction model. The same transaction protocol is used for all

transactions 152 in the blockchain 150. The same node protocol is used by all the nodes 104 in the network 106.

When a given party 103, say Alice, wishes to send a new transaction 152j to be included in the blockchain 150, then she formulates the new transaction in accordance with the relevant transaction protocol (using the wallet function in her client application 105). She then sends the transaction 152 from the client application 105 to one or more blockchain nodes 104 to which she is connected. E.g. this could be the blockchain node 104 that is best connected to Alice's computer 102. When any given blockchain node 104 receives a new transaction 152j, it handles it in accordance with the blockchain node protocol and its respective role. This comprises first checking whether the newly received transaction 152j meets a certain condition for being "valid", examples of which will be discussed in more detail shortly. In some transaction protocols, the condition for validation may be configurable on a per-transaction basis by scripts included in the transactions 152. Alternatively the condition could simply be a built-in feature of the node protocol, or be defined by a combination of the script and the node protocol.

On condition that the newly received transaction 152j passes the test for being deemed valid (i.e. on condition that it is "validated"), any blockchain node 104 that receives the transaction 152j will add the new validated transaction 152 to the ordered set of transactions 154 maintained at that blockchain node 104. Further, any blockchain node 104 that receives the transaction 152j will propagate the validated transaction 152 onward to one or more other blockchain nodes 104 in the network 106. Since each blockchain node 104 applies the same protocol, then assuming the transaction 152j is valid, this means it will soon be propagated throughout the whole network 106.

Once admitted to the ordered pool of pending transactions 154 maintained at a given blockchain node 104, that blockchain node 104 will start competing to solve the proof-of-work puzzle on the latest version of their respective pool of 154 including the new transaction 152 (recall that other blockchain nodes 104 may be trying to solve the puzzle based on a different pool of transactions 154, but whoever gets there first will define the set of transactions that are included in the latest block 151. Eventually a blockchain node 104

will solve the puzzle for a part of the ordered pool 154 which includes Alice's transaction 152j). Once the proof-of-work has been done for the pool 154 including the new transaction 152j, it immutably becomes part of one of the blocks 151 in the blockchain 150. Each transaction 152 comprises a pointer back to an earlier transaction, so the order of the transactions is also immutably recorded.

Different blockchain nodes 104 may receive different instances of a given transaction first and therefore have conflicting views of which instance is 'valid' before one instance is published in a new block 151, at which point all blockchain nodes 104 agree that the published instance is the only valid instance. If a blockchain node 104 accepts one instance as valid, and then discovers that a second instance has been recorded in the blockchain 150 then that blockchain node 104 must accept this and will discard (i.e. treat as invalid) the instance which it had initially accepted (i.e. the one that has not been published in a block 151).

An alternative type of transaction protocol operated by some blockchain networks may be referred to as an "account-based" protocol, as part of an account-based transaction model. In the account-based case, each transaction does not define the amount to be transferred by referring back to the UTXO of a preceding transaction in a sequence of past transactions, but rather by reference to an absolute account balance. The current state of all accounts is stored, by the nodes of that network, separate to the blockchain and is updated constantly. In such a system, transactions are ordered using a running transaction tally of the account (also called the "position"). This value is signed by the sender as part of their cryptographic signature and is hashed as part of the transaction reference calculation. In addition, an optional data field may also be signed the transaction. This data field may point back to a previous transaction, for example if the previous transaction ID is included in the data field.

UTXO-BASED MODEL

Figure 2 illustrates an example transaction protocol. This is an example of a UTXO-based protocol. A transaction 152 (abbreviated "Tx") is the fundamental data structure of the blockchain 150 (each block 151 comprising one or more transactions 152). The following will

be described by reference to an output-based or “UTXO” based protocol. However, this is not limiting to all possible embodiments. Note that while the example UTXO-based protocol is described with reference to bitcoin, it may equally be implemented on other example blockchain networks.

5

In a UTXO-based model, each transaction (“Tx”) 152 comprises a data structure comprising one or more inputs 202, and one or more outputs 203. Each output 203 may comprise an unspent transaction output (UTXO), which can be used as the source for the input 202 of another new transaction (if the UTXO has not already been redeemed). The UTXO includes a value specifying an amount of a digital asset. This represents a set number of tokens on the distributed ledger. The UTXO may also contain the transaction ID of the transaction from which it came, amongst other information. The transaction data structure may also comprise a header 201, which may comprise an indicator of the size of the input field(s) 202 and output field(s) 203. The header 201 may also include an ID of the transaction. In embodiments the transaction ID is the hash of the transaction data (excluding the transaction ID itself) and stored in the header 201 of the raw transaction 152 submitted to the nodes 104.

Say Alice 103a wishes to create a transaction 152j transferring an amount of the digital asset in question to Bob 103b. In Figure 2 Alice’s new transaction 152j is labelled “ T_{X1} ”. It takes an amount of the digital asset that is locked to Alice in the output 203 of a preceding transaction 152i in the sequence, and transfers at least some of this to Bob. The preceding transaction 152i is labelled “ T_{X0} ” in Figure 2. T_{X0} and T_{X1} are just arbitrary labels. They do not necessarily mean that T_{X0} is the first transaction in the blockchain 151, nor that T_{X1} is the immediate next transaction in the pool 154. T_{X1} could point back to any preceding (i.e. antecedent) transaction that still has an unspent output 203 locked to Alice.

The preceding transaction T_{X0} may already have been validated and included in a block 151 of the blockchain 150 at the time when Alice creates her new transaction T_{X1} , or at least by the time she sends it to the network 106. It may already have been included in one of the blocks 151 at that time, or it may be still waiting in the ordered set 154 in which case it will soon be included in a new block 151. Alternatively T_{X0} and T_{X1} could be created and sent to

30

the network 106 together, or Tx_0 could even be sent after Tx_1 if the node protocol allows for buffering “orphan” transactions. The terms “preceding” and “subsequent” as used herein in the context of the sequence of transactions refer to the order of the transactions in the sequence as defined by the transaction pointers specified in the transactions (which transaction points back to which other transaction, and so forth). They could equally be replaced with “predecessor” and “successor”, or “antecedent” and “descendant”, “parent” and “child”, or such like. It does not necessarily imply an order in which they are created, sent to the network 106, or arrive at any given blockchain node 104. Nevertheless, a subsequent transaction (the descendent transaction or “child”) which points to a preceding transaction (the antecedent transaction or “parent”) will not be validated until and unless the parent transaction is validated. A child that arrives at a blockchain node 104 before its parent is considered an orphan. It may be discarded or buffered for a certain time to wait for the parent, depending on the node protocol and/or node behaviour.

One of the one or more outputs 203 of the preceding transaction Tx_0 comprises a particular UTXO, labelled here $UTXO_0$. Each UTXO comprises a value specifying an amount of the digital asset represented by the UTXO, and a locking script which defines a condition which must be met by an unlocking script in the input 202 of a subsequent transaction in order for the subsequent transaction to be validated, and therefore for the UTXO to be successfully redeemed. Typically the locking script locks the amount to a particular party (the beneficiary of the transaction in which it is included). I.e. the locking script defines an unlocking condition, typically comprising a condition that the unlocking script in the input of the subsequent transaction comprises the cryptographic signature of the party to whom the preceding transaction is locked.

The locking script (aka scriptPubKey) is a piece of code written in the domain specific language recognized by the node protocol. A particular example of such a language is called “Script” (capital S) which is used by the blockchain network. The locking script specifies what information is required to spend a transaction output 203, for example the requirement of Alice’s signature. Unlocking scripts appear in the outputs of transactions. The unlocking script (aka scriptSig) is a piece of code written the domain specific language that provides

the information required to satisfy the locking script criteria. For example, it may contain Bob's signature. Unlocking scripts appear in the input 202 of transactions.

So in the example illustrated, $UTXO_0$ in the output 203 of T_{X0} comprises a locking script
 5 [Checksig P_A] which requires a signature Sig P_A of Alice in order for $UTXO_0$ to be redeemed (strictly, in order for a subsequent transaction attempting to redeem $UTXO_0$ to be valid). [Checksig P_A] contains a representation (i.e. a hash) of the public key P_A from a public-private key pair of Alice. The input 202 of T_{X1} comprises a pointer pointing back to T_{X0} (e.g. by means of its transaction ID, T_{XID_0} , which in embodiments is the hash of the whole
 10 transaction T_{X0}). The input 202 of T_{X1} comprises an index identifying $UTXO_0$ within T_{X0} , to identify it amongst any other possible outputs of T_{X0} . The input 202 of T_{X1} further comprises an unlocking script $\langle \text{Sig } P_A \rangle$ which comprises a cryptographic signature of Alice, created by Alice applying her private key from the key pair to a predefined portion of data (sometimes called the "message" in cryptography). The data (or "message") that needs to
 15 be signed by Alice to provide a valid signature may be defined by the locking script, or by the node protocol, or by a combination of these.

When the new transaction T_{X1} arrives at a blockchain node 104, the node applies the node protocol. This comprises running the locking script and unlocking script together to check
 20 whether the unlocking script meets the condition defined in the locking script (where this condition may comprise one or more criteria). In embodiments this involves concatenating the two scripts:

$\langle \text{Sig } P_A \rangle \langle P_A \rangle || [\text{Checksig } P_A]$

25

where " $||$ " represents a concatenation and " $\langle \dots \rangle$ " means place the data on the stack, and " $[...]$ " is a function comprised by the locking script (in this example a stack-based language). Equivalently the scripts may be run one after the other, with a common stack, rather than concatenating the scripts. Either way, when run together, the scripts use the public key P_A
 30 of Alice, as included in the locking script in the output of T_{X0} , to authenticate that the unlocking script in the input of T_{X1} contains the signature of Alice signing the expected portion of data. The expected portion of data itself (the "message") also needs to be

included in order to perform this authentication. In embodiments the signed data comprises the whole of TX_1 (so a separate element does not need to be included specifying the signed portion of data in the clear, as it is already inherently present).

- 5 The details of authentication by public-private cryptography will be familiar to a person skilled in the art. Basically, if Alice has signed a message using her private key, then given Alice's public key and the message in the clear, another entity such as a node 104 is able to authenticate that the message must have been signed by Alice. Signing typically comprises hashing the message, signing the hash, and tagging this onto the message as a signature,
10 thus enabling any holder of the public key to authenticate the signature. Note therefore that any reference herein to signing a particular piece of data or part of a transaction, or such like, can in embodiments mean signing a hash of that piece of data or part of the transaction.
- 15 If the unlocking script in TX_1 meets the one or more conditions specified in the locking script of TX_0 (so in the example shown, if Alice's signature is provided in TX_1 and authenticated), then the blockchain node 104 deems TX_1 valid. This means that the blockchain node 104 will add TX_1 to the ordered pool of pending transactions 154. The blockchain node 104 will also forward the transaction TX_1 to one or more other blockchain nodes 104 in the network 106,
20 so that it will be propagated throughout the network 106. Once TX_1 has been validated and included in the blockchain 150, this defines $UTXO_0$ from TX_0 as spent. Note that TX_1 can only be valid if it spends an unspent transaction output 203. If it attempts to spend an output that has already been spent by another transaction 152, then TX_1 will be invalid even if all the other conditions are met. Hence the blockchain node 104 also needs to check
25 whether the referenced UTXO in the preceding transaction TX_0 is already spent (i.e. whether it has already formed a valid input to another valid transaction). This is one reason why it is important for the blockchain 150 to impose a defined order on the transactions 152. In practice a given blockchain node 104 may maintain a separate database marking which UTXOs 203 in which transactions 152 have been spent, but ultimately what defines whether
30 a UTXO has been spent is whether it has already formed a valid input to another valid transaction in the blockchain 150.

If the total amount specified in all the outputs 203 of a given transaction 152 is greater than the total amount pointed to by all its inputs 202, this is another basis for invalidity in most transaction models. Therefore such transactions will not be propagated nor included in a block 151.

5

Note that in UTXO-based transaction models, a given UTXO needs to be spent as a whole. It cannot “leave behind” a fraction of the amount defined in the UTXO as spent while another fraction is spent. However the amount from the UTXO can be split between multiple outputs of the next transaction. E.g. the amount defined in $UTXO_0$ in TX_0 can be split between multiple UTXOs in TX_1 . Hence if Alice does not want to give Bob all of the amount defined in $UTXO_0$, she can use the remainder to give herself change in a second output of TX_1 , or pay another party.

10

In practice Alice will also usually need to include a fee for the bitcoin node 104 that successfully includes her transaction 104 in a block 151. If Alice does not include such a fee, TX_0 may be rejected by the blockchain nodes 104, and hence although technically valid, may not be propagated and included in the blockchain 150 (the node protocol does not force blockchain nodes 104 to accept transactions 152 if they don't want). In some protocols, the transaction fee does not require its own separate output 203 (i.e. does not need a separate UTXO). Instead any difference between the total amount pointed to by the input(s) 202 and the total amount of specified in the output(s) 203 of a given transaction 152 is automatically given to the blockchain node 104 publishing the transaction. E.g. say a pointer to $UTXO_0$ is the only input to TX_1 , and TX_1 has only one output $UTXO_1$. If the amount of the digital asset specified in $UTXO_0$ is greater than the amount specified in $UTXO_1$, then the difference may be assigned by the node 104 that wins the proof-of-work race to create the block containing $UTXO_1$. Alternatively or additionally however, it is not necessarily excluded that a transaction fee could be specified explicitly in its own one of the UTXOs 203 of the transaction 152.

20

25

30

Alice and Bob's digital assets consist of the UTXOs locked to them in any transactions 152 anywhere in the blockchain 150. Hence typically, the assets of a given party 103 are scattered throughout the UTXOs of various transactions 152 throughout the blockchain 150.

There is no one number stored anywhere in the blockchain 150 that defines the total balance of a given party 103. It is the role of the wallet function in the client application 105 to collate together the values of all the various UTXOs which are locked to the respective party and have not yet been spent in another onward transaction. It can do this by querying the copy of the blockchain 150 as stored at any of the bitcoin nodes 104.

Note that the script code is often represented schematically (i.e. not using the exact language). For example, one may use operation codes (opcodes) to represent a particular function. “OP_...” refers to a particular opcode of the Script language. As an example, OP_RETURN is an opcode of the Script language that when preceded by OP_FALSE at the beginning of a locking script creates an unspendable output of a transaction that can store data within the transaction, and thereby record the data immutably in the blockchain 150. E.g. the data could comprise a document which it is desired to store in the blockchain.

Typically an input of a transaction contains a digital signature corresponding to a public key P_A . In embodiments this is based on the ECDSA using the elliptic curve secp256k1. A digital signature signs a particular piece of data. In some embodiments, for a given transaction the signature will sign part of the transaction input, and some or all of the transaction outputs. The particular parts of the outputs it signs depends on the SIGHASH flag. The SIGHASH flag is usually a 4-byte code included at the end of a signature to select which outputs are signed (and thus fixed at the time of signing).

The locking script is sometimes called “scriptPubKey” referring to the fact that it typically comprises the public key of the party to whom the respective transaction is locked. The unlocking script is sometimes called “scriptSig” referring to the fact that it typically supplies the corresponding signature. However, more generally it is not essential in all applications of a blockchain 150 that the condition for a UTXO to be redeemed comprises authenticating a signature. More generally the scripting language could be used to define any one or more conditions. Hence the more general terms “locking script” and “unlocking script” may be preferred.

As shown in Figure 1, the client application on each of Alice and Bob's computer equipment 102a, 120b, respectively, may comprise additional communication functionality. This additional functionality enables Alice 103a to establish a separate side channel 107 with Bob 103b (at the instigation of either party or a third party). The side channel 107 enables exchange of data separately from the blockchain network. Such communication is sometimes referred to as "off-chain" communication. For instance this may be used to exchange a transaction 152 between Alice and Bob without the transaction (yet) being registered onto the blockchain network 106 or making its way onto the chain 150, until one of the parties chooses to broadcast it to the network 106. Sharing a transaction in this way is sometimes referred to as sharing a "transaction template". A transaction template may lack one or more inputs and/or outputs that are required in order to form a complete transaction. Alternatively or additionally, the side channel 107 may be used to exchange any other transaction related data, such as keys, negotiated amounts or terms, data content, etc.

The side channel 107 may be established via the same packet-switched network 101 as the blockchain network 106. Alternatively or additionally, the side channel 301 may be established via a different network such as a mobile cellular network, or a local area network such as a local wireless network, or even a direct wired or wireless link between Alice and Bob's devices 102a, 102b. Generally, the side channel 107 as referred to anywhere herein may comprise any one or more links via one or more networking technologies or communication media for exchanging data "off-chain", i.e. separately from the blockchain network 106. Where more than one link is used, then the bundle or collection of off-chain links as a whole may be referred to as the side channel 107. Note therefore that if it is said that Alice and Bob exchange certain pieces of information or data, or such like, over the side channel 107, then this does not necessarily imply all these pieces of data have to be send over exactly the same link or even the same type of network.

CLIENT SOFTWARE

Figure 3A illustrates an example implementation of the client application 105 for implementing embodiments of the presently disclosed scheme. The client application 105 comprises a transaction engine 401 and a user interface (UI) layer 402. The transaction engine 401 is configured to implement the underlying transaction-related functionality of the client 105, such as to formulate transactions 152, receive and/or send transactions and/or other data over the side channel 301, and/or send transactions to one or more nodes 104 to be propagated through the blockchain network 106, in accordance with the schemes discussed above and as discussed in further detail shortly.

The UI layer 402 is configured to render a user interface via a user input/output (I/O) means of the respective user's computer equipment 102, including outputting information to the respective user 103 via a user output means of the equipment 102, and receiving inputs back from the respective user 103 via a user input means of the equipment 102. For example the user output means could comprise one or more display screens (touch or non-touch screen) for providing a visual output, one or more speakers for providing an audio output, and/or one or more haptic output devices for providing a tactile output, etc. The user input means could comprise for example the input array of one or more touch screens (the same or different as that/those used for the output means); one or more cursor-based devices such as mouse, trackpad or trackball; one or more microphones and speech or voice recognition algorithms for receiving a speech or vocal input; one or more gesture-based input devices for receiving the input in the form of manual or bodily gestures; or one or more mechanical buttons, switches or joysticks, etc.

Note: whilst the various functionality herein may be described as being integrated into the same client application 105, this is not necessarily limiting and instead they could be implemented in a suite of two or more distinct applications, e.g. one being a plug-in to the other or interfacing via an API (application programming interface). For instance, the functionality of the transaction engine 401 may be implemented in a separate application than the UI layer 402, or the functionality of a given module such as the transaction engine 401 could be split between more than one application. Nor is it excluded that some or all of the described functionality could be implemented at, say, the operating system layer.

Where reference is made anywhere herein to a single or given application 105, or such like,

it will be appreciated that this is just by way of example, and more generally the described functionality could be implemented in any form of software.

Figure 3B gives a mock-up of an example of the user interface (UI) 500 which may be rendered by the UI layer 402 of the client application 105a on Alice's equipment 102a. It will be appreciated that a similar UI may be rendered by the client 105b on Bob's equipment 102b, or that of any other party.

By way of illustration Figure 3B shows the UI 500 from Alice's perspective. The UI 500 may comprise one or more UI elements 501, 502, 502 rendered as distinct UI elements via the user output means.

For example, the UI elements may comprise one or more user-selectable elements 501 which may be, such as different on-screen buttons, or different options in a menu, or such like. The user input means is arranged to enable the user 103 (in this case Alice 103a) to select or otherwise operate one of the options, such as by clicking or touching the UI element on-screen, or speaking a name of the desired option (N.B. the term "manual" as used herein is meant only to contrast against automatic, and does not necessarily limit to the use of the hand or hands). The options enable the user (Alice) to generate a signature to be embedded within a transaction.

Alternatively or additionally, the UI elements may comprise one or more data entry fields 502, through which the user can enter data to be embedded within a signature. These data entry fields are rendered via the user output means, e.g. on-screen, and the data can be entered into the fields through the user input means, e.g. a keyboard or touchscreen. Alternatively the data could be received orally for example based on speech recognition.

Alternatively or additionally, the UI elements may comprise one or more information elements 503 output to output information to the user. E.g. this/these could be rendered on screen or audibly.

It will be appreciated that the particular means of rendering the various UI elements, selecting the options and entering data is not material. The functionality of these UI elements will be discussed in more detail shortly. It will also be appreciated that the UI 500 shown in Figure 3 is only a schematized mock-up and in practice it may comprise one or more further UI elements, which for conciseness are not illustrated.

CRYPTOGRAPHIC PRELIMINARIES

ECDSA - Elliptic Curve Groups

- 10 $(E, \oplus)$ is a cyclic elliptic curve group over finite field $\mathbb{F}_p$ where p is prime. The number of elements in E is n where n is prime. $G \in E$ is the generator point of the elliptic curve group, meaning:

$$\forall Y \in E \quad \exists i \in \{1, \dots, n\} : Y = i \cdot G.$$

- The group operation ' $\oplus$ ' is standard elliptic curve point addition and $i \cdot G$ denotes i repetitions of the group operation on G

$$i \cdot G = \underbrace{G \oplus G \oplus \dots \oplus G}_{i \text{ times}}$$

In the following all operations on integers are modulo n unless the context requires otherwise.

20 Elliptic Curve Digital Signature Algorithm

Key generation is done as follows:

- 1) Choose a private signing key $j \in \{1, \dots, n - 1\}$
- 2) The public key is $Y = j \cdot G$ where G is the generator point

- 25 The signing algorithm takes the private key j message m and an ephemeral key k and generates the signature:

- 3) Choose random $k \in \{1, \dots, n - 1\}$ (ephemeral key)
- 4) Calculate $R = (r_x, r_y) = k \cdot G$ - EC point
- 5) Calculate $r = r_x \bmod n$

- 30 a. If $r = 0$ go to step 3

- 6) Generate signature $s = k^{-1}(e + jr)$ where $e = \text{hash}(m)$.

- 7) If $s = 0$ go to step 3
- 8) Output $[r, s]$ as the signature for message m

The verification algorithm then takes the signature and message and reconstructs r using the signer's public key and verifies the r value given in the signature.

- 1) Calculate $e = \text{hash}(m)$
- 2) Calculate $k_1 = es^{-1} \bmod n$ and $k_2 = rs^{-1} \bmod n$
- 3) Calculate $Q = (q_x, q_y) = k_1 \cdot G + k_2 \cdot Y$
- 4) If $q_x \equiv r \bmod n$ then the signature is valid. Otherwise invalid.

The following notation is used for the signature:

$$\text{Sig}_Y = [r_Y, s_Y],$$

where $[r_Y, s_Y]$ is a valid signature when verified using the public key Y .

Diffie-Hellman (DH) key exchange

Two parties may establish a secure communication channel by creating a symmetric secret key in the following way. Assume that Alice and Bob want to create a shared secret key, and that Alice has knowledge of the private key sk_A corresponding to the public key $PK_A = sk_A \cdot G$ and Bob knows the private key sk_B corresponding to his public key $PK_B = sk_B \cdot G$.

In order to find the shared secret key, they do the following steps.

1. Alice calculates the Diffie-Hellman key $sk_{AB} = sk_A \cdot PK_B$.
2. Bob calculates the Diffie-Hellman key $sk_{AB} = sk_B \cdot PK_A$.

Another method for establishing a shared secret key is described in WO2017/145016 in which a pre-agreed message is added onto a DH key, creating a new key. This message can be changed with each new communication that is sent, creating a set of deterministic keys.

For example, the message may be $m = \text{hash}(\text{date} || \text{time})$. Alice can then use the message to generate a private key $sk_{A1} = sk_A + \text{hash}(\text{date} || \text{time})$, and similarly Bob can generate

a private key $sk_{B1} = sk_B + \text{hash}(\text{date}||\text{time})$. Both Alice and Bob can then generate the shared private key $sk_{AB1} = sk_{A1} \cdot PK_{B1} = sk_{B1} \cdot PK_{A1}$.

HD Wallets

5 Hierarchical Deterministic wallets, of which a Bitcoin Improvement Proposal 32 (BIP32) wallet is a particular type, are deterministic wallets where many keys can be derived from a single input. The input is some random entropy called the seed, from which a master key is derived. The master key is then used to derive multiple child keys, as shown in Figure 2.

10 In BIP32 the master private key is the left 32 bytes of the result of the HMAC-SHA512 of the seed, or explicitly, it is

$$sk_{master} = \text{HMAC-SHA512}_L('Bitcoin Seed', seed),$$

15 and the chain code is

$$c_{master} = \text{HMAC-SHA512}_R('Bitcoin Seed', seed),$$

and all child keys can be then derived from these, where

20

$$\text{HMAC-SHA512}(c, K) = \text{SHA512}(c \oplus \text{opad} || \text{SHA512}((c \oplus \text{ipad}) || K))$$

is the HMAC using the SHA512 hash function. In the equation above, *opad* is the block-sized outer padding, and *ipad* is the block-sized inner padding.

25

A HMAC requires two inputs, i.e. *c* and *K*. For simplicity and so that users are only required to remember or store a single seed, the BIP32 protocol sets the first input as the string 'Bitcoin Seed', i.e. $c = 'Bitcoin Seed'$. It will be appreciated that this is one example protocol for generating a HD wallet and that different protocols may require different

30 inputs, e.g. two randomly generated seeds. In other words, the use of the string 'Bitcoin Seed' is not a necessary requirement for generating a HD wallet.

The equation for calculating a hardened child private key sk_{child} from a parent private key sk_{parent} is

$$sk_{child} = sk_{parent} + HMAC-SHA512_L(c_{parent}, sk_{parent} || index) ,$$

5

where c_{parent} is the parent chain code, $0 \leq index < 2^{31}$ is the child index, and $HMAC-SHA512_L$ is the left 32 bytes of the result of the HMAC function calculated with the SHA-512 hash function. The corresponding equation for the child public key is derived by simply point multiplying this equation by the base point G . The child chain code c_{child} is defined to be the right 32 bytes of the result of the HMAC function, $c_{child} = HMAC-SHA512_R(c_{parent}, sk_{parent} || index)$.

10

The equation for calculating a non-hardened child private key sk_{child} from a parent public key pk_{parent} and parent private key sk_{parent} is

15

$$sk_{child} = sk_{parent} + HMAC-SHA512_L(c_{parent}, pk_{parent} || index) ,$$

where c_{parent} is the parent chain code, $2^{31} \leq index < 2^{32}$ is the child index, and $HMAC-SHA512$ is the HMAC function calculated with the SHA-512 hash function. Similar to hardened keys, the child chain code c_{child} for non-hardened keys is defined to be the right 32 bytes of the result of the HMAC function:

20

$$c_{child} = HMAC-SHA512_R(c_{parent}, pk_{parent} || index) .$$

25

This second type of child key allows for child public keys to be derived by anyone with knowledge of the parent public key and chain code using the equation

$$pk_{child} = pk_{parent} + HMAC-SHA512_L(c_{parent}, pk_{parent} || index) \cdot G .$$

30

This can be used by external parties to derive various payment addresses as required, avoiding key reuse, whilst reducing rounds of communication and storage.

In general, a HD wallet should generate some hierarchical tree-like structure of private-public key pairs. This provides a high number of key pairs that can all be regenerated from one seed.

DIGITAL SIGNATURES

Figure 4 illustrates an example system 400 for generating a digital signature according to some embodiments of the present invention. In general, the system comprises at least a signing party 401 (i.e. a signature generating party) and a verifying party 402 (i.e. a signature verifying party). In some examples, the system comprises one or more blockchain nodes 104 and/or a second party 403.

Each party 401, 402, 403 operates respective computing equipment (not shown). Each of the respective computing equipment of the respective parties 401, 402, 403 comprises respective processing apparatus comprising one or more processors, e.g. one or more central processing units (CPUs), accelerator processors (GPUs), application specific processors and/or field programmable gate arrays (FPGAs). The respective computing equipment may also comprise memory, i.e. computer-readable storage in the form of a non-transitory computer-readable medium or media. The memory may comprise one or more memory units employing one or more memory media, e.g. a magnetic medium such as a hard disk; an electronic medium such as a solid-state drive (SSD), flash memory or EEPROM; and/or an optical medium such as an optical disk drive. The respective computing equipment may comprise at least one user terminal, e.g. a desktop or laptop computer, a tablet, a smartphone, or a wearable device such as a smartwatch. Alternatively or additionally, the respective computing equipment may comprise one or more other networked resources, such as cloud computing resources accessed via the user terminal (the cloud computing resources comprising resources of one or more physical server devices implemented at one or more sites). It will be appreciated that any act described as being

performed by a party of the system 400 may be performed by the respective computing apparatus operated by that party.

Whilst the present invention is not limited only to use in a blockchain context, the following will be described with the signing party 401 being equated to Alice 103a as described with reference to Figures 1 to 3. That is, in some examples Alice 103a is the signing party 401. In those examples, Bob 103b may be the second party 403. The verifying party 402 will be referred to below as Carol.

In these embodiments, Alice 401 would like to generate a signature and prove to Carol 402 that she generated that signature without re-using the same private key, e.g. without generating two signatures using the same private key.

Alice 401 obtains the message to be signed, e.g. some or part of a blockchain transaction, a document, a contract, etc. Alice 401 may generate the message herself, or Alice 401 may receive the message, e.g. from Bob 403. Alice 401 also obtains an external data item. Alice 401 may already have the external data item, e.g. a name, passport number, public key, etc., or Alice 401 may generate the external data item. For instance, and as will be described in more detail below, the external data item may be a signature (a “second signature”) generated by Alice 401.

Alice 401 generates an ephemeral private key based on a hash of the external data item. For instance, the ephemeral private key may comprise (include or consist of) the hash of the external data item. Put another way, instead of assigning a random value as the ephemeral private key (see the preliminaries section above), the ephemeral private key is now a function of the result of hashing the external data item. The hash function used to generate the hash of the external data item may be any suitable hash function, e.g. SHA256, SHA512, and may comprise applying one or more hash functions multiple times. For instance, the hash function may be a double-hash function, e.g. $SHA256d(x) = SHA256(SHA256(x))$.

Alice 401 then generates first and second components of a signature. The first signature component is generated based on a public key corresponding to the ephemeral private key. That is, Alice 401 generates an ephemeral public key corresponding to the ephemeral private key. A public key has two components (e.g. the x and y values). In examples, the first signature component is based on the first component of the ephemeral public key, e.g. the x value. The second signature component is based on the message to be signed, the ephemeral private key, the first signature component and a private key (a “first private key”). The first private key may be any private key, e.g. a random private key, or as discussed below, a private key that can be linked to an identity-linked public key.

Alice 401 has therefore generated a signature that incorporates an external data item known, at this point, only to Alice 401, e.g. a personal identifier. Note that the external data item itself need not be a secret. It is preferred that the external data item being embedded within the signature is initially kept secret, but this is not essential. For example, the external data item may be a certified public key which itself is known to one or more parties.

Alice 401 may make the signature available to Carol 402. E.g. Alice 401 may send the signature to Carol 402, or Alice may publish or otherwise broadcast the signature. Preferably the signature and the message are sent or published together. Alice 401 may also make the external data item available to Carol 402, either at the same time as the signature or at a different time, e.g. at a later time. The external data item may be made available in the same way as the signature or in a different way. For example, Alice 401 may send the external data item to Carol 402 via a secure communication channel.

As mentioned above, the external data item may be, or at least include, another signature. In that case, Alice 401 obtains a second message and generates a second signature based on at least the second message and a “main private key”. In the broadest examples, “main” is used merely as a label to distinguish from the first private key. That is, the main private key may be any private key owned by Alice 401 other than the first private key used to generate the second signature component of the first signature. In this example, the ephemeral private key used to generate the second signature may be randomly generated, as opposed

to the ephemeral private key used to generate the first signature which is based on the external data item.

Alice 401 may generate at least part of the second message herself. Additionally or
 5 alternatively, Alice 401 may receive or otherwise obtain at least part of the second message from another party, e.g. Carol 402. That is, Carol 402 may send some or all of the second message to Alice 401, or Alice 401 and Carol 402 may have previously agreed upon at least part of the message. For instance, Alice 401 and Carol 402 may have agreed to include an indication of the data and/or time at which the second signature is generated. In some
 10 examples, the second message may comprise or be generated based on the first message. For instance, the second message may comprise the first message with additional data concatenated onto the start or end of the first message.

Alice 401 sends at least the second signature to Carol 402, or Alice 401 may publish the
 15 second signature. If Carol 402 does not already have access to the second message, Alice 401 sends it to Carol 402, or she may publish the second message. Alice 401 may also send a main public key corresponding to the main private key to Carol 402, or at least indicate where Carol 402 may obtain the main public key from, e.g. a location storing a certificate issued by a certificate authority and certifying the main public key as being linked to Alice
 20 401.

Whilst in some examples there is no link between the main private key and the first private key, in other examples the first private key is linked to, i.e. derived from, the main private key. For instance, the main private key may be a master private key of a HD wallet owned by
 25 Alice 401, and the first private key may be a child key derived from the master private key, i.e. the first private key is deterministically linked to the main private key. In other examples, the first private key may be generated based on Alice's main private key and Bob's main private key (or equivalently based on Bob's main public key corresponding to his main private key). For instance, the first private key may be a DH private key, or a private key
 30 derived using the techniques in WO2017/145016, an example of which is provided below. In summary, in WO2017/145016 a common secret is generated based on Alice's main private key and Bob's main public key. Alice 401 may then generate the first private key based on

her main private key and the common secret. Bob's main public key may be linked to his identity, e.g. it may be a certified public key.

5 The ephemeral private key used to generate the first signature is based on at least a hash of the external data item, e.g. a double-hash of the second signature. The ephemeral private key may also be based on a randomly generated salt value, i.e. a value added to the hash of the external data item. Preferably a salt value is used only once, i.e. a different salt value is used to generate different instances of the first signature. In these examples, Alice 401 may generate a third signature based on a third message and the salt value. That is, the salt
10 value is used as a private key for generating the third signature. The third message may be generated based on the first and/or second messages. The third message may be the same as the second message. Alice 401 may send the third signature to Carol 402. If Alice has generated the third message, she may also send it to Carol 402. Or, Carol 402 may have sent the third message to Alice 401 in which case Alice 401 does not need to re-send it to Carol
15 402, although she may choose to do so.

As mentioned above, embodiments of the present invention are not limited to use with the blockchain 150. However in those that are, the first message may be a blockchain transaction. For instance, Alice 401 may sign some or all of a blockchain transaction, e.g.
20 one or more inputs and/or one or more outputs of the transaction. Alice 401 may then include the first signature in an input of the transaction that she has not signed. The transaction may comprise an output that is locked to Bob 403 and/or Carol 402, e.g. the output may be a pay-to-public-key (P2PK) or pay-to-public-key-hash (P2PKH) output locked a public key owned by Bob 403. The second message may comprise the transaction. The
25 second message may also comprise data relating to the blockchain 150, e.g. a current block height of the blockchain at the time the transaction is generated. In these examples, Alice 401 may make the first message available to Carol 402 by transmitting the transaction to the blockchain 150 from which Carol 402 may then access. This is illustrated in Figure 4.

30 Figure 5 illustrates an example sequence of steps which may be taken by Alice 401 to generate a signature according to some embodiments of the present invention. It will be appreciated that some of the steps may be performed in a different order. In step S501,

Alice 401 obtains a first message, e.g. a blockchain transaction. In step S502, Alice 401 obtains an external data item, e.g. a second signature. In step S503, Alice 401 generates an ephemeral private key based on the external data item, e.g. based on a hash of the second signature. In step S504, Alice 401 generates a signature based on the ephemeral private key, and in step S505 she sends at least the external data item to the verifying party Carol 402.

The actions taken by the verifying party, Carol 402, will now be described. Carol 402 would like Alice 401 to prove that Alice generated a signature. Carol 402 obtains a first signature. Alice 402 may send the first signature to Carol 402, or the first signature may be publicly accessible, e.g. recorded on the blockchain 150. If the first signature is included in an input of a blockchain transaction, Carol 402 obtains the first signature by extracting it from the transaction. Carol 402 also obtains a candidate external data item from Alice 401. Here, “candidate” is used to refer to an external data item that Alice 401 claims to have been embedded within the first signature. If that is indeed the case, the candidate external data item is the same as the external data item discussed above. However at this point Carol 402 cannot confirm that to be the case, hence the term “candidate”.

Carol 402 uses the candidate external data item to generate a candidate ephemeral private key, similar to how Alice 401 generated the ephemeral private key. Note that Carol 402 does not have to use exactly the same method as Alice 401. For instance, Alice 401 may use a salt value which Carol 402 does not have access to in order to generate her ephemeral private key. Carol 402 generates a candidate ephemeral public key corresponding to the candidate ephemeral private key, and from that she generates a candidate first signature component. For instance, the candidate first signature component may be the first component (e.g. x value) of the candidate ephemeral public key.

The first signature obtained by Carol 401 comprises a first signature component and a second signature component. To verify that Alice 401 generated the first signature, Carol 402 compares the candidate first signature component to the first signature component. If they match, Carol 402 can be sure that Alice 401 did indeed generate the first signature. That is, in order for the candidate first signature component and the first signature component to be a match, the candidate external data item must be the external data item

used to generate the first signature. Since Alice 401 provided Carol 402 with the candidate external data item, this proves that Alice 401 generated the first signature. This process is illustrated in steps S601 to S605 of Figure 6.

- 5 Carol 402 may also verify that the first signature is a valid signature when validated against the corresponding public key. If the first signature is used to sign a blockchain transaction and that transaction has been recorded on the blockchain, Carol 401 may assume that the first signature is a valid signature (i.e. the transaction would not have been accepted by a blockchain node if the signature was not valid). However Carol 401 may still verify that the
10 unlocking script that is being spent contains a signature check (i.e. to make sure that a blockchain node has performed a signature check on the signature during transaction validation). To do this, Carol 401 may check that the unlocking script of the spent transaction includes an OP_CHECKSIG script.
- 15 As discussed above when describing embodiments of the invention from Alice's perspective, the external data item may itself be a signature, i.e. a second signature. In this case, Carol 402 may obtain a second message, e.g. from Alice 401, and verify that the second signature is a valid signature when verified using a public key provided by or otherwise linked to Alice 401, e.g. a certified public key.
20
- If Alice 401 has used a salt value to generate the ephemeral private key used to generate the first signature, Alice 401 may provide Carol 402 with a public key corresponding to the salt value. Carol 402 may then generate the candidate first signature component based on the "salt public key", e.g. based on a combination of the candidate ephemeral public key
25 and the salt public key. The x-value of said combination may be used to generate the candidate first signature component. An example is provided further below. In these examples, Alice 401 may also provide Carol 402 with a third signature and a third message. Carol 402 may verify that the third signature is a valid signature when verified using the salt public key.

30

The following provides a specific detailed example of the present invention. In particular, the following details a method for cryptographically linking transaction signatures and

private keys using hash functions, as to enable a proof-of-knowledge of private keys. This case involves linking a public key corresponding to someone's identity to a given signature, but the method may be used to include any external data into a signature. It will be appreciated that this is just one example implementation, and that at least some of the following are optional features unique to this particular implementation.

The method, referred to as a "secure key exchange attest method" (SKEAM) is a sequential interaction between a prover, Alice 401, and a verifier, Carol 402. Alice 401 uses a pair of signatures to definitively link a payment transaction to her identity public key whilst ensuring that her identity key does not appear on the blockchain 150 itself. The cryptographic security of the algorithm relies on linking of signatures and ephemeral keys using cryptographic hash functions.

Setup: Alice 401 and Bob 403 have public keys PK_A and PK_B . These are fixed public keys linked to their identity. Alice 401 and Bob 403 use a protocol to derive a set of shared key pairs $PK_{AB1}, \dots, PK_{ABN}$. Alternatively, Alice could use a BIP32 protocol, setting the master key as her 'identity key' and using the (non-hardened) child-key derivation method to derive keys used for signing. The core feature of both protocols is to utilize a key derivation path that links a fixed identity key with a set of transaction signing keys.

$PK_{A1}, \dots, PK_{AN}$ are the transaction keys (i.e. private keys used to sign transactions and public keys to validate the signatures) and are derived by adding a common secret with a respective identity key (see below).

A simplified example of a shared secret key derivation based on Diffie-Hellman uses the following calculations

$$\begin{aligned} A &= sk_A \cdot PK_B = sk_B \cdot PK_A, \\ sk_{AB1} &= H(A), \\ PK_{AB1} &= sk_{AB1} \cdot G. \end{aligned}$$

where $H(\cdot)$ is a hash algorithm and A is the shared Diffie-Hellman key that only Alice 401 and Bob 403 can compute. Alice 401 and Bob 403 can then use the established shared secret integer, sk_{AB1} , to calculate transaction signing keys:

$$\begin{aligned} \text{Alice: } PK_{A1} &= PK_A \oplus PK_{AB1} \\ sk_{A1} &= sk_A + sk_{AB1} \end{aligned}$$

$$\begin{aligned} \text{Bob: } PK_{B1} &= PK_B \oplus PK_{AB1} \\ sk_{B1} &= sk_B + sk_{AB1} \end{aligned}$$

Here, private key addition is modulo n , and public key ‘addition’ uses the secp256k1 group operation $\oplus$. The homomorphic properties of elliptic curve groups permit the relationship between private keys and public keys to hold under addition.

The following describes how to set up a signature that can be used for attestation. In the following the message to be signed is a bitcoin transaction, but this does not need to be the case.

Step 1: Alice 401 creates an unsigned transaction message $Tx1'$ paying to Bob’s transaction public key (or address generated by hashing Bob’s public key). An example transaction is shown schematically below.

$TxID1'$			
Inputs		Outputs	
Value	ScriptSig	Value	ScriptPubKey
1 BSV		0.99 BSV	OP_DUP OP_HASH160 < $H_{160}(PK_{B1})$ > OP_EQUALVERIFY OP_CHECKSIG

Step 2: Alice 401 creates a signature (the “second signature”) using her identity key and the transaction message, M (the “second message”). In this particular example, the message is

composed of the serialized transaction bytes concatenated with the current chain tip block height. The block height is included as an optional feature of the message.

$$M = Tx1' || block_hash,$$

$$Sig_{PK_A}(M) = [r_A, s_A].$$

- 5 Note that r_A is a one-time value derived using a random ephemeral key. The message M and signature $[r_A, s_A]$ now have three source components: unsigned transaction data, block hash (timestamp) and a random ephemeral key.

- 10 Step 3a: Alice 401 uses r_A, s_A and a one-time random salt value, $w \in Z_n$ to create an ephemeral key

$$k_{A1} = SHA256d([r_A, s_A]) + w \bmod n,$$

$$r_{A1} = [k_{A1} \cdot G]_x \bmod n,$$

where $SHA256d(x) = SHA256(SHA256(x))$.

- 15 Step 3b: Alice 401 computes the public value, W and signature using w

$$Sig_w(M), \quad W = w \cdot G.$$

Step 4: The ephemeral key generated in Step 3a is used to generate the signature

$$s_{A1} = k_{A1}^{-1}(H(Tx1') + sk_{A1} \cdot r_{A1}),$$

- 20 where $PK_{A1} = sk_{A1} \cdot G$.

Step 5: $[r_{A1}, s_{A1}]$ now represents a valid signature (the “first signature”) for $Tx1$ when verified against PK_{A1} :

$$Sig_{PK_{A1}}(Tx1') = [r_{A1}, s_{A1}].$$

The table below schematically illustrates the final state of the transaction. As r_{A1} has been derived from a signature generated by Alice's identity key, Alice's identity key is now embedded into the signature for $Tx1$.

$TxID1$			
Inputs		Outputs	
Value	ScriptSig	Value	ScriptPubKey
1 BSV	$< [r_{A1} \ s_{A1}] > \ < PK_{A1} >$	0.99 BSV	OP_DUP OP_HASH160 $< H_{160}(PK_{B1}) >$ OP_EQUALVERIFY OP_CHECKSIG

5

Assume that Carol (a third party) 402 wants to verify that Alice 401 was the creator of $Tx1$ and provably link signatures from sk_{A1} and sk_A . Alice 401 can now provide proof that she generated and signed $Tx1$ without having to re-use sk_{A1} .

- 10 Step 1: Carol 402 obtains $Tx1$ and checks that the transaction is a valid bitcoin transaction (i.e. is valid according to network consensus rules). She extracts $[r_{A1} \ s_{A1}]$ from the ScriptSig of $Tx1$.

Step 2a: Alice 401 shares M and $Sig_{PK_A}(M)$ from Step 2 above.

15

Step 2b: Alice shares W and $Sig_W(M)$ from Step 3b above. Carol 402 may wish to create a different (random) message for Alice 401 to sign using w .

Step 3: Carol 402 verifies the following three conditions.

20

- 1) $Sig_{PK_A}(M)$ is a valid ECDSA signature for M when verified using PK_A
- 2) The last 32 bytes of M is a valid block hash

$$M[\text{len}(M) - 32 : \text{len}(M)] = \text{block_hash}.$$

25

Carol 402 may wish to impose additional constraints on the block-distance between the block represented by *block_hash* and the block that *Tx1* appears in. This would ensure that Alice 401 created an accurate timestamp for her signature.

- 5 3) The preceding bytes are equal to $Tx1'$ (the unsigned transaction bytes for $Tx1$)

$$M[0 : \text{len}(M) - 32] = Tx1'.$$

Step 4a: Carol 402 computes

$$\begin{aligned} k' &= SHA256d(Sig_{PK_A}(M)), \\ r' &= [k' \cdot G + W]_x \bmod n. \end{aligned}$$

Step 4b: Carol 402 checks that $Sig_W(M)$ is a valid ECDSA signature for M when verified with public value W .

15

Step 5: Carol 402 checks if $r' = r_{A1}$. If so then Alice 401 has proven to Carol 402 that her identity key was used in the signature generation for $Tx1$.

In the scheme above the signing key sk_{AB1} is derived from the identity key sk_A using a common secret. That is:

$$sk_{AB1} = sk_A + sk_{AB}$$

where sk_{AB} is a shared secret between Alice 401 and Bob 403. Alice 401 can prove to Carol 42 that the signing key have this link in one of two ways.

25 Carol 402 may use a zero-knowledge proof of knowledge. Carol 402 knows the public key corresponding to sk_A and sk_{AB1} from the proof above, which are denoted by PK_A and PK_{AB1} correspondingly.

1. Alice 401 sends Carol 402 the public key corresponding to the shared secret sk_{AB} and a random value $y \in_R \{1, \dots, k-1\}$. That is, she sends $X = sk_{AB} \cdot G$ and $Y = r \cdot G$.
2. Carol 402 returns the challenge to Alice $c = hash(x)$.

3. Alice 401 calculates $u = c \cdot sk_{AB} + y$ and sends this back to Carol 402.
4. Carol 402 checks that $u \cdot G = c \cdot X + Y$ and that $PK_{AB1} = PK_A + X$.

If this holds, Carol 402 knows that Alice 401 indeed knows sk_{AB1} . This process is essentially a zero-knowledge proof on a discrete log. The challenge may be some pre-agreed or standard challenge so that step 2 can be skipped resulting in the proof being non-interactive.

Note that knowledge of the salt w may be proved with this method instead of a signature as described in the previous section.

Alternatively Alice 401 may create another signature with sk_{AB} which is verified with $X = sk_{AB} \cdot G$. The only other participant with knowledge of this is Bob 403, but since Alice 401 has also provided a signature with her corresponding private key, one can assume that Alice 401 is giving the proof.

It was mentioned above that BIP32 keys may be used in this scheme. In this case a child key has the general form

$$sk_{child} = sk_{parent} + hash(K_{parent})$$

where K_{parent} is either the public or private key of the parent key sk_{parent} , and the hash function is an HMAC created out of the SHA-512 hash function with some additional random input called a chain code depending on the parent, and a counter so that multiple child keys may be derived.

In this case, assuming that the public key corresponding to the parent key $PK_{parent} = sk_{parent} \cdot G$ is the same as the identity described above, and sk_{child} corresponds to the signing key, the two proofs hold above where instead $X = hash(K_{parent})$. The explicit form of the second term is not important to a verifier, instead that Alice knows the second term is the aim of the proof.

As a final note the signing key does not necessarily need to be derived from an identity key. Including the identity key in the derivation of the ephemeral key is enough to prove a link to a certain signer.

5 CONCLUSION

Other variants or use cases of the disclosed techniques may become apparent to the person skilled in the art once given the disclosure herein. The scope of the disclosure is not limited by the described embodiments but only by the accompanying claims.

10

For instance, some embodiments above have been described in terms of a bitcoin network 106, bitcoin blockchain 150 and bitcoin nodes 104. However it will be appreciated that the bitcoin blockchain is one particular example of a blockchain 150 and the above description may apply generally to any blockchain. That is, the present invention is in by no way limited

15

to the bitcoin blockchain. More generally, any reference above to bitcoin network 106, bitcoin blockchain 150 and bitcoin nodes 104 may be replaced with reference to a blockchain network 106, blockchain 150 and blockchain node 104 respectively. The blockchain, blockchain network and/or blockchain nodes may share some or all of the described properties of the bitcoin blockchain 150, bitcoin network 106 and bitcoin nodes

20

104 as described above.

In preferred embodiments of the invention, the blockchain network 106 is the bitcoin network and bitcoin nodes 104 perform at least all of the described functions of creating, publishing, propagating and storing blocks 151 of the blockchain 150. It is not excluded that there may be other network entities (or network elements) that only perform one or some but not all of these functions. That is, a network entity may perform the function of propagating and/or storing blocks without creating and publishing blocks (recall that these entities are not considered nodes of the preferred bitcoin network 106).

25

30

In non-preferred embodiments of the invention, the blockchain network 106 may not be the bitcoin network. In these embodiments, it is not excluded that a node may perform at least one or some but not all of the functions of creating, publishing, propagating and storing

blocks 151 of the blockchain 150. For instance, on those other blockchain networks a “node” may be used to refer to a network entity that is configured to create and publish blocks 151 but not store and/or propagate those blocks 151 to other nodes.

- 5 Even more generally, any reference to the term “bitcoin node” 104 above may be replaced with the term “network entity” or “network element”, wherein such an entity/element is configured to perform some or all of the roles of creating, publishing, propagating and storing blocks. The functions of such a network entity/element may be implemented in hardware in the same way described above with reference to a blockchain node 104.

10

It will be appreciated that the above embodiments have been described by way of example only. More generally there may be provided a method, apparatus or program in accordance with any one or more of the following Statements.

- 15 Statement 1. A computer-implemented method of generating a digital signature, wherein the method is performed by a signing party and comprises:

obtaining a first message;

generating an ephemeral private key based on at least a hash of an external data item; and

20

generating a first signature comprising first and second signature components, wherein the first signature component is generated based on an ephemeral public key corresponding to the ephemeral private key, and wherein the second signature component is generated based on the first message, the ephemeral private key, the first signature component and a first private key.

25

The external data item may comprise an identifier of the signing party, e.g. a name, address, phone number, national insurance number, passport number, public key, etc.

Preferably the first signature is an ECDSA signature.

30

Statement 2. The method of statement 1, comprising making the external data item and the first signature available to a verifying party for proving that the signing party generated the first signature.

- 5 Statement 3. The method of statement 2, wherein obtaining the first message comprises generating the first message, and wherein the method comprising making the first message available to the verifying party.

For instance, the signing party may transmit the first signature and the first message to the
10 verifying party. Alternatively, the signing party may publish the first signature and the first message, e.g. on the internet, on the blockchain, etc.

Statement 4. The method of any preceding statement, comprising:

- obtaining a second message; and
15 generating a second signature based on at least the second message and a main private key of the signing party, and wherein the external data item comprises the second signature.

The second signature comprises a respective first signature component and a respective
20 second signature component, and wherein each of the first and second signature component is based on a second ephemeral private key, i.e. a random ephemeral private key.

- Statement 5. The method of statement 4, comprising making the second signature and the
25 second message available for proving that the second signature is valid signature for the second message when verified using a main public key corresponding to the main private key.

For instance, the signing party may transmit the second signature and the second message
30 to the verifying party.

Statement 6. The method of statement 4 or statement 5, wherein the second message is generated based on the first message.

The signing party may generate the second message, or the second message may be obtained from another party, e.g. the verifying party.

5

Statement 7. The method of any of statements 4 to 6, wherein the main public key corresponding to the main private key is linked to an identity of the signing party.

Therefore the identity of the signing party is embedded within the first signature.

10

Statement 8. The method of any of statements 4 to 7, wherein the first private key is generated based on at least the main private key.

Statement 9. The method of statement 8, wherein the main private key is a master private key of a hierarchical deterministic key structure, the HD key structure comprised a set of child private keys generated based on the master private key, and wherein the first private key is one of the set of child private keys.

15

Statement 10. The method of statement 8, wherein the first private key is generated based on the main private key and a common secret known to both the signing party and a second party.

20

Statement 11. The method of statement 10, wherein the common secret is generated based on the main private key of the signing party and a main public key corresponding to a main private key of the second party.

25

Statement 12. The method of statement 11, wherein the main public key of the second party is linked to an identity of the second party.

Preferably the second party is distinct from the verifying party.

30

Statement 13. The method of any preceding statement, wherein the hash of the external data item is a double-hash of the external data item.

5 Statement 14. The method of any preceding statement, wherein the second signature component is generated based on a hash or double-hash of the first message.

Statement 15. The method of any preceding statement, wherein the first ephemeral private key is generated based on a random salt value.

10 Statement 16. The method of statement 15, wherein the random salt value is a private key, and wherein the method comprises:

obtaining a third message;

generating a third signature based on at least the random salt value and the third message; and

15 making the third signature, the third message and a public key corresponding to the random salt value available to the verifying party for proving that the third signature is a valid signature for the third message when verified using the public key corresponding to the random salt value.

20 Statement 17. The method of statement 16, wherein the third message comprises the second message.

E.g. the third message may be the same as the second message.

25 Statement 18. The method of any preceding statement, wherein the first message comprises at least part of a blockchain transaction.

The first signature may be included in the blockchain transaction, e.g. in an input of the blockchain transaction.

Statement 19. The method of statement 18, wherein making the first message available to the verifying party comprises transmitting the blockchain transaction to the blockchain network.

- 5 Statement 20. The method of statement 18 or statement 19, wherein the second message comprise data pertaining to the blockchain.

For instance, the data pertaining to the blockchain may comprise a current block height of the blockchain.

10

Statement 21. A computer-implemented method of verifying that a digital signature has been generated by a signing party, wherein the method is performed by a verifying party and comprises:

- obtaining a first signature comprising first and second signature components;
- 15 obtaining a candidate external data item from the signing party;
- generating a candidate ephemeral private key based on a hash of the candidate external data item;
- generating a candidate first signature component based on at least a public key corresponding to the candidate ephemeral private key; and
- 20 verifying that the first signature has been generated by the signing party based on whether the candidate first signature component corresponds to the first signature component.

- Statement 22. The method of statement 21, wherein the candidate external data item is a
25 second signature.

Statement 23. The method of statement 22, comprising:

- obtaining a second message;
- obtaining a main public key corresponding to a main private key of the signing party;
- 30 and
- verifying that the second signature is a valid signature for the second message when verified using the main public key.

Statement 24. The method of any of statements 21 to 23, comprising obtaining a public key corresponding to a random salt value, and wherein the candidate first signature component is generated based on the public key corresponding to the random salt value.

5

Statement 25. The method of statement 24, comprising:

obtaining a third message;

obtaining a third signature; and

verifying that the third signature is a valid signature for the third message when

10 verified using the public key corresponding to the random salt value.

Statement 26. The method of any of statements 21 to 25, wherein the first signature signs a first message, and wherein the method comprises:

obtaining a first public key corresponding to a private key used to generate the first

15 signature; and

verifying that the first signature is a valid signature for the first message when

verified using the first public key.

Statement 27. The method of any of statements 21 to 26, wherein one, some or all of the

20 first, second and third signatures are received from the signing party.

Statement 28. The method of any of statements 21 to 27, wherein one, some or all of the first, second and third messages are received from the signing party.

25 Statement 29. The method of any of statements 21 to 27, wherein one, some or all of the first, second and third messages are generated by the verifying party.

Statement 30. The method of any of statements 21 to 29, wherein the first message comprises at least part of a blockchain transaction.

30

Statement 31. The method of statement 30, wherein obtaining the first message comprises obtaining the blockchain transaction from the blockchain.

Statement 32. The method of any statement 30 or statement 31, wherein obtaining the first signature comprises extracting the first signature from the blockchain transaction.

- 5 Statement 33. The method of statement 23 or any statement dependent thereon, wherein the second message comprises data pertaining to the blockchain, and wherein the method comprises verifying the data pertaining to the blockchain.

- Statement 34. The method of statement 30 or any statement dependent thereon, wherein
10 an input of the blockchain transaction comprises the first signature, and wherein the method comprises:

verifying that an output of a previous blockchain transaction referenced by the input of the blockchain transaction comprises a signature verification script.

- 15 Statement 35. Computer equipment comprising:
memory comprising one or more memory units; and
processing apparatus comprising one or more processing units, wherein the memory stores code arranged to run on the processing apparatus, the code being configured so as when on the processing apparatus to perform the method of any of statements 1 to 34.

20

Statement 36. A computer program embodied on computer-readable storage and configured so as, when run on one or more processors, to perform the method of any of statements 1 to 34.

- 25 According to another aspect disclosed herein, there may be provided a method comprising the actions of the signing party and the verifying party.

According to another aspect disclosed herein, there may be provided a system comprising the computer equipment of the signing party and the verifying party.

30

CLAIMS

1. A computer-implemented method of generating a digital signature, wherein the method is performed by a signing party and comprises:
 - obtaining a first message;
 - generating an ephemeral private key based on at least a hash of an external data item; and
 - generating a first signature comprising first and second signature components, wherein the first signature component is generated based on an ephemeral public key corresponding to the ephemeral private key, and wherein the second signature component is generated based on the first message, the ephemeral private key, the first signature component and a first private key.
2. The method of claim 1, comprising making the external data item and the first signature available to a verifying party for proving that the signing party generated the first signature.
3. The method of claim 2, wherein obtaining the first message comprises generating the first message, and wherein the method comprising making the first message available to the verifying party.
4. The method of any preceding claim, comprising:
 - obtaining a second message; and
 - generating a second signature based on at least the second message and a main private key of the signing party, and wherein the external data item comprises the second signature.
5. The method of claim 4, comprising making the second signature and the second message available for proving that the second signature is valid signature for the second message when verified using a main public key corresponding to the main private key.
6. The method of claim 4 or claim 5, wherein the second message is generated based on the first message.

7. The method of any of claims 4 to 6, wherein the main public key corresponding to the main private key is linked to an identity of the signing party.
8. The method of any of claims 4 to 7, wherein the first private key is generated based on at least the main private key.
9. The method of claim 8, wherein the main private key is a master private key of a hierarchical deterministic key structure, the HD key structure comprised a set of child private keys generated based on the master private key, and wherein the first private key is one of the set of child private keys.
10. The method of claim 8, wherein the first private key is generated based on the main private key and a common secret known to both the signing party and a second party.
11. The method of claim 10, wherein the common secret is generated based on the main private key of the signing party and a main public key corresponding to a main private key of the second party.
12. The method of claim 11, wherein the main public key of the second party is linked to an identity of the second party.
13. The method of any preceding claim, wherein the hash of the external data item is a double-hash of the external data item.
14. The method of any preceding claim, wherein the second signature component is generated based on a hash or double-hash of the first message.
15. The method of any preceding claim, wherein the first ephemeral private key is generated based on a random salt value.

16. The method of claim 15, wherein the random salt value is a private key, and wherein the method comprises:

obtaining a third message;

generating a third signature based on at least the random salt value and the third message; and

making the third signature, the third message and a public key corresponding to the random salt value available to the verifying party for proving that the third signature is a valid signature for the third message when verified using the public key corresponding to the random salt value.

17. The method of claim 16, wherein the third message comprises the second message.

18. The method of any preceding claim, wherein the first message comprises at least part of a blockchain transaction.

19. The method of claim 18, wherein making the first message available to the verifying party comprises transmitting the blockchain transaction to the blockchain network.

20. The method of claim 18 or claim 19, wherein the second message comprise data pertaining to the blockchain.

21. A computer-implemented method of verifying that a digital signature has been generated by a signing party, wherein the method is performed by a verifying party and comprises:

obtaining a first signature comprising first and second signature components;

obtaining a candidate external data item from the signing party;

generating a candidate ephemeral private key based on a hash of the candidate external data item;

generating a candidate first signature component based on at least a public key corresponding to the candidate ephemeral private key; and

verifying that the first signature has been generated by the signing party based on whether the candidate first signature component corresponds to the first signature component.

22. The method of claim 21, wherein the candidate external data item is a second signature.

23. The method of claim 22, comprising:

obtaining a second message;

obtaining a main public key corresponding to a main private key of the signing party;

and

verifying that the second signature is a valid signature for the second message when verified using the main public key.

24. The method of any of claims 21 to 23, comprising obtaining a public key corresponding to a random salt value, and wherein the candidate first signature component is generated based on the public key corresponding to the random salt value.

25. The method of claim 24, comprising:

obtaining a third message;

obtaining a third signature; and

verifying that the third signature is a valid signature for the third message when verified using the public key corresponding to the random salt value.

26. The method of any of claims 21 to 25, wherein the first signature signs a first message, and wherein the method comprises:

obtaining a first public key corresponding to a private key used to generate the first signature; and

verifying that the first signature is a valid signature for the first message when verified using the first public key.

27. The method of any of claims 21 to 26, wherein one, some or all of the first, second and third signatures are received from the signing party.
28. The method of any of claims 21 to 27, wherein one, some or all of the first, second and third messages are received from the signing party.
29. The method of any of claims 21 to 27, wherein one, some or all of the first, second and third messages are generated by the verifying party.
30. The method of any of claims 21 to 29, wherein the first message comprises at least part of a blockchain transaction.
31. The method of claim 30, wherein obtaining the first message comprises obtaining the blockchain transaction from the blockchain.
32. The method of any claim 30 or claim 31, wherein obtaining the first signature comprises extracting the first signature from the blockchain transaction.
33. The method of claim 23 or any claim dependent thereon, wherein the second message comprises data pertaining to the blockchain, and wherein the method comprises verifying the data pertaining to the blockchain.
34. The method of claim 30 or any claim dependent thereon, wherein an input of the blockchain transaction comprises the first signature, and wherein the method comprises:
 - verifying that an output of a previous blockchain transaction referenced by the input of the blockchain transaction comprises a signature verification script.
35. Computer equipment comprising:
 - memory comprising one or more memory units; and
 - processing apparatus comprising one or more processing units, wherein the memory stores code arranged to run on the processing apparatus, the code being configured so as when on the processing apparatus to perform the method of any of claims 1 to 34.

36. A computer program embodied on computer-readable storage and configured so as, when run on one or more processors, to perform the method of any of claims 1 to 34.



Application No: GB2012873.2

Examiner: Contract Unit Examiner

Claims searched: 1-36

Date of search: 28 April 2021

Patents Act 1977: Search Report under Section 17

Documents considered to be relevant:

Category	Relevant to claims	Identity of document and passage or figure of particular relevance
Y	1-36	US2011/213982 A1 (BROWN DANIEL RICHARD L) paragraph [0031] - paragraph [0039]
Y	1-36	US2011/208970 A1 (BROWN DANIEL RICAHRD L ET AL) paragraph [0049] - paragraph [0050], figure 5
Y	1-36	WO2017/145048 A1 (NCHAIN HOLDINGS LTD) paragraph [0013] - paragraph [0014], paragraph [0129] - paragraph [0241]

Categories:

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

Field of Search:

Search of GB, EP, WO & US patent documents classified in the following areas of the UKC^X :

Worldwide search of patent documents classified in the following areas of the IPC

The following online and other databases have been used in the preparation of this search report

International Classification:

Subclass	Subgroup	Valid From
H04L	0009/32	01/01/2006
H04L	0009/08	01/01/2006