

SET SHUFFLING**TECHNICAL FIELD**

- 5 The present disclosure relates to a computer-implemented method of set shuffling, i.e. shuffling a set of data items.

BACKGROUND

- 10 A blockchain refers to a form of distributed data structure, wherein a duplicate copy of the blockchain is maintained at each of a plurality of nodes in a distributed peer-to-peer (P2P) network (referred to below as a “blockchain network”) and widely publicised. The blockchain comprises a chain of blocks of data, wherein each block comprises one or more transactions. Each transaction, other than so-called “coinbase transactions”, points back to
- 15 a preceding transaction in a sequence which may span one or more blocks going back to one or more coinbase transactions. Coinbase transactions are discussed further below. Transactions that are submitted to the blockchain network are included in new blocks. New blocks are created by a process often referred to as “mining”, which involves each of a plurality of the nodes competing to perform “proof-of-work”, i.e. solving a cryptographic
- 20 puzzle based on a representation of a defined set of ordered and validated pending transactions waiting to be included in a new block of the blockchain. It should be noted that the blockchain may be pruned at some nodes, and the publication of blocks can be achieved through the publication of mere block headers.
- 25 The transactions in the blockchain may be used for one or more of the following purposes: to convey a digital asset (i.e. a number of digital tokens), to order a set of entries in a virtualised ledger or registry, to receive and process timestamp entries, and/or to time-order index pointers. A blockchain can also be exploited in order to layer additional functionality on top of the blockchain. For example blockchain protocols may allow for
- 30 storage of additional user data or indexes to data in a transaction. There is no pre-specified limit to the maximum data capacity that can be stored within a single transaction, and

therefore increasingly more complex data can be incorporated. For instance this may be used to store an electronic document in the blockchain, or audio or video data.

Nodes of the blockchain network (which are often referred to as “miners”) perform a distributed transaction registration and verification process, which will be described in more detail later. In summary, during this process a node validates transactions and inserts them into a block template for which they attempt to identify a valid proof-of-work solution. Once a valid solution is found, a new block is propagated to other nodes of the network, thus enabling each node to record the new block on the blockchain. In order to have a transaction recorded in the blockchain, a user (e.g. a blockchain client application) sends the transaction to one of the nodes of the network to be propagated. Nodes which receive the transaction may race to find a proof-of-work solution incorporating the validated transaction into a new block. Each node is configured to enforce the same node protocol, which will include one or more conditions for a transaction to be valid. Invalid transactions will not be propagated nor incorporated into blocks. Assuming the transaction is validated and thereby accepted onto the blockchain, then the transaction (including any user data) will thus remain registered and indexed at each of the nodes in the blockchain network as an immutable public record.

The node who successfully solved the proof-of-work puzzle to create the latest block is typically rewarded with a new transaction called the “coinbase transaction” which distributes an amount of the digital asset, i.e. a number of tokens. The detection and rejection of invalid transactions is enforced by the actions of competing nodes who act as agents of the network and are incentivised to report and block malfeasance. The widespread publication of information allows users to continuously audit the performance of nodes. The publication of the mere block headers allows participants to ensure the ongoing integrity of the blockchain.

In an “output-based” model (sometimes referred to as a UTXO-based model), the data structure of a given transaction comprises one or more inputs and one or more outputs. Any spendable output comprises an element specifying an amount of the digital asset that is derivable from the proceeding sequence of transactions. The spendable output is

sometimes referred to as a UTXO (“unspent transaction output”). The output may further comprise a locking script specifying a condition for the future redemption of the output. A locking script is a predicate defining the conditions necessary to validate and transfer digital tokens or assets. Each input of a transaction (other than a coinbase transaction) comprises a pointer (i.e. a reference) to such an output in a preceding transaction, and may further comprise an unlocking script for unlocking the locking script of the pointed-to output. So consider a pair of transactions, call them a first and a second transaction (or “target” transaction). The first transaction comprises at least one output specifying an amount of the digital asset, and comprising a locking script defining one or more conditions of unlocking the output. The second, target transaction comprises at least one input, comprising a pointer to the output of the first transaction, and an unlocking script for unlocking the output of the first transaction.

In such a model, when the second, target transaction is sent to the blockchain network to be propagated and recorded in the blockchain, one of the criteria for validity applied at each node will be that the unlocking script meets all of the one or more conditions defined in the locking script of the first transaction. Another will be that the output of the first transaction has not already been redeemed by another, earlier valid transaction. Any node that finds the target transaction invalid according to any of these conditions will not propagate it (as a valid transaction, but possibly to register an invalid transaction) nor include it in a new block to be recorded in the blockchain.

An alternative type of transaction model is an account-based model. In this 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 separate to the blockchain and is updated constantly.

SUMMARY

30

Set shuffling refers to a method for multiple parties (e.g. users) to change the order of some items (e.g. cards, tokens, addresses, etc.) without any one party being able to determine the

value or order of the items. This ensures that each player can guarantee other players cannot determine the value of any item and thus all players can interact based on these items on a trust-less peer-to-peer (P2P) basis. Note here that value does not necessarily imply a monetary value. A shuffled set can form the basis of fair interactions (e.g. votes) that typically involve parties selecting or committing to unknown values, which are later revealed and evaluated based on some predefined rules. For example, a shuffled set can be used for mental poker protocols. Mental poker refers to a set of cryptographic problems or interactions that can be carried out fairly between parties on a trust-less basis e.g., without the need for a third officiating party. It is commonly explained in terms of card games or flipping a coin and verifying, but has wider, non-game applications.

According to one aspect disclosed herein, there is provided a computer-implemented method of shuffling an ordered set of data items for enabling pseudorandom selection of one or more data items between a group of two or more parties. The method is performed by a first party of the group and comprises generating an ordered set of shuffle keys, wherein each shuffle key is a respective elliptic curve point. Said generating comprises, for each respective data item in the ordered set: either deriving the respective shuffle key based on a respective private key, or generating the respective shuffle key and deriving the respective private key based on the respective shuffle key. The method further comprises deriving, based on the respective private key, a respective public key, and mapping the respective public key to the respective data item. The method further comprises encrypting each respective shuffle key with a first scalar value to obtain a respective first encrypted shuffle key; placing each respective first encrypted shuffle keys in an ordered set, wherein the order of the respective first encrypted shuffle keys is shuffled compared with the order of the respective shuffle keys in the ordered set of respective shuffle keys; and sharing the ordered set of respective first encrypted shuffle keys with at least a second party of the group.

The present application discloses a novel set-shuffling protocol which utilises the properties of elliptic curve points. A group comprises multiple parties. Each party is directly or indirectly associated with an index representing an order in which the parties are involved in the protocol, i.e. an order in which particular parties perform particular actions. The

protocol involves a first party generating a plurality of shuffle keys (elliptic curve points). Each shuffle key is used to derive a public key which is then mapped to a data item. A data item can take any form of representation of an element, such as a card, token, payment, etc. The shuffle keys themselves may be first generated, from which a private key is
5 generated (e.g. extracted) for deriving a corresponding public key. Alternatively, a private key may be first generated for deriving the shuffle key and the public key. According to the protocol, a selection of a data item (element) is made by obtaining the private key corresponding to a particular public key.

- 10 The shuffle keys are placed in an ordered set (i.e. sequence). Each shuffle key is encrypted with a scalar value. The encrypted shuffle keys are then shuffled, i.e. the order of the encrypted shuffle keys is changed compared to the initial order of the non-encrypted shuffle keys, and the shuffled order of encrypted shuffle keys is shared with at least a second party. Due to being encrypted with the scalar value, the second party cannot determine which
15 shuffle key corresponds to which public key (i.e. which shuffle key is used to derive which public key). Thus the second party cannot determine which data items are mapped to which public keys. Moreover, since the shuffle keys are encrypted, the second party cannot derive any private keys necessary for selecting a data item (element). The set of data items can therefore be considered as shuffled since the order or value cannot be determined.

20

- According to one aspect disclosed herein, there is provided a computer-implemented method for enabling pseudorandom selection of one or more data items between a group of two or more parties. The method is performed by a party of the group and comprises receiving an ordered set of encrypted shuffle keys, wherein the ordered set of encrypted
25 shuffle keys is generated by generating an ordered set of shuffle keys, wherein each shuffle key is a respective elliptic curve point, and wherein said generating comprises, for each respective data item in the ordered set, deriving the respective shuffle key based on a respective private key, or generating the respective shuffle key and deriving the respective private key based on the respective shuffle key, encrypting each respective shuffle key with
30 a first scalar value to obtain a respective first encrypted shuffle key; placing each respective first encrypted shuffle key in an ordered set, wherein the order of the respective first encrypted shuffle keys is shuffled compared with the order of the respective shuffle keys in

the ordered set of shuffle keys; and optionally, performing one or more further rounds of a next party receiving the ordered set of first encrypted shuffle keys, encrypting each first encrypted shuffle keys with a respective scalar value to obtain a respective next encrypted shuffle key, placing each respective next encrypted shuffle key in a next ordered set,
5 wherein the order of the respective next encrypted shuffle keys is shuffled compared with the order of the respective shuffle keys in the received ordered set of shuffle keys, and encrypting each first or next encrypted shuffle key with a respective scalar value to obtain a respective next encrypted shuffle key; placing each respective next encrypted shuffle key in a next ordered set, wherein the order of the respective next encrypted shuffle keys is
10 shuffled compared with the order of the respective shuffle keys in the received ordered set of shuffle keys; and sharing the next ordered set of next encrypted shuffle keys with one or more parties of the group.

The second party performs similar steps to the first party. That is, the second party further
15 encrypts each encrypted shuffle with a respective scalar value, and then shuffles the order of further encrypted shuffle keys compared to the order of encrypted shuffle keys that was received from the first party. Depending on the number of parties in the group, the second party may send the further encrypted shuffle keys to a third party (if there are three or more parties), or send the further encrypted shuffle keys to the first party (if there are only
20 two parties).

In embodiments, certain stages of the set shuffling procedure may be attested to by submitting commitments to the blockchain. For instance, the first party commit to the mapping of data items to public keys, the order of shuffle keys, and the order of encrypted
25 shuffle keys. Submitting such commitments to the blockchain has several advantages, including providing a means for off-chain settlements, a proof against cheating, and debugging (error spotting).

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:

5 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,

10 Figure 3 schematically illustrates an example system for implementing a set shuffling protocol,

Figure 4 schematically illustrates an example mental poker protocol,

15 Figure 5 shows an example procedure for a generalised mental poker protocol,

Figure 6 shows one example relationship between private keys, public keys and shuffle keys,

20 Figure 7 shows another example relationship between private keys, public keys and shuffle keys,

Figure 8 shows an outline of shuffling and point multiplication phase for Alice with scalar a , Bob with scalar b and Charlie with scalar c ,

25 Figure 9 shows an outline of discrete element point multiplication,

Figure 10 shows an example object mapping procedure,

30 Figure 11 shows an example element shuffling and point multiplication procedure with blockchain attestation,

Figure 12 shows an example element mapping procedure performed by a board executive to commit 100 public addresses to the blockchain,

Figure 13 shows an outline of a shuffling and point multiplication phase for each of 100
5 voting members and the executive,

Figure 14 shows an example distinct element point multiplication procedure for all 100 board members and the executive, and

10 Figure 15 shows an example of secret commitments for dispute resolution over voting.

DETAILED DESCRIPTION OF EMBODIMENTS

1. EXAMPLE SYSTEM OVERVIEW

15

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
20 a near-complete graph. Each blockchain node 104 is therefore highly connected to other blockchain nodes 104.

Each blockchain node 104 comprises computer equipment of a peer, with different ones of
25 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
30 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 blockchain 150 comprises a chain of blocks of data 151, wherein a respective copy of
5 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
10 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
15 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.

20

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
25 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
30 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 152j, the (or each) input comprises a pointer referencing the output of a preceding transaction 152i in the sequence of transactions, specifying that this output is to be redeemed or “spent” in the present transaction 152j. Spending or redeeming does not necessarily imply transfer of a financial asset, though that is certainly one common application. More generally spending could be described as consuming the output, or assigning it to one or more outputs in another, onward transaction. 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
5 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
10 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
15 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
20 condition defined in the output of the preceding transaction 152i which the new transaction
spends (or “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
25 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
30 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 (or “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 spends or 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
5 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
10 (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,
15 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.

20 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
25 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
30 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.

5

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.

10

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.

15

20

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 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.

25

30

The instance of the client application or software 105 on each computer equipment 102 is 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 is a public facility which provides trust in transactions in part through its public visibility).

5 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 blockchain network 106. The transaction protocol and the node protocol correspond to one
10 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.

15 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
20 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
25 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
30 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.

5

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.

30

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.

2. 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.

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_{X_I} ”. 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 " TX_0 " in Figure 2. TX_0 and TX_1 are just arbitrary labels. They do not necessarily mean that TX_0 is the first transaction in the blockchain 151, nor that TX_1 is the immediate next transaction in the pool 154. TX_1 could point back to any preceding (i.e. antecedent) transaction that still has an unspent output 203 locked to Alice.

The preceding transaction TX_0 may already have been validated and included in a block 151 of the blockchain 150 at the time when Alice creates her new transaction TX_1 , 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 TX_0 and TX_1 could be created and sent to 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.

- 5 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
- 10 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 Tx_0 comprises a locking script
- 15 [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 Tx_1 comprises a pointer pointing back to Tx_0 (e.g. by means of its transaction ID, $TxID_0$, which in embodiments is the hash of the whole
- 20 transaction Tx_0). The input 202 of Tx_1 comprises an index identifying $UTXO_0$ within Tx_0 , to identify it amongst any other possible outputs of Tx_0 . The input 202 of Tx_1 further comprises an unlocking script <Sig P_A > 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
- 25 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 Tx_1 arrives at a blockchain node 104, the node applies the node protocol. This comprises running the locking script and unlocking script together to check
- 30 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:

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

- where “||” represents a concatenation and “<...>” means place the data on the stack, and
- 5 “[...]” 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 of Alice, as included in the locking script in the output of TX_0 , to authenticate that the unlocking script in the input of TX_1 contains the signature of Alice signing the expected
- 10 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).
- 15 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,
- 20 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.
- 25 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,
- 30 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 T_{X_I} will be invalid even if all the other conditions are met. Hence the blockchain node 104 also needs to check whether the referenced UTXO in the preceding transaction T_{X_O} 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 a UTXO has been spent is whether it has already formed a valid input to another valid transaction in the blockchain 150.

10

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.

15

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 $UTXO_O$ in T_{X_O} can be split between multiple UTXOs in T_{X_I} . Hence if Alice does not want to give Bob all of the amount defined in $UTXO_O$, she can use the remainder to give herself change in a second output of T_{X_I} , or pay another party.

20

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, T_{X_O} 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 $UTXO_O$ is

30

the only input to Tx_I , and Tx_I has only one output $UTXO_I$. If the amount of the digital asset specified in $UTXO_O$ is greater than the amount specified in $UTXO_I$, then the difference may be assigned (or spent) by the node 104 that wins the proof-of-work race to create the block containing $UTXO_I$. 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.

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.

10 3. SIDE CHANNEL

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 15 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 20 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, 25 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 30 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
 5 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.

4. CRYPTOGRAPHIC PRINCIPLES

10 4.1 Encryption with Elliptical Curve (EC) Arithmetic

Encryption is a term for scrambling data from one form to another, in such a way that only authorized parties can understand the data e.g., convert back to its original form. In this disclosure, encryption is defined in terms of elliptical curve (EC) arithmetic. Given some point P on an EC, say secp256k1 as in Bitcoin, scalar multiplication can be performed on that
 15 point to obtain a new point. Multiplying point P by scalar a forms a new point on the EC, P^* , defined by:

$$P^* = aP$$

Where point multiplication by scalar a , is point addition of P to itself a number of times, as
 20 such:

$$P^* = a \cdot P = aP = P + P + P \dots + P$$

Given P^* , it is computationally infeasible to derive point P or scalar a under an EC, such as secp256k1. This is described as the EC Discrete Logarithmic Problem. However, it is trivial to
 25 compute P , given P^* and scalar a . This is done by performing point multiplication of the inverse of a , as such:

$$P = a^{-1}P^*$$

Where, the inverse scalar is defined by:

30
$$a^{-1}a = 1$$

Throughout this disclosure, encryption is described as the multiplication of some EC point P by some scalar a . Similarly, decryption is described as point multiplication of the result by some inverse a^{-1} .

- 5 Encryption using point multiplication has two properties that make it a viable protocol for set shuffling: it is commutative and homomorphic. Commutative encryption describes the property that EC points can be multiplied by different scalars in any order and still yield the same result, as such:

$$abcP = bcaP = cabP = \dots$$

10

This also applies to the decryption process, using the inverse scalars, as such:

$$P = a^{-1}b^{-1}c^{-1}abcP = b^{-1}a^{-1}c^{-1}cabP = \dots$$

- Homomorphic encryption describes the property that encrypted data can be further
15 encrypted and decrypted using the same decryption keys. Here, it means that EC points once multiplied, can be further multiplied, and decrypted in the same way. This allows EC points to be encrypted across multiple parties, in any order and decrypted provided the corresponding decryption keys. This is particularly useful for mental poker and set shuffling.

20 4.2 Mental Poker

Mental poker refers to a set of cryptographic problems or interactions that can be carried out fairly between parties on a trust-less basis e.g., without the need for a trusted 3rd party. It should be appreciated that mental poker is not limited to games and instead has wider applications where (pseudo)random selections and/or outcomes are required.

25

- Mental poker typically makes use of a shuffling protocol which is a method for multiple users to encrypt and change the order of some items (e.g. cards, tokens, etc.) without any one user being able to determine the order or value of the items afterwards. As each party adds their own encryption key and then shuffles, they can be assured that the other parties
30 cannot determine the values. To this end, homomorphic encryption methods can be used across multiple users on the same object to efficiently encrypt and decrypt an object.

Figure 4 shows an outline of the various phases of an example mental poker protocol. As shown, mental poker typically makes use of the follow phases:

1. Interaction outline and element mapping:
 - a. Agree to the rules,
 - 5 b. Map elements as a set e.g., 'Ace', 'King' and 'Queen' are elements 1, 2 and 3 within the set.
 - c. Map these elements to some public keys: P_1, P_2 and P_3 .
2. Encrypt and shuffle:
 - a. Alice encrypts the set with a secret key a ,
 - 10 b. Alice then changes the sequence of elements within the set,
 - c. Alice then passes the encrypted and shuffled set to Bob,
 - d. Bob repeats a)-c) with scalar b and passes the set to Claire,
 - e. Claire repeats a)-c) with scalar c and then shares the set with Alice and Bob.
- 15 3. Start interaction:
 - a. Now that there is a shuffled and encrypted set (that typically simulates a card deck) each player can select an element, or elements, from the set at 'random',
 - b. We consider this random as no player can know what any of the
 - 20 encrypted values represent,
4. Decryption:
 - a. After selecting elements, each party reveals their decryption keys: a^{-1}, b^{-1} and c^{-1} respectively,
 - b. Now each player can reveal their original public keys: P_1, P_2 and P_3 .
- 25 5. Outcome:
 - a. After decrypting the element each party can reveal which element/card they have: 'Ace', 'King or 'Queen'.

5. SET SHUFFLING

30

Figure 3 illustrates an example system 300 for implementing the set-shuffling protocol described herein. The system 300 comprises a plurality of parties 103. Three parties (Alice

103a, Bob 103b and Claire 103c) are shown in Figure 3 but in general there may be any number of parties 103. Each party 103 operates computer equipment 102 like Alice 103 and Bob 103b and is configured to perform some or all of the actions described above as being performed by Alice 103 and/or Bob 103b. In particular, each party 103 is configured to
5 communicate with one another, e.g. via respective side channels 107. Each party 103 is also configured to submit blockchain transactions to a blockchain network 106.

The set shuffling protocol is primarily used to change the order of some data items representing objects, elements, etc. The objects etc. may or may not exist in the real-world
10 as physical objects. The objects may instead exist in a virtual-world. The objects may be purely computer-implemented. In general the data items may represent any objects. For instance, the objects may be playing cards or tokens which give the token holder a right to perform a particular action.

15 A first party, e.g. Alice 103a, obtains (e.g. defines) a set of data items. The data items have an initial order. The initial order may be deliberately chosen. For instance, the objects may be playing cards which are ordered from Ace to King. Alternatively, the initial order may be chosen at random.

20 Alice 103a generates an ordered set of shuffle keys, one per data item. Each shuffle key is a unique elliptic curve point. Any suitable elliptic curve may be chosen, such as secp256k1. The shuffle key will later be used to derive a private key corresponding to a public key which has been mapped to a particular data item. There are several ways in which a shuffle key may be generated. One option is to first generate a private key, and then derive the shuffle
25 key from the private key. For example, the private key may be chosen to be the x-coordinate of an elliptic curve point, with the y-coordinate being derived from the x-coordinate. The skilled person will be familiar with how to derive the y-coordinate of an elliptic curve from the x-coordinate. A public key is then generated from the private key and mapped to a data item. Alice 103a performs these actions for each data item. That is, for each data item a
30 private key is chosen and used to derive a shuffle key and a public key.

An alternative is to randomly choose an elliptic curve point as a shuffle key. A private key is then derived from the shuffle key. For instance, the private key may be generated by hashing a concatenation of the coordinates of the shuffle key. As another example, the private key may be chosen as the x-coordinate of the shuffle key. A public key is then
5 generated from the private key and mapped to a data item. Again, Alice 103a performs these actions for each data item.

Any suitable method for generating a shuffle key so long a private key can be derived from the shuffle key. Preferably, the method also allows the shuffle key to be derivable from the
10 private key.

The mapping of the public keys to the data items may be made available to the group. For example, Alice 103a may send the mapping to each other party. As another example, Alice 103 may publish the mapping on a publicly available source, such as the blockchain 150. In
15 some examples, in addition to or instead of sending and/or publishing the mapping itself, Alice 103a may send and/or publish a commitment of the mapping. For example, the commitment may be a hash of the mapping. Herein, publishing data on the blockchain is taken to mean storing the data in a blockchain transaction and submitting the blockchain transaction to the blockchain network 106.

20

The order of the shuffle keys corresponds to the order of the data items and the order of the public keys. That is, a first shuffle key in the ordered set of shuffle keys is generated for the first data item in the ordered set of data items which is mapped to a first public key in the ordered set of public keys. Similarly, a second shuffle key in the ordered set of shuffle
25 keys is generated for the second data item in the ordered set of data items which is mapped to a second public key in the ordered set of public keys, and so on. In other words, the shuffle keys, data items, and public keys are indexed the same. E.g. a first data item has a first index, the public key mapped to that first data item has the same first index, and the shuffle key used to derive the public key has the same first index.

30

Having generated the ordered set of shuffle keys, Alice 103a encrypts each shuffle key with a first scalar value, as described in section 4.1 above. Each shuffle key thus becomes an

encrypted shuffle key, encrypted with the first scalar value. Then, the order of the encrypted shuffle keys is shuffled (i.e. changed) such that the positions of one or more of the encrypted shuffle keys differs before and after encryption. Alice 103a then sends the ordered set of encrypted shuffle keys to Bob 103b. These encrypted shuffle keys are also
5 referred to as “first encrypted shuffle keys”, referring to the fact that they correspond to the shuffle keys that have been encrypted by the first party Alice 103a. Equivalently, the “first” indicates that the shuffle keys have been encrypted with the first scalar value.

Alice 103a may store a commitment of the ordered set of shuffle keys on the blockchain.
10 That is, Alice 103a sends a blockchain transaction to the blockchain network 106 which includes a commitment to the initial order of the shuffle keys. The commitment may be a hash of the shuffle keys. Additionally or alternatively, Alice 103a may store a commitment (e.g. hash) of initial order of the first encrypted shuffle keys on the blockchain. That is, Alice 103a send a blockchain transaction to the blockchain network 106 which includes a
15 commitment to the initial order of the first encrypted shuffle keys. Alice 103a may also store a commitment to the shuffled order of the first encrypted shuffle keys in a blockchain transaction and send that blockchain transaction to the blockchain network 106. The commitments may be stored in the same or different transactions.

20 The second party, e.g. Bob 103b, receives the ordered set of first encrypted shuffle keys from Alice 103a. Bob 103b uses a second scalar value to encrypt each of the first encrypted shuffle keys. The resulting shuffle keys will be referred to as “second encrypted shuffle keys”. Bob 103b then shuffles the set of second encrypted shuffle keys compared to give an ordered set of second encrypted shuffle keys, where the order of the second encrypted
25 shuffle keys differs from the order of the first encrypted shuffle keys. Like Alice 103a, Bob 103b may also commit to an initial order of the second encrypted shuffle keys. Additionally or alternatively, Bob 103b may also commit to the shuffled order of the second encrypted shuffle keys.

30 If Alice 103 and Bob 103b are the only parties in the group, Bob 103b sends the ordered set of second encrypted shuffle keys to Alice 103a. In this case, the second encrypted shuffle keys are “final encrypted shuffle keys”, where “final” refers to the shuffle keys having been

encrypted by each party 103 with a respective scalar value. If the group includes one or more additional parties, e.g. Claire 103c, Bob 130b sends the ordered set of second encrypted shuffle keys to Claire 103c. Claire 103c then performs the same actions as Bob 103b except for using a third scalar value for encryption. The process of encrypting, shuffling
5 and sharing the encrypted shuffle keys continues until each party 103 has encrypted the shuffle keys with a respective scalar value. This process is scalable to as many parties as necessary. The encrypted shuffle keys that have been encrypted by each party are referred to as “final encrypted shuffle keys” and are sent to Alice 103a. In other words, Alice 103a received an ordered set of encrypted shuffle keys, where each shuffle key has been
10 encrypted with n scalar values, where n is the number of parties in the group.

In some embodiments, the final encrypted shuffle keys undergo no further encryption and are instead used by the parties to choose (i.e. select) an object or element mapped to the corresponding public keys. Since no party knows which encrypted shuffle keys correspond
15 to which objects (since all parties have encrypted the shuffle keys), the parties can make random selections of the objects, as is now described. Alice 103a selects one of the final encrypted shuffle keys and makes a commitment to the selection. This commitment may take any suitable form. In some examples, committing to a selection simply means communicating the selection to the other parties 103. In some examples, committing to a
20 selection involves hashing the selected final encrypted shuffle key and communicating the hash to the other parties. Communicating the commitment may be done by including the commitment in a blockchain transaction and submitting the transaction to the blockchain network 106. The selection is thus made immutable and irrefutable.

25 In these embodiments, Alice 103a reveals (e.g. sends), to the other parties 103, an inverse of the first scalar value. This may take place once each party has committed to a selection. Similarly, the other parties reveal the inverses of the scalar values used by those parties 103. For example, Bob 103b reveals an inverse of the second scalar value, and Claire 103c reveals an inverse of the third scalar value. Alice 103a uses each inverse to decrypt the final
30 encrypted shuffle key that she has selected. Decrypting the selected final encrypted shuffle key reveals one of the shuffle keys in its raw form, i.e. an x-coordinate and y-coordinate of an elliptic curve point. Recall from above that each shuffle key is used to derive a private

key. The method for deriving a private key from a shuffle key is known to and agreed to by each party. Alice 103a thus derives the private key (e.g. the x-coordinate of the shuffle key). Alice 103a then derives a public key from the private key. Using the mapping of public keys to data items, Alice 103a determines which data item is mapped to the public key derived from the selected shuffle key. Alice 103a has therefore determined which object, element, etc. she has selected.

The other parties 103 perform similar actions to Alice 103a. For example, Bob 103b commits to selecting a different one of the final encrypted shuffle keys, obtains the inverse values from the other parties 103 and decrypts the final encrypted shuffle key using the inverse values. Having obtained an unencrypted shuffle key, Bob 103b derives a private key and then a public key. Bob 103b determines which data item and therefore ultimately which object is mapped to the public key. Note that in these embodiments all parties may be required to commit to their selection before any party reveals their inverse value to the other parties for decryption.

In alternative embodiments, the final encrypted shuffle keys undergo further encryption by the parties 103. In these embodiments, the final encrypted shuffle keys will be referred to as "first-round final encrypted shuffle keys", referring to the notion that these keys will undergo a second round of encryption. Similarly, the scalar values that were used by the parties to generate the first-round final encrypted shuffle keys will be referred to as "first-round scalar values". Upon receiving the first-round final encrypted shuffle keys, Alice 103a uses the inverse of her first-round scalar value (i.e. an inverse of the first scalar value) to decrypt each of the first-round final encrypted shuffle keys. The resulting keys, which are still encrypted with each of the other parties' first-round scalar values, will be referred to as "first target encrypted shuffle keys". Alice 103a then encrypts each first target encrypted shuffle key with a different second-round scalar value. The first target encrypted shuffle keys, once encrypted with a different second-round scalar value, will be referred to as "second-round first encrypted shuffle keys". "Second-round" refers to the keys being generated as part of a second encryption stage, whilst "first" refers to the keys being generated by the first party 103a. Alice 103a then shares the set of second-round first

encrypted keys with Bob 103b. Note that in contrast to the first-round encryption, the order of the second-round encrypted shuffle keys is not changed, i.e. the keys are not shuffled.

The other parties 103 perform similar actions to Alice 103a. For example, Bob 103b receives
5 the set of second-round first encrypted shuffle keys from Alice 103a, and uses an inverse of his first-round scalar value (i.e. an inverse of the second scalar value) to decrypt the second-round first encrypted shuffle keys, producing a set of second target encrypted shuffle keys. Bob 103a then encrypts each second target encrypted shuffle key with a different second-round second scalar value, generating a set of second-round second encrypted shuffle keys,
10 which he sends to Claire 103c (if the group comprises three or more parties 103), or to Alice 103a. Each party 103 performs the same process of decrypting with a first-round scalar value and then encrypting with a plurality of different second-round scalar values, until Alice 103a receives a set of second-round final encrypted shuffle keys from the last party. For example, Bob 103b may be the last party in which case the second-round second encrypted
15 shuffle keys are the second-round final encrypted shuffle keys.

Alice 103a then selects one of the second-round final encrypted shuffle keys and makes a commitment to the selection. As before, committing to a selection may mean
communicating the selection to the other parties 103. In some examples, committing to a
20 selection involves hashing the selected second-round final encrypted shuffle key and communicating the hash to the other parties. Communicating the commitment may be done by including the commitment in a blockchain transaction and submitting the transaction to the blockchain network 106.

25 In these embodiments, Alice 103a obtains (e.g. receives), from the other parties 103, a respective (indexed) inverse of each of the second-round scalar value used by the other parties to generate the selected final-round encrypted shuffle key. For instance, if Alice 103a selects the key occupying the third position in the ordered set of second-round final encrypted shuffle keys, Bob 103, Claire, etc. send to Alice an inverse of the respective
30 second-round scalar value that they used to encrypt the key occupying the third position in the ordered set of second round encrypted shuffle keys which they received from the previous party.

Alice 103a uses each inverse (along with an inverse of the second-round scalar value used by Alice 103a) to decrypt the selected second-round final encrypted shuffle key. Decrypting the selected second-round final encrypted shuffle key reveals one of the shuffle keys in its raw form, i.e. an x-coordinate and y-coordinate of an elliptic curve point. Alice 103a then derives the private key (e.g. the x-coordinate of the shuffle key). Alice 103a then derives a public key from the private key. Using the mapping of public keys to data items, Alice 103a determines which data item is mapped to the public key derived from the selected shuffle key. Alice 103a has therefore determined which object, element, etc. she has selected.

10

When another party (e.g. Bob 103b) commits to a selection of one of the second-round final encrypted shuffle keys, Alice 103a sends to Bob 103b an inverse of the second-round scalar value used by Alice 103a to encrypt the selected second-round final encrypted shuffle key.

15 The other parties 103 perform similar actions to Alice 103a in order to decrypt a selected second-round final encrypted shuffle key and determine which object, etc. has been selected.

In some examples, each of the public keys that are mapped to a data item may be used to lock a respective transaction output, i.e. a UTXO. One or more of the UTXOs may belong to (i.e. be part of) the same transaction. Similarly, one or more of the UTXOs may belong to different transactions. Alice 103a or a different one of the parties 103 may generate the UTXOs. Alternatively, a different entity (e.g. a trusted third party) may generate the UTXOs. In these examples, deriving the private key and public key from a selected shuffle key may be used to identify and unlock one of the UTXOs. For instance, the derived private key may be used to generate a signature for unlocking the UTXO. As an example, the UTXO may be locked by a pay-to-public-key-hash locking script which requires a public key and a corresponding signature to be provided in an unlocking script of the spending transaction.

30 6. SET SHUFFLING USING BLOCKCHAIN

This section describes a specific example of the general set shuffling protocol described above. Some of the features described in this section are optional. Moreover, the following protocol will be described with reference to Bitcoin and the secp256k1 elliptic curve, but other blockchains and other elliptic curves may be used instead.

5

6.1 Outline and element mapping

Figure 5 illustrates the architecture of an example mental poker protocol. The initial stages of any set shuffling interaction will involve the outline of the protocol related to the interaction. These could follow established rules that exist for common interactions. The rules could also be more unique requiring a rigorous outline for such details. It may be important to communicate such details securely. Using Diffie-Hellman type procedures, it is assumed that the involved parties can communicate and exchange values securely to one and other, over an encrypted channel.

Element mapping considers the representation of real-world objects such as card values, tokens, etc., as usable elements for an online game. One option is to represent real-world objects in string format where, for example, the three of hearts card could be represented as a string, as such:

'3Hearts'

20

This method may be used to represent any real-world data which is formed from multiple elements of a single set, such as options within a roulette table, top trump cards or voting tokens. Herein, objects will be mapped using a set of ECC public keys, $\{P_i\}$, derived from a set of private keys, $\{s_i\}$, as shown:

25

$$P_i = s_i \cdot G$$

Exchanging private keys rather than public keys during a set shuffle, whose values are revealed at the end ensures:

- An individual can easily calculate the public key for that private key, revealing the object it is mapped to.

30

- Anyone who can provide a zero-knowledge proof (ECDSA signature for example) for a public key value, must have knowledge of the private key. This can guarantee a party has that element in their possession without the party having to reveal it.
 - Funds can be locked to known public keys and only unlocked by the relevant private
- 5 keys that are revealed and/or exchanged during the shuffle.

However, exchanging private keys using EC arithmetic is not possible as private keys are scalar values, not EC points. Further, private keys cannot be derived directly from public keys. Therefore a new method is described herein for element mapping, that allows private

10 keys to be represented as EC points for manipulation, but from which they can also be derived. These EC points are referred to as shuffle keys.

This set shuffling mechanism outlined in this disclosure requires an EC point that can be converted to a private key scalar. The EC point will be used for encryption and decryption

15 using scalar multiplication. Utilizing EC shuffle points allows for an efficient homomorphic and commutative encryption using scalar multiplication. Further, it allows a party to derive the private key directly from the EC point after say shuffling and decrypting, a feature unavailable with say the usual public key representation. This allows a generalised method to obscure and mix private keys within the set.

20

An initial EC point is used for any shuffling using EC scalar multiplication. Once shuffling is completed the EC point is decrypted to its original value, it is necessary that it can then be converted to a private key and public key. This relationship is shown in Figure 6.

- 25 One method that may be used for selecting an EC point and deriving a scalar from that point involves hashing the concatenated coordinates of the EC point to form the scalar as such:

$$s_i = H(x_i || y_i)$$

Whilst this method may be used, it is one directional as it is not possible to calculate the

30 shuffle key coordinates (x_i, y_i) from the private key s_i , due to the pre-image resistance of cryptographic hash functions.

Therefore a preferred method involves representing a scalar private key, s_i , as the x-coordinate of an EC point referred to as a shuffle key, P'_i . This provides a bi-directional relationship, where the EC point or shuffle key P'_i can be calculated from the private key s_i , and vice versa as highlighted in Figure 7.

5

The equation for an EC point within the secp256k1 protocol is:

$$y^2 = x^3 + 7$$

defined over the finite field, $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$.

- 10 The first stage of this method is to select a point from the curve, referred to as the shuffle point, P'_i , defined as:

$$P'_i = (s_i, \pm\sqrt{s_i^3 + 7})$$

- From this shuffle point coordinate, one can extract the x-coordinate as the private key, s_i , which will be used for element mapping. Note that when converting from the private key, s_i , back to the shuffle key, P'_i , there will be 2 y-coordinate solutions, for each of the positive and negative square root derivations. The protocol (or one of the parties) specifies which solution to take.

- 20 Typically, private key values, s_i , are chosen at random from finite field $\mathbb{F}_p$, e.g. within the range $1 - p$. EC points have order q , where $p \approx q$ with regards to secp256k1. However, selecting a random value from the finite field $\mathbb{F}_p$ and using that value as the x-coordinate for an EC point may not result in a valid point being selected. In fact, there is an approximately 50% that a random value from the finite field $\mathbb{F}_p$ will yield a valid EC point and then it will typically be two valid EC points, formed from the positive and negative square root solutions for the y-coordinate:

$$y = \pm\sqrt{s_i^3 + 7}$$

- Therefore, utilising private keys as the x-coordinates for shuffle keys, limits the search space by a factor of 2, reducing security of the private-public keys pairs. If selecting a private key value first, there is a 50% chance that this will not result in a valid EC point and shuffle key. If

30

this occurs, Alice 103a can simply iterate through private key values, by adding the integer 1 to the private key value and attempting to derive a valid EC point again. As there is an approximately 50% chance this technique will ensure shuffle key derivation efficiently. Alternatively, Alice 103a can select the EC point that represents the shuffle key first and
5 derive the private key from this as the x-coordinate. Every EC points x-coordinate will correspond to a valid private key from finite field $\mathbb{F}_p$. This will be the technique used throughout this section.

The 50% security reduction only occurs if an attacker knows which positive or negative square root solutions for the y-coordinate has been selected for the protocol. Keeping this
10 secret removes the security reduction, however, this must be shared with people involved in any interactions. The protocol may stipulate that any funds associated with shuffle keys will only be active for the time required for shuffle interaction, which will typically be in the order of hours or at most days. Based on this it is still computationally infeasible to break a
15 private-public key pair with a security reduction of factor 2.

To summarise, element mapping is carried out as follows:

1. Select shuffle key set $\{P'_i\}$;
2. Extract private key set $\{s_i\}$ as x-coordinates of shuffle key set;
- 20 3. Calculate public key set $\{P_i\}$ from private key set;
4. Share public key mapping as object set.

It may be necessary to establish rules and element mapping prior to any interaction. Once these have been agreed, both parties should acknowledge and confirm their agreement and
25 have this agreement represented on the blockchain. The interaction rules and the element mapping can be embedded within a blockchain transaction that is signed by all parties involved. This storage may be performed using the functionality of the opcodes OP_RETURN or OP_PUSHDATA // OP_DROP. In some instances, there may be too much interaction information to efficiently store all details directly as string within the transaction; in such
30 scenarios one could store, in the transaction, the hash of the interaction information instead, while the raw data is available off-chain. All players may be required to sign the transaction that contains the hash (or raw data) of this document, which can then

effectively be used to confirm their agreement to play by those interaction rules using the stated element mappings.

6.2 Shuffling and element encryption

5 Now that shuffle keys and element mapping have been detailed, this section will outline the set shuffling and element encryption protocols. Set shuffling involves the concealing of elements e.g., shuffle key P'_i , and changing the order to ensure no party knows which element is which. The method involves each party performing point multiplication on the set and reordering each element, as shown in Figure 8.

10

In this example each party, e.g. Alice 103a, Bob 103b and Claire 103c, will:

1. Individually generate a scalar value [EC private key range] that they will keep private and secure (the values a, b, and c, respectively),
2. Multiply the set of shuffle points by their secret scalar,
- 15 3. Rearrange the sequence in which the set of elements are listed.

As an example, Alice 103a would multiply each of the shuffle points in the set $\{P'_i\}$ by the scalar value a to produce the new set of points $\{aP'_i\}$ ("first encrypted shuffle keys"). The order of these points would then be altered. These shuffled and encrypted points are then
20 passed on to the next party (Bob 103b) who repeats the previous steps using $\{aP'_i\}$ to produce a further shuffled set $\{abP'_j\}$. This repeats for every party 103 and can be expanded to as many parties as necessary.

After each stage, each party 103 has the responsibility of ensuring that the element order of
25 their resultant shuffled set is recorded in a conformed blockchain transaction that they themselves have signed (for example in Bitcoin using OP_RETURN). This creates an event chain which can be used to ensure no party can dispute their stated resultant shuffled set. These state commitments provide an irrefutable record of the set at each stage. This can be used to debug where and error has occurred during the process, be that error unintentional
30 or by a malicious actor.

Once the shuffling is completed by all parties Alice, Bob and Claire, they will now all have access to the final shuffled set (“final encrypted shuffle keys”):

$$\left\{ \begin{array}{l} abcP'_{k1} \\ abcP'_{k2} \\ abcP'_{k3} \\ \dots \\ abcP'_{kn} \end{array} \right\}$$

5

The availability of this final shuffled set may be sufficient for some purposes. An example of such, is a game of highest card winner, in which each party 103 first selects an element from the shuffled-encrypted set. Each party 103 will then commit to that element’s selection by recording their selection within a blockchain transaction.

10

After all parties 103 have committed to their selected elements, each party reveals their inverse scalar multiplication keys. This allows Alice 103a, Bob 103b and Claire 103c to decrypt the final shuffled set and reveal the original set $\{P'_i\}$ and thus what their selection corresponds to. This also allows them to decrypt the entire set, negating the shuffling as each elements original identity is now revealed.

15

This could be disadvantageous if there were multiple phases of the interaction and the shuffled set needed to be reused. Further, some interactions may require that specific elements remain secret to the individual with that element, for example if funds were tied to the public address related to the private keys revealed during inverse multiplication. Therefore, to enable specific element reveals (P'_i values rather than the entire set $\{P'_i\}$), each of Alice 103a, Bob 103b and Claire 103c may perform the following steps sequentially on the previously shuffled set:

20

1. Multiply the shuffled set with the inverse of the scalar key used in the previous stage (the values a^{-1} , b^{-1} , and c^{-1} , for Alice, Bob and Claire’s turns respectively),
2. Multiply each element of the set with distinct keys (“second-round scalar values”) $((a_1, a_2, a_3, \dots, a_n), (b_1, b_2, b_3, \dots, b_n)$ and $(c_1, c_2, c_3, \dots, c_n)$, for Alice, Bob and Claire’s turns respectively).

25

This process is summarised in Figure 9.

Once distinct key multiplication is completed, Alice 103a, Bob 103b and Claire 103c have access to the set ("second-round final encrypted shuffle keys"):

$$\begin{matrix} 5 & \left\{ \begin{array}{l} a_1 b_1 c_1 P'_{k1} \\ a_2 b_2 c_2 P'_{k2} \\ a_3 b_3 c_3 P'_{k3} \\ \dots \\ a_n b_n c_n P'_{kn} \end{array} \right\} \end{matrix}$$

6.3 Interaction and element reveal

From this stage parties Alice 103a, Bob 103b and Claire 103c would begin the interaction they had initially outlined. For shuffled set interactions, this includes selecting some
 10 unknown elements at random and then revealing them. One of the benefits of the protocol outlined in this disclosure is selection commitment using the blockchain. Selecting an unknown element at random, and then committing that element to a signed blockchain transaction (e.g. using OP_RETURN), creates an irrefutable instance of that selection which
 15 transactions.

Once selections have been committed, parties request the specific inverse multiplication keys to reveal their selected element. For example, if Alice 103a chooses element $\{a_3 b_3 c_3 P'_{k3}\}$, she only requires distinct keys b_3 and c_3 from Bob 103b and Claire 103c
 20 respectively. These keys do not allow Alice 103a to reveal any information about other elements in the set.

Once Alice 103a has decrypted to reveal her chosen P'_k , she can take the x-coordinate for this point as the private key s_k and map the associated public key and work out what value
 25 she has. This is repeated for Bob 103b and Claire 103c. The parties 103 may continue with the interaction until all relevant inverse multiplications have been completed.

6.4 Outcome and Settlement

From here parties 103 can prove they have certain objects, e.g. using a zero-knowledge proof (ECDSA signature for example) for the public key value that represents the object. Based on the previously agreed rule set winners and losers can be determined either off-chain, or on-chain.

6.5 Worked example – Highest Card Game

Alice 103a and Bob 103b wish to play a game, involving a set $\mathbb{D}$ of 3 elements that represent cards: Ace, King and Queen. They establish that each player must randomly select 1 card and whoever has the highest card wins, where:

$$Ace > King > Queen$$

The initial stage would require Alice 103a define an indexed representation of the set, $\mathbb{D}$:

$$\mathbb{D} = \{Ace, King, Queen\} = \{Element_1, Element_2, Element_3\}$$

Alice 103a must now represent these indexed objects with private-public key-pairs. She first selects 3 shuffle key points, P'_i :

$$\{P'_i\} = \{P'_1, P'_2, P'_3\}$$

From these 3 shuffle key points, Alice 103a extracts the x-coordinate to form the set $\mathbb{S}$ of corresponding 3 private keys:

$$\mathbb{S} = \{s_1, s_2, s_3\}$$

These private keys are used to derive the set $\mathbb{P}$ of corresponding public keys

$$\mathbb{P} = \{P_1, P_2, P_3\}$$

Where:

$$P_i = s_i \cdot G$$

And G is the generator defined from secp256k1.

The public key object mapping is now submitted to Bob 103b and the blockchain, to immutably confirm that:

$$\{Ace, King, Queen\} = \{P_1, P_2, P_3\}$$

- 5 Figure 10 summarises the process. Once Bob 103b has confirmed he has received and is happy with the mapping, Alice 103a can proceed to the next stage: shuffling.

At the end of Bob's shuffle, the set can be considered as sufficiently shuffled and encrypted, such that no party could determine the original value of one of the elements. This is a
 10 single-phase interaction, with off-chain settlements, so there is no need for Alice 103a and Bob 103b to go through the process of distinct key multiplication for P'_1 , P'_2 and P'_3 . It is important to note that state commitments performed at each stage of shuffling provide an irrefutable tool for debugging if an error occurs during the reveal phase, or if a party attempts to cheat. State commitments are shown in Figure 11.

15

From the previous stage Alice 103a and Bob 103b have a shuffled and obscured set that is common to both:

$$\begin{aligned} &\{abP'_2\} \\ &\{abP'_3\} \\ &\{abP'_1\} \end{aligned}$$

- 20 Alice 103a selects the second element, $\{abP'_3\}$, from the set and submits this selection to a blockchain transaction (using OP_RETURN or OP_PUSHDATA // OP_DROP), creating a commitment to this selection prior to revealing the shuffle key value P'_3 .

- Bob 103b, seeing Alice's selection commitment on the blockchain, now selects another
 25 element, $\{abP'_2\}$, from the remaining set and likewise submits this selection to a blockchain transaction, creating a commitment to this selection prior to revealing its value. Now that both Alice 103a and Bob 103b have committed to an element from the set, they exchange the relevant inverse multiplication keys, a^{-1} and b^{-1} .

Using the exchanged key, and their own inverse multiplication key for the element, Alice 103a and Bob 103b can decrypt the element and determine they have shuffle key values P'_3 and P'_2 respectively, where:

$$\begin{aligned} P'_3 &= (s_3, +\sqrt{s_3^3 + 7}) \\ P'_2 &= (s_2, +\sqrt{s_2^3 + 7}) \end{aligned}$$

Both parties can extract the positive x-coordinate from these points and relate them to the publicly available mapping system for public keys to card values.

10

Now that the game stage has been completed, e.g. Alice 103a and Bob 103b have both selected and decrypted their cards, they move to determining the outcome and settlement of the game. Alice can determine that her private key corresponds to the Queen card, as such:

$$s_3 \cdot G = P_3 \rightarrow 'Queen'$$

Bob can also determine that his private key corresponds to the King card:

$$s_2 \cdot G = P_2 \rightarrow 'King'$$

20 Based on the previously agreed rules, that highest card wins, Bob has won the game.

The described protocol provides a novel solution for set shuffling using the blockchain, which provides the following benefits and contributions:

- Shuffle keys – Elliptical curve arithmetic provides an efficient solution for homomorphic and commutative point manipulation. Relating private keys to shuffle keys provides a novel solution to representing private keys as the shuffled elements, allowing EC arithmetic to be performed, whilst also be able to derive the private key scalar value during the reveal phase. This allows:
 - An individual to prove they are in possession of an element, represented as a private key, without revealing the element using a ZKP.

- Funds to be locked to known public keys and only unlocked by the relevant private keys that are revealed and/or exchanged during the shuffle.
- State commitments – Set shuffling typically involves contributions from multiple and potentially untrusted parties. In this disclosure, each party must provide a scalar multiplication to the set and re-order the set. Committing the set state to the blockchain as a transaction after each phase e.g., after each person has shuffled, provides a mechanism for debugging if there is an error in the final reveal.
 - For example, consider Alice, Bob and Claire shuffle a card deck and during the final reveal, the private-public key pairs do not correspond to any of the originally mapped elements. They can then go through each phase of shuffling using the blockchain enabled state attestation, revealing their scalar multipliers and calculate where the error was made. This can be used to detect malicious actors.
- Selection commitments – Many set shuffling interactions involve the selection of an encrypted element, prior to revealing its decrypted value. Committing to an element selection using a blockchain transaction provides an irrefutable commitment by a party to that selection. Further, applying digital signatures to transactions that contain data related to some rules, creates a guarantee that a party with respective private key signed that data at that instance. This can be used for off-chain dispute settlement.
- Private communication – The private-public pairs generated as part of the blockchain ecosystem can be used for private communication and data exchange, e.g. using Diffie-Helman principles.

7. EXAMPLE USE CASE - VOTE TOKENS WITH PSEUDO-ANONYMITY

Consider the board of a company which wants to vote on an upcoming decision, or series of decisions. The board wants to ensure pseudo-anonymity for its members voting, but also guarantee that each board member can only cast one vote. It is important that the individual setting up the vote (i.e. the first party 103a), in this case the board executive, can provably guarantee that only a fixed number of votes can be cast. The executive of the company can utilise the set shuffling mechanism outlined in this disclosure to achieve these goals.

Voting public key addresses will be defined and randomly assigned to the board members. Once each board member has private-public key pair corresponding to a voting address, the executive will generate vote tokens and transfer them to the voting addresses. These vote
 5 tokens will then be used to cast votes by board members anonymously, as no one can know who controls which address.

In this use case, a vote token is defined as a UTXO representing a nominal value (1 Satoshi, for example) plus any fees necessary for transacting it. The executive will assign these vote
 10 tokens to voting public key addresses, in a large transaction, called an *issuance transaction*. These vote tokens will then be used to cast a vote by spending them in a second transaction from the assigned voting public key addresses, called a *vote transaction*.

The process can be broken down as follows:

- 15 1. Voting address setup:
 - a. Vote address mapping,
 - b. Set shuffling,
 - c. Element selection and decryption
2. Vote token setup:
 - 20 a. Secret commitment,
 - b. Vote token issuance,
3. Voting.

Voting address setup has been covered above. Vote token setup and voting represent novel
 25 contributions of this use case, as will be outlined.

The executive 103a of the company can begin the process by mapping the voting addresses. She generates 100 shuffle keys, one for each of the 100 board members:

$$\{P'_i\} = \left\{ \begin{array}{c} P'_1 \\ P'_2 \\ \vdots \\ P'_{100} \end{array} \right\}$$

From these 100 unique shuffle keys, the executive extracts the x-coordinates to form a set of corresponding 100 private keys, $\{s_i\}$:

$$\{s_i\} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_{100} \end{pmatrix}$$

- 5 And corresponding set of public keys, $\{P_i\}$:

$$\{P_i\} = \begin{pmatrix} P_1 \\ P_2 \\ \vdots \\ P_{100} \end{pmatrix}$$

- The executive can then commit these 100 public keys to a blockchain transaction, recording them with the opcodes OP_RETURN or OP_PUSHDATA // OP_DROP, or some hashed version
 10 of these values. The executive may require that all 100 board members sign this transaction to provably consent they are happy with the arrangement, or simply use it as a method of attestation for later use. The process is illustrated in Figure 12.

- 15 At this stage the executive can provably state that only these 100 public keys, committed within the transaction, can be used to cast a vote once assigned a vote token. Now the executive can begin the process of assigning the respective keys to each board member using set shuffling.

- 20 Firstly, she encrypts the shuffle key set by point multiplication of each element, by some secret scalar a , that is known only to her, as such:

$$\begin{pmatrix} aP'_1 \\ aP'_2 \\ \vdots \\ aP'_{100} \end{pmatrix}$$

- She then changes the sequence of the elements within the set to a new sequence, that only
 25 she knows:

$$\begin{pmatrix} a^{P'}_{i1} \\ a^{P'}_{i2} \\ \vdots \\ a^{P'}_{i100} \end{pmatrix}$$

The executive can now pass the mixed and encrypted set to the first board member (e.g. Bob 103b), who performs the same routine. Firstly, encrypting the set through point
 5 multiplication of some scalar b , that is known only to that board member, and then proceeding to mix the set sequence, forming a new set:

$$\begin{pmatrix} ab^{P'}_{j1} \\ ab^{P'}_{j2} \\ \vdots \\ ab^{P'}_{j100} \end{pmatrix}$$

Each board member (e.g. Claire 103c) then repeats this process in turn on the set:

- 10 1. Point multiplication with their secret scalar e.g., b for board member 1, c for board member 2, etc.
 2. Mixes the set sequence.

Once the 100th board member has encrypted the set with their secret scalar σ and shuffled
 15 the set sequence, the board have a completely shuffled set that has been encrypted with 101 scalar values and mixed 101 times, as such:

$$\begin{pmatrix} abc \dots \sigma^{P'}_{z1} \\ abc \dots \sigma^{P'}_{z2} \\ \vdots \\ abc \dots \sigma^{P'}_{z100} \end{pmatrix}$$

The entire element shuffling process is summarised in Figure 13:

20

It may be necessary for each of the board members to publish their reordered and encrypted sets, as a means of state commitment to debug each phase if the final revealed keys don't correspond to the original keys. This could be the result of an unintentional miscalculation or a malicious act. However, for the purposes of this example, it is assumed
 25 that the board members are all competent and cooperative, as the only outcome to be

gained from maliciously substituting values would be that the vote couldn't be carried out with this shuffled set.

- After the set has been shuffled and encrypted by each of the board members, the order of the elements is now fixed. The executive takes the set from the 100th board member and removes her scalar multiplicative a from the set by performing point multiplication with the inverse, a^{-1} , forming the following set:

$$\left\{ \begin{array}{c} bc \dots \sigma P'_{z1} \\ bc \dots \sigma P'_{z2} \\ \vdots \\ bc \dots \sigma P'_{z100} \end{array} \right\}$$

- 10 She now multiplies each of the elements by a distinct scalar, $\{a_1, a_2, \dots, a_{100}\}$, as such:

$$\left\{ \begin{array}{c} a_1 bc \dots \sigma P'_{z1} \\ a_2 bc \dots \sigma P'_{z2} \\ \vdots \\ a_{100} bc \dots \sigma P'_{z100} \end{array} \right\}$$

Each of the 100 board members then, in turn:

1. Performs point multiplication with their specific inverse e.g., b^{-1} for board member 1, c^{-1} for board member 2, etc.
2. Performs point multiplication on each of the set elements with a distinct scalar e.g., each of the elements by a distinct scalar e.g., $\{b_1, b_2, \dots, b_{100}\}$ for board member 1, $\{c_1, c_2, \dots, c_{100}\}$ for board member 2, etc.

- 20 Once the 100th board member has removed their multiplicative scalar using σ^{-1} and multiplied by the 100 distinct encryption keys, $\{\sigma_1, \sigma_2, \dots, \sigma_{100}\}$, the final distinct key encrypted set is formed:

$$\left\{ \begin{array}{c} a_1 b_1 c_1 \dots \sigma_1 P'_{z1} \\ a_2 b_2 c_2 \dots \sigma_2 P'_{z2} \\ \vdots \\ a_{100} b_{100} c_{100} \dots \sigma_{100} P'_{z100} \end{array} \right\}$$

- 25 The entire distinct key encryption process is summarised in Figure 14.

This set may be sent to all board members and may also be committed to the blockchain for transparency.

Now each board member must select/receive an element from the set. As the set has been thoroughly mixed and no one can derive which element is which, it is more efficient to simply assign the elements in the order in which they were exchanged during shuffling, as such:

$$\begin{aligned} \{a_1 b_1 c_1 \dots \sigma_1 P'_{z1}\} &\rightarrow \text{Board member 1} \\ \{a_2 b_2 c_2 \dots \sigma_2 P'_{z2}\} &\rightarrow \text{Board member 2} \\ &\vdots \\ \{a_{100} b_{100} c_{100} \dots \sigma_{100} P'_{z100}\} &\rightarrow \text{Board member 100} \end{aligned}$$

Note, that each of the board members can know the identity of the other board members, as would be expected. However, they can never know any information that maps a voting public key to a voter's identity, that is, without all 99 board members colluding to identify the voting address of the 100th board member. This allows easy communication and data transmission across the various phases.

15

Now that each board member has some encrypted element, they request the distinct inverse multiplication keys from the other board members, as such:

- Board member 1 requests: a_1^{-1} from the executive and c_1^{-1}, d_1^{-1} , etc. from the other board members.
- 20 • Board member 2 requests: a_2^{-1} from the executive and b_2^{-1}, d_2^{-1} , etc. from the other board members.
- Board member 3 requests: a_3^{-1} from the executive and b_3^{-1}, c_3^{-1} , etc. from the other board members.
- This is continued for all 100 board members.

25

After each board member receives the inverse multiplication keys, they can reveal their respective shuffle key, P'_i .

Extracting the x-coordinate from this shuffle key reveals their respective private key, s_i .

Now each board member can determine which public address, P_i , they have with:

$$P_i = s_i \cdot G$$

- 5 At this stage each board member should have a private-public key pair that corresponds to one of the 100 voting public keys outlined by executive in the first committed transaction.

Now that board member has private-public key pair, representing one of the voting addresses, the group can move to the vote token setup.

10

In some examples, board members may commit to some hashed secret, S_i , prior to voting. This secret will later be hashed with a salt value as means to provably link a vote to a board member.

- 15 They will confirm these secrets by committing the hashed versions of them to a blockchain transaction, e.g. by recording them with the opcodes OP_RETURN or OP_PUSHDATA // OP_DROP. The hashed versions of these secrets, $H(S_i)$, should be visible for other board members as a means of identifying votes later, should the need arise.

- 20 The secret commitment transactions will look as such:

25

Secret Transaction, TxD_{Si}			
Inputs		Outputs	
Value	Script	Value	Script
1 Satoshi + fee	$\langle Sig(P_i, Tx_{IT}) \rangle$ $\langle P_i \rangle$	1 Satoshi	OP_RETURN $\langle "H(S_i)" \rangle$

The preimage secrets, S_i , remains secret and will later be used with a salt to form a new hashed message when voting. This may be used to prevent the executive from signing on behalf of the board members, as will be discussed.

30

Once all board members have submitted their secret transactions, the executive generates and submits an *issuance transaction*. This transaction transfers 1 voting token to each of the 100 voting addresses.

- 5
- This transaction has a funding input, signed by the executive creating the transaction, and 100 P2PK outputs for each of the 100 voting public keys. As part of the voting phase, the executive would outline some rules for voting, such as ID for this specific vote, matter being voted on, deadline for voting, etc, for this specific vote *issuance transaction* using OP_RETURN, and share the reference to all board members. This would create an
- 10
- immutable copy of them for future reference.

Below is an outline of such *issuance transaction*:

15

Issuance Transaction, $TxID_{IT}$			
Inputs		Outputs	
Value	Script	Value	Script
$Vote\ token_{1-100+}$	$\langle Sig(P_E, Tx_E) \rangle$	$Vote\ token_1$	$\langle P_1 \rangle$
fees	$\langle P_E \rangle$		OP_CHECKSIG
		$Vote\ token_2$	$\langle P_2 \rangle$
			OP_CHECKSIG
		...	...
		$Vote\ token_{100}$	$\langle P_{100} \rangle$
			OP_CHECKSIG
		1 Satoshi + fee	OP_RETURN
			$\langle 'Vote$
			Confirmation
			ID'
			'Vote rules,
			time limit etc.>

20

25

Note that each board members (or voters) can also view the blockchain to verify that the *issuance transaction* has been generated correctly for each of the previously committed voting public key addresses.

5 Now the board members can conduct a vote, by transferring their vote token. This may take the form of vote tokens submitted to one address, representing a 'yes' vote, and vote tokens sent to another address, representing a 'no' vote on a particular matter. Where these addresses are predefined and agreed by the executive and board members. Here, voting is defined as the transfer of vote tokens from voting addresses back to the
10 executive's public address, with the voting recorded within the transaction using OP_RETURN. This provides greater flexibility for voting options. Within this OP_RETURN output voters may input one or more of the following:

- Vote code reference – Reference to the matter being voted on, this unique identifier could be used so that an automated vote registering system can search for the voting
15 number efficiently.
- Voting decisions – The choice of the voter, such as yes, no, Abstain, etc.
- Hashed secret with salt – Here the board member will input a hash of their secret, S_i , from their original commitment, concatenated with some salt that the board member also selects.

20

This secret with salt, $H(S_i, salt_i)$, ensures that a board member can link their vote to themselves, as only they can connect $H(S_i, salt_i)$ and $H(S_i)$, assuming pre-image resistance of SHA-256 for example. Note that no one can link this vote to one of the originally committed hashed secrets and thus voter identities. However, it provides a mechanism for a
25 board member to prove a vote was, or wasn't, committed by them should the need arise. This could be required if the executive attempted to vote on a board members behalf, as she knows all the associated private keys for voting public keys. The process is highlighted below:

30 If a dispute arose over who signed for a voting public key, this pre-image secret and salt may be revealed and provably linked to both the original identity linked secret commitment and

the vote. Which would strongly disincentivise the executive from falsely signing for a vote token.

The vote transactions may look as such for each vote:

5	Vote Transaction, $TxID_{Vi}$			
	Inputs		Outputs	
	Value	Script	Value	Script
	$Vote\ token_i +$ fee	$\langle Sig(P_i, Tx_{IT}) \rangle$ $\langle P_i \rangle$	$Vote\ token_i$ + fee	$\langle P_v \rangle$ OP_CHECKSIG
10			fee	OP_RETURN < "Board Code Reference" "Voting Decision" $H(S_i, salt_i)$ >
15				

Once all the board members had voted, or after the predetermined amount of time for voting had elapsed, the executive and board members can utilise the blockchain to determine the vote count, which is the number of vote tokens cast for any decision. This

creates an immutable and irrefutable method for 100 participants to vote.

This setup would only be required once, and board members could continue to use these private-public key pairs to vote across multiple matters with pseudo-anonymity. The executive, or some other board member, could now repeatedly distribute vote tokens using *issuance transactions* to the voting addresses to conduct multiple votes.

25

Each of the board members and executive can confirm, if and how votes have been cast by searching for the UTXOs that represent the vote tokens, generated in the *issuance transaction*.

30 This is a benefit of voting by signing for individual outputs of an *issuance transaction*.

Mechanisms have been proposed which conduct voting by collecting signatures for inputs to a transaction. However, this requires a coordinator to do so, who can use the signature

collection phase to potentially deduce who is signing for what. Complex communication protocols can be used to pass signatures anonymously, but this is deemed less effective than the system detailed here.

- 5 As board members will likely know the identities of one another, a board member can indicate they haven't signed for a vote token off-chain, this would mean one of two things:
1. The private keys have not been assigned correctly and the process should be restarted. This would be the case if some vote token had not been signed for.
 - 10 2. The executive has intentionally misallocated private keys to manipulate the voting. This would be the case if all vote tokens had been signed for, but some board members had reported they had not signed.

15 Either of these may allow a denial-of-service to at least delay, if not prevent a vote. Using state commitments at each stage of the set shuffle can allow board members and/or the executive to determine and what stage any issues occurred. These can be associated with the board member who committed this state, and appropriate actions to remove this board member taken if necessary.

20 If no issues occur, it can be assumed each board member has one of the unique addresses. Now the board can perform multiple votes with pseudo-anonymity. Note that neither the executive, nor any of the board members, can determine who has which private-public key pair. Conducting voting through the signing of individual transaction outputs on-chain preserves this anonymity. For example, voting on-chain through the input of some transaction, such as a large multi-signature for all board members, would require one party to receive and collate all the votes. This presents an unnecessary additional security risk, where this collating party could derive who has voted for what or manipulate the votes. However, the executive does know each of the private key values for each of the voting public key addresses. This could allow them to hijack addresses and vote against the will of the board member for which the addresses corresponded. The secret commitment protocol
30 provides a mechanism to provably link a vote to a board member, doing so may reveal their

identity though. However, this should provide sufficient reason for the executive to act fairly.

8. FURTHER REMARKS

5

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 to the bitcoin blockchain. More generally, any reference above to bitcoin network 106,
15 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 104 as described above.

20

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
25 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).

In other embodiments of the invention, the blockchain network 106 may not be the bitcoin
30 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.

Even more generally, any reference to the term “bitcoin node” 104 above may be replaced
5 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.

Statement 1. A computer-implemented method of shuffling an ordered set of data items
15 for enabling pseudorandom selection of one or more data items between a group of two or more parties, wherein the method is performed by a first party of the group and comprises:
generating an ordered set of shuffle keys, wherein each shuffle key is a respective elliptic curve point, and wherein said generating comprises, for each respective data item in the ordered set:

20 either deriving the respective shuffle key based on a respective private key,
or generating the respective shuffle key and deriving the respective private key based on the respective shuffle key,

deriving, based on the respective private key, a respective public key, and
mapping the respective public key to the respective data item;
25 encrypting each respective shuffle key with a first scalar value to obtain a respective first encrypted shuffle key;

placing each respective first encrypted shuffle keys in an ordered set, wherein the
order of the respective first encrypted shuffle keys is shuffled compared with the order of
the respective shuffle keys in the ordered set of respective shuffle keys; and

30 sharing the ordered set of respective first encrypted shuffle keys with at least a
second party of the group.

The second party is configured to encrypt each respective first encrypted shuffle key with a second scalar value to obtain a respective second encrypted shuffle key, and place each respective shuffle key in an ordered set, wherein the order of the respective encrypted shuffle keys is shuffled compared with the order of the respective first encrypted shuffle keys in the ordered set of respective first encrypted shuffle keys. The second party shares the ordered set of respective first encrypted shuffle keys with the first party or a third party of the group.

Statement 2. The method of statement 1, wherein said deriving of the respective private key comprises selecting a respective x-coordinate of the respective shuffle key as the respective private key.

Statement 3. The method of statement 1, wherein said deriving of the respective private key comprises deriving the respective private key based on a hash of the respective shuffle key.

Statement 4. The method of any preceding statement, comprising one, some or all of:

- a) storing a commitment of the ordered set of shuffle keys in a first blockchain transaction, and sending the first blockchain transaction to one or more nodes of a blockchain network;
- b) storing a commitment of each first encrypted shuffle key in a first blockchain transaction, and sending the second blockchain transaction to one or more nodes of the blockchain network;
- c) storing a commitment of the ordered set of first encrypted shuffle keys in a third blockchain transaction, and sending the third blockchain transaction to one or more nodes of the blockchain network.

Statement 5. The method of statement 5, wherein the first, second, and third blockchain transactions are different transactions. Alternatively, one or more of the first, second and third blockchain transactions may be the same transactions.

Statement 6. The method of any preceding statement, comprising:

sharing the mapping of each respective public key to the respective data item with the group of parties.

Statement 7. The method of any preceding statement, comprising:

- 5 storing a commitment of the mapping of each respective public key to the respective data item a fourth blockchain transaction, and sending the fourth blockchain transaction to one or more nodes of the blockchain network.

Statement 8. The method of any preceding statement, comprising:

- 10 obtaining an ordered set of final encrypted shuffle keys, wherein each respective final encrypted shuffle key is generated by each party encrypting the respective first encrypted shuffle key with a respective scalar value.

- Put another way, the ordered set of respective final encrypted shuffle keys is generated by
15 each next party in the group, starting with the second party, performing an iterative process of: i) obtaining a respective ordered set of respective encrypted shuffle keys from a respective previous party, encrypting each respective encrypted shuffle key with a respective scalar value to obtain a respective next encrypted shuffle key, placing each respective next encrypted shuffle key in a respective next ordered set, where the order of
20 the respective next encrypted shuffle keys is shuffled compared to the order of respective shuffle keys in the respective obtained ordered set, and sharing the respective next ordered set with at least the next party.

Statement 9. The method of statement 8,

- 25 committing to a selection of one of the final encrypted shuffle keys;
 revealing a first inverse value of the first scalar value to the other parties, wherein each other party reveals a respective inverse value of the respective scalar value used by that other party;
 decrypting at least the selected final encrypted shuffle key using each inverse value
30 to obtain a selected shuffle key;
 deriving a selected private key from the selected shuffle key;
 deriving a selected public key from the selected private key; and

determining a selected data item based on the mapping of the respective public keys to the respective data items.

Statement 10. The method of statement 9, wherein said committing to the selection of one
5 of the final encrypted shuffle keys comprises:

storing a commitment of the selected final encrypted shuffle key in a fifth blockchain transaction, and sending the fifth blockchain transaction to one or more nodes of the blockchain network.

10 Statement 11. The method of any preceding statement, comprising:

decrypting each respective final encrypted shuffle key with the first scalar value to obtain a respective target encrypted shuffle key, wherein the respective scalar values used to obtain the respective final encrypted shuffle keys are first-round scalar values;

15 encrypting each respective target encrypted shuffle key with a different second-round first scalar value to obtain an ordered set of respective second-round first encrypted shuffle keys, wherein the order of the ordered set of respective second-round first encrypted shuffle keys is the same as the order of the final ordered set of final encrypted shuffle keys;
sharing the ordered set of respective second-round first encrypted shuffle keys with at least the second party.

20

Statement 12. The method of statement 11, comprising:

obtaining an ordered set of respective second-round final encrypted shuffle keys, wherein the ordered set of respective second-round final encrypted shuffle keys is generated by each next party in the group, starting with the second party, performing a
25 process of: i) obtaining a respective ordered set of respective second-round encrypted shuffle keys from a respective previous party, decrypting each respective second-round encrypted shuffle key with the respective first-round scalar value to obtain a respective second-round next encrypted shuffle key, and sharing the respective next ordered set with at least the next party;

30 committing to a selection of one of the second-round final encrypted shuffle keys;

obtaining, from each other party, a respective second-round inverse value of the respective second-round scalar value used by that other party to encrypt the selected second-round final encrypted shuffle keys;

- 5 decrypting the selected second-round final encrypted shuffle key using each
 5 respective second-round inverse value to obtain a selected shuffle key;
 deriving a selected private key from the selected shuffle key;
 deriving a selected public key from the selected private key; and
 determining a selected data item based on the mapping of the respective public keys
to the respective data items.

10

Statement 13. The method of statement 12, wherein said committing to the selection of one of the second-round final encrypted shuffle keys comprises:

- storing a commitment of the selected second-round final encrypted shuffle key in a
sixth blockchain transaction, and sending the sixth blockchain transaction to one or more
15 nodes of the blockchain network.

Statement 14. The method of any preceding statement, wherein each respective public key is used to lock a respective unspent transaction output.

- 20 Statement 15. The method of statement 14, comprising:

 locking the respective unspent transaction output to the respective public key.

Statement 16. The method of statement 14 or statement 15 when dependent on any of statements 8 to 13, comprising:

- 25 using the selected private key to unlock the respective unspent transaction output
locked to the selected public key.

Statement 17. The method of statement 16, wherein a sixth blockchain transaction is used to unlock the respective unspent transaction output locked to the selected public key, and

- 30 wherein the method comprises:

storing a commitment of a secret value in a seventh blockchain transaction, and sending the seventh blockchain transaction to one or more nodes of the blockchain network;

storing a commitment of a combination of the secret value and a salt value in the sixth blockchain transaction.

The method may comprise revealing the salt value and the secret value to prove that the sixth blockchain transaction was generated by the first party.

10 Statement 18. The method of any preceding statement, wherein each respective data item is a respective token.

Statement 19. The method of statement 18, wherein each respective token represents a respective vote.

15

Statement 20. A computer-implemented method for enabling pseudorandom selection of one or more data items between a group of two or more parties, wherein the method is performed by a party of the group and comprises:

receiving an ordered set of encrypted shuffle keys, wherein the ordered set of encrypted shuffle keys is generated by:

generating an ordered set of shuffle keys, wherein each shuffle key is a respective elliptic curve point, and wherein said generating comprises, for each respective data item in the ordered set, deriving the respective shuffle key based on a respective private key, or generating the respective shuffle key and deriving the respective private key based on the respective shuffle key,

encrypting each respective shuffle key with a first scalar value to obtain a respective first encrypted shuffle key;

placing each respective first encrypted shuffle key in an ordered set, wherein the order of the respective first encrypted shuffle keys is shuffled compared with the order of the respective shuffle keys in the ordered set of shuffle keys; and

optionally, performing one or more further rounds of a next party receiving the ordered set of first encrypted shuffle keys, encrypting each first encrypted shuffle keys

with a respective scalar value to obtain a respective next encrypted shuffle key, placing each respective next encrypted shuffle key in a next ordered set, wherein the order of the respective next encrypted shuffle keys is shuffled compared with the order of the respective shuffle keys in the received ordered set of shuffle keys, and

5 encrypting each first or next encrypted shuffle key with a respective scalar value to obtain a respective next encrypted shuffle key;

 placing each respective next encrypted shuffle key in a next ordered set, wherein the order of the respective next encrypted shuffle keys is shuffled compared with the order of the respective shuffle keys in the received ordered set of shuffle keys; and

10 sharing the next ordered set of next encrypted shuffle keys with one or more parties of the group.

Statement 21. Computer equipment comprising:

 memory comprising one or more memory units; and

15 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 20.

Statement 22. A computer program embodied on computer-readable storage and

20 configured so as, when run on one or more processors, to perform the method of any of statements 1 to 20.

According to another aspect disclosed herein, there may be provided a method comprising the actions of each party of the group.

25

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

CLAIMS

1. A computer-implemented method of shuffling an ordered set of data items for enabling pseudorandom selection of one or more data items between a group of two or more parties, wherein the method is performed by a first party of the group and comprises:
 - generating an ordered set of shuffle keys, wherein each shuffle key is a respective elliptic curve point, and wherein said generating comprises, for each respective data item in the ordered set:
 - either deriving the respective shuffle key based on a respective private key, or generating the respective shuffle key and deriving the respective private key based on the respective shuffle key,
 - deriving, based on the respective private key, a respective public key, and mapping the respective public key to the respective data item;
 - encrypting each respective shuffle key with a first scalar value to obtain a respective first encrypted shuffle key;
 - placing each respective first encrypted shuffle keys in an ordered set, wherein the order of the respective first encrypted shuffle keys is shuffled compared with the order of the respective shuffle keys in the ordered set of respective shuffle keys; and
 - sharing the ordered set of respective first encrypted shuffle keys with at least a second party of the group.
2. The method of claim 1, wherein said deriving of the respective private key comprises selecting a respective x-coordinate of the respective shuffle key as the respective private key.
3. The method of claim 1, wherein said deriving of the respective private key comprises deriving the respective private key based on a hash of the respective shuffle key.
4. The method of any preceding claim, comprising one, some or all of:
 - a) storing a commitment of the ordered set of shuffle keys in a first blockchain transaction, and sending the first blockchain transaction to one or more nodes of a blockchain network;

b) storing a commitment of each first encrypted shuffle key in a first blockchain transaction, and sending the second blockchain transaction to one or more nodes of the blockchain network;

c) storing a commitment of the ordered set of first encrypted shuffle keys in a third blockchain transaction, and sending the third blockchain transaction to one or more nodes of the blockchain network.

5. The method of claim 4, wherein the first, second, and third blockchain transactions are different transactions.

6. The method of any preceding claim, comprising:
sharing the mapping of each respective public key to the respective data item with the group of parties.

7. The method of any preceding claim, comprising:
storing a commitment of the mapping of each respective public key to the respective data item a fourth blockchain transaction, and sending the fourth blockchain transaction to one or more nodes of the blockchain network.

8. The method of any preceding claim, comprising:
obtaining an ordered set of final encrypted shuffle keys, wherein each respective final encrypted shuffle key is generated by each party encrypting the respective first encrypted shuffle key with a respective scalar value.

9. The method of claim 8,
committing to a selection of one of the final encrypted shuffle keys;
revealing a first inverse value of the first scalar value to the other parties, wherein each other party reveals a respective inverse value of the respective scalar value used by that other party;
decrypting at least the selected final encrypted shuffle key using each inverse value to obtain a selected shuffle key;
deriving a selected private key from the selected shuffle key;

deriving a selected public key from the selected private key; and
determining a selected data item based on the mapping of the respective public keys to the respective data items.

10. The method of claim 9, wherein said committing to the selection of one of the final encrypted shuffle keys comprises:

storing a commitment of the selected final encrypted shuffle key in a fifth blockchain transaction, and sending the fifth blockchain transaction to one or more nodes of the blockchain network.

11. The method of any of claims 8 to 10, comprising:

decrypting each respective final encrypted shuffle key with the first scalar value to obtain a respective target encrypted shuffle key, wherein the respective scalar values used to obtain the respective final encrypted shuffle keys are first-round scalar values;

encrypting each respective target encrypted shuffle key with a different second-round first scalar value to obtain an ordered set of respective second-round first encrypted shuffle keys, wherein the order of the ordered set of respective second-round first encrypted shuffle keys is the same as the order of the final ordered set of final encrypted shuffle keys;

sharing the ordered set of respective second-round first encrypted shuffle keys with at least the second party.

12. The method of claim 11, comprising:

obtaining an ordered set of respective second-round final encrypted shuffle keys, wherein the ordered set of respective second-round final encrypted shuffle keys is generated by each next party in the group, starting with the second party, performing a process of: i) obtaining a respective ordered set of respective second-round encrypted shuffle keys from a respective previous party, decrypting each respective second-round encrypted shuffle key with the respective first-round scalar value to obtain a respective second-round next encrypted shuffle key, and sharing the respective next ordered set with at least the next party;

committing to a selection of one of the second-round final encrypted shuffle keys;

obtaining, from each other party, a respective second-round inverse value of the respective second-round scalar value used by that other party to encrypt the selected second-round final encrypted shuffle keys;

decrypting the selected second-round final encrypted shuffle key using each respective second-round inverse value to obtain a selected shuffle key;

deriving a selected private key from the selected shuffle key;

deriving a selected public key from the selected private key; and

determining a selected data item based on the mapping of the respective public keys to the respective data items.

13. The method of claim 12, wherein said committing to the selection of one of the second-round final encrypted shuffle keys comprises:

storing a commitment of the selected second-round final encrypted shuffle key in a sixth blockchain transaction, and sending the sixth blockchain transaction to one or more nodes of the blockchain network.

14. The method of any preceding claim, wherein each respective public key is used to lock a respective unspent transaction output.

15. The method of claim 14, comprising:

locking the respective unspent transaction output to the respective public key.

16. The method of claim 14 or claim 15 when dependent on any of claims 8 to 13, comprising:

using the selected private key to unlock the respective unspent transaction output locked to the selected public key.

17. The method of claim 16, wherein a sixth blockchain transaction is used to unlock the respective unspent transaction output locked to the selected public key, and wherein the method comprises:

storing a commitment of a secret value in a seventh blockchain transaction, and sending the seventh blockchain transaction to one or more nodes of the blockchain network;

storing a commitment of a combination of the secret value and a salt value in the sixth blockchain transaction.

18. The method of any preceding claim, wherein each respective data item is a respective token.

19. The method of claim 18, wherein each respective token represents a respective vote.

20. A computer-implemented method for enabling pseudorandom selection of one or more data items between a group of two or more parties, wherein the method is performed by a party of the group and comprises:

receiving an ordered set of encrypted shuffle keys, wherein the ordered set of encrypted shuffle keys is generated by:

generating an ordered set of shuffle keys, wherein each shuffle key is a respective elliptic curve point, and wherein said generating comprises, for each respective data item in the ordered set, deriving the respective shuffle key based on a respective private key, or generating the respective shuffle key and deriving the respective private key based on the respective shuffle key,

encrypting each respective shuffle key with a first scalar value to obtain a respective first encrypted shuffle key;

placing each respective first encrypted shuffle key in an ordered set, wherein the order of the respective first encrypted shuffle keys is shuffled compared with the order of the respective shuffle keys in the ordered set of shuffle keys; and

optionally, performing one or more further rounds of a next party receiving the ordered set of first encrypted shuffle keys, encrypting each first encrypted shuffle keys with a respective scalar value to obtain a respective next encrypted shuffle key, placing each respective next encrypted shuffle key in a next ordered set, wherein the order of the respective next encrypted shuffle keys is shuffled compared with the order of the respective shuffle keys in the received ordered set of shuffle keys, and

wherein the method further comprises:

encrypting each first or next encrypted shuffle key with a respective scalar value to obtain a respective next encrypted shuffle key;

placing each respective next encrypted shuffle key in a next ordered set, wherein the order of the respective next encrypted shuffle keys is shuffled compared with the order of the respective shuffle keys in the received ordered set of shuffle keys; and

sharing the next ordered set of next encrypted shuffle keys with one or more parties of the group.

21. 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 20.

22. 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 20.