

SET SHUFFLING**TECHNICAL FIELD**

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

5 shuffling a set of data items.

BACKGROUND

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)

10 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 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.

15 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 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
20 the blockchain may be pruned at some nodes, and the publication of blocks can be achieved through the publication of mere block headers.

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
25 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 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
30 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
5 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
10 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
15 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
20 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.

25 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
30 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

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
5 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
10 one or more data items between a group of two or more parties, each party having a respective index, and wherein the method is performed by a first party of the group and comprises generating an ordered set of first party shuffle keys, wherein each first party 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 first party
15 shuffle key based on a respective first private key, or generating the respective first party shuffle key and deriving the respective first private key based on the respective first party shuffle key. The method further comprises, for each respective data item, deriving, based on the respective first private key, a respective first party public key, and mapping the respective first party public key to the respective data item; obtaining, from each other
20 respective party, a respective party public key mapped to each respective data item, wherein a respective combined public key is generated based on each respective party public key mapped to that respective data item, the respective combined public key also being mapped to that respective data item. The method further comprises obtaining, from each other respective party, a respective ordered set of respective party encrypted shuffle
25 keys, each respective party encrypted shuffle key being generated by encrypting a respective party shuffle key with a respective scalar value; and for each respective data item, generating a respective sequence of elements by placing the respective first party shuffle key corresponding to that respective data item at a first position in the respective sequence and placing each respective party encrypted shuffle key corresponding to that
30 respective data item at a respective position in the respective sequence based on the respective index of the respective party, wherein the respective sequences of elements are placed in an initial ordered set. The method also comprises encrypting each respective

sequence of elements with a first scalar value to generate a respective first encrypted sequence; generating a first ordered set of respective first encrypted sequences of elements, wherein the order of the respective first encrypted sequences of elements differs compared to the initial ordered set of respective sequences of elements; and sending the
5 first ordered set of respective first encrypted sequences of elements to at least the second party.

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
10 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 object, such as a card, token, payment, etc.
15 The shuffle keys themselves may be first generated, from which a private key is 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.

20

Each other party performs an equivalent process of deriving a shuffle key, private key and public key for each data item, with each public key being mapped to the data item. Therefore each data item is mapped to multiple public keys. The multiple public keys mapped to the same data item are used to derive a combined public key which is also
25 mapped to that data item. The combined public key is used to select a data item. The combined public key may be generated by the first party or a different party.

Each other party (e.g. second party, third party, etc.) encrypts their shuffle keys with a respective scalar value and places the encrypted shuffle keys in an ordered set. These
30 parties send their ordered sets of encrypted shuffle keys to the first party. For each data item, the first generates a respective sequence of elements. The first element in each sequence is the shuffle key generated by the first party and corresponds to the public key

(i.e. the public key derived from the shuffle key) is mapped to the respective data item. The second element in each sequence is the encrypted shuffle key received from the second party and corresponds to the public key mapped to the respective data item. In other words, each sequence of elements has a (encrypted) shuffle key generated by each party for
5 the same data item.

The sequences have an initial order - the first sequence corresponding to the first data item, the second sequence corresponding to the second data item, and so on. The first party encrypts each sequence, separately, with a first scalar value. The first party then changes
10 the ordered of the sequences - producing a first order of encrypted sequences, and sends the first order of encrypted sequences to the second party.

Due to being encrypted with the first scalar value, the second party cannot determine which sequence of elements (i.e. encrypted shuffle keys) corresponds to which sequence of public
15 keys. Thus the second party cannot determine which data items are mapped to which sequences. Moreover, since the shuffle keys are encrypted, the second party cannot derive any private keys necessary for selecting a data item. The set of data items can therefore be considered as shuffled since the order or value cannot be determined.

20 The second party obtains the first order of encrypted sequences from the first party. The second party decrypts each encrypted sequence using the inverse of the scalar value used by the second party to initially encrypt the second party's shuffle keys. The second party then encrypts each resulting sequence of encrypted elements with a different scalar value, before shuffling the sequences to produce a second order of encrypted sequences.
25 Depending on the number of parties in the group, the second party may send the second order of encrypted sequences to a third party (if there are three or more parties), or to the first party (if there are only two parties).

In embodiments, certain stages of the set shuffling procedure may be attested to by
30 submitting commitments to the blockchain. For instance, the first party may commit to the mapping of data items to public keys, the order of shuffle keys, and the first order of encrypted sequences. 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).

Furthermore, in embodiments the combined public keys are used to lock respective UTXOs
5 of one or more blockchain transactions. Thus when a selection is made, the corresponding UTXO may be spent to show that the data item mapped to the combined public key has been chosen. This creates an immutable record of the selected data item.

BRIEF DESCRIPTION OF THE DRAWINGS

10 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:

Figure 1 is a schematic block diagram of a system for implementing a blockchain,
15 Figure 2 schematically illustrates some examples of transactions which may be recorded in a blockchain,
Figure 3 schematically illustrates an example system for implementing a set shuffling protocol,
Figure 4 schematically illustrates an example mental poker protocol,
20 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,
Figure 7 shows another example relationship between private keys, public keys and shuffle keys,
Figure 8 shows an outline of Alice collecting shuffle points from Bob, Claire and Zach and
25 organising them in the same order,
Figure 9 shows an outline of set encryption and shuffling performed by Alice, where index i represents a new element order,
Figure 10 shows an outline of set encryption and shuffling performed by Bob, Zach and all other group members in between, where index j and k represents a new element order
30 within the set,
Figure 11 shows a summary of the procedure of encrypting the set with distinct keys for each element by each group member,

Figure 12 shows an example element mapping procedure performed by Alice and Bob for a three card game,

Figure 13 shows an example setup for Alice and Bob to perform address exchange,

Figure 14 shows an example point multiplication and shuffling procedure for paired private

5 keys between Alice and Bob, and

Figure 15 shows example element selection, key exchange and inverse multiplication phases of an example game.

DETAILED DESCRIPTION OF EMBODIMENTS

1. EXAMPLE SYSTEM OVERVIEW

Figure 1 shows an example system 100 for implementing a blockchain 150. The system 100 may comprise a packet-switched network 101, typically a wide-area internetwork such as the Internet. The packet-switched network 101 comprises a plurality of blockchain nodes 104 that may be arranged to form a peer-to-peer (P2P) network 106 within the packet-switched network 101. Whilst not illustrated, the blockchain nodes 104 may be arranged as
15 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
20 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
25 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.

30 The blockchain 150 comprises a chain of blocks of data 151, wherein a respective copy of the blockchain 150 is maintained at each of a plurality of blockchain nodes 104 in the distributed or blockchain network 106. As mentioned above, maintaining a copy of the

blockchain 150 does not necessarily mean storing the blockchain 150 in full. Instead, the blockchain 150 may be pruned of data so long as each blockchain node 150 stores the block header (discussed below) of each block 151. Each block 151 in the chain comprises one or more transactions 152, wherein a transaction in this context refers to a kind of data structure. The nature of the data structure will depend on the type of transaction protocol used as part of a transaction model or scheme. A given blockchain will use one particular transaction protocol throughout. In one common type of transaction protocol, the data structure of each transaction 152 comprises at least one input and at least one output. Each output specifies an amount representing a quantity of a digital asset as property, an example of which is a user 103 to whom the output is cryptographically locked (requiring a signature or other solution of that user in order to be unlocked and thereby redeemed or spent). Each input points back to the output of a preceding transaction 152, thereby linking the transactions.

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

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

In a given present transaction 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
5 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
10 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
15 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
20 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
25 (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.

30 According to an output-based transaction protocol such as bitcoin, when a party 103, such as an individual user or an organization, wishes to enact a new transaction 152j (either manually or by an automated process employed by the party), then the enacting party

sends the new transaction from its computer terminal 102 to a recipient. The enacting party or the recipient will eventually send this transaction to one or more of the blockchain nodes 104 of the network 106 (which nowadays are typically servers or data centres, but could in principle be other user terminals). It is also not excluded that the party 103 enacting the

5 new transaction 152j could send the transaction directly to one or more of the blockchain nodes 104 and, in some examples, not to the recipient. A blockchain node 104 that receives a transaction checks whether the transaction is valid according to a blockchain node protocol which is applied at each of the blockchain nodes 104. The blockchain node protocol typically requires the blockchain node 104 to check that a cryptographic signature in the

10 new transaction 152j matches the expected signature, which depends on the previous transaction 152i in an ordered sequence of transactions 152. In such an output-based transaction protocol, this may comprise checking that the cryptographic signature or other authorisation of the party 103 included in the input of the new transaction 152j matches a condition defined in the output of the preceding transaction 152i which the new transaction

15 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 in the output of the preceding transaction 152i. Alternatively it could simply be fixed by the

20 blockchain node protocol alone, or it could be due to a combination of these. Either way, if the new transaction 152j is valid, the blockchain node 104 forwards it to one or more other blockchain nodes 104 in the blockchain network 106. These other blockchain nodes 104 apply the same test according to the same blockchain node protocol, and so forward the new transaction 152j on to one or more further nodes 104, and so forth. In this way the new

25 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

30 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
5 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
10 154 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
15 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
20 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
25 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
30 155 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
5 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
10 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
15 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
20 150. 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
25 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
30 forms the first transaction of the new block 151n. The proof-of-work signals the intent of the node that constructs the new block to follow the protocol rules allowing this special transaction to be redeemed later. The blockchain protocol rules may require a maturity

period, for example 100 blocks, before this special transaction may be redeemed. Often a regular (non-generation) transaction 152 will also specify an additional transaction fee in one of its outputs, to further reward the blockchain node 104 that created the block 151n in which that transaction was published. This fee is normally referred to as the “transaction fee”, and is discussed below.

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

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

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

Some or all of the parties 103 may be connected as part of a different network, e.g. a network overlaid on top of the blockchain network 106. Users of the blockchain network (often referred to as “clients”) may be said to be part of a system that includes the

blockchain network 106; however, these users are not blockchain nodes 104 as they do not perform the roles required of the blockchain nodes. Instead, each party 103 may interact with the blockchain network 106 and thereby utilize the blockchain 150 by connecting to (i.e. communicating with) a blockchain node 106. Two parties 103 and their respective
5 equipment 102 are shown for illustrative purposes: a first party 103a and his/her respective computer equipment 102a, and a second party 103b and his/her respective computer equipment 102b. It will be understood that many more such parties 103 and their respective computer equipment 102 may be present and participating in the system 100, but for convenience they are not illustrated. Each party 103 may be an individual or an
10 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.

15 The computer equipment 102 of each party 103 comprises respective processing apparatus comprising one or more processors, e.g. one or more CPUs, GPUs, other accelerator processors, application specific processors, and/or FPGAs. The computer equipment 102 of each party 103 further comprises memory, i.e. computer-readable storage in the form of a non-transitory computer-readable medium or media. This memory may comprise one or
20 more memory units employing one or more memory media, e.g. a magnetic medium such as hard disk; an electronic medium such as an SSD, flash memory or EEPROM; and/or an optical medium such as an optical disc drive. The memory on the computer equipment 102 of each party 103 stores software comprising a respective instance of at least one client application 105 arranged to run on the processing apparatus. It will be understood that any
25 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
30 networked resources, such as cloud computing resources accessed via the user terminal.

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

The client application 105 comprises at least a “wallet” function. This has two main functionalities. One of these is to enable the respective party 103 to create, authorise (for example sign) and send transactions 152 to one or more bitcoin nodes 104 to then be
10 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

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
20 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 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
30 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). 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 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.

10 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
15 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
20 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
25 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
30 104 applies the same protocol, then assuming the transaction 152j is valid, this means it will soon be propagated throughout the whole network 106.

Once admitted to the ordered pool of pending transactions 154 maintained at a given blockchain node 104, that blockchain node 104 will start competing to solve the proof-of-work puzzle on the latest version of their respective pool of 154 including the new transaction 152 (recall that other blockchain nodes 104 may be trying to solve the puzzle based on a different pool of transactions 154, but whoever gets there first will define the set of transactions that are included in the latest block 151. Eventually a blockchain node 104 will solve the puzzle for a part of the ordered pool 154 which includes Alice's transaction 152j). Once the proof-of-work has been done for the pool 154 including the new transaction 152j, it immutably becomes part of one of the blocks 151 in the blockchain 150. Each transaction 152 comprises a pointer back to an earlier transaction, so the order of the transactions is also immutably recorded.

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

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

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

The preceding transaction T_{x0} may already have been validated and included in a block 151 of the blockchain 150 at the time when Alice creates her new transaction T_{x1} , or at least by the time she sends it to the network 106. It may already have been included in one of the blocks 151 at that time, or it may be still waiting in the ordered set 154 in which case it will soon be included in a new block 151. Alternatively T_{x0} and T_{x1} could be created and sent to the network 106 together, or T_{x0} could even be sent after T_{x1} 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 T_{x0} 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.

30

The locking script (aka scriptPubKey) is a piece of code written in the domain specific language recognized by the node protocol. A particular example of such a language is called

“Script” (capital S) which is used by the blockchain network. The locking script specifies what information is required to spend a transaction output 203, for example the requirement of Alice’s signature. Unlocking scripts appear in the outputs of transactions. The unlocking script (aka scriptSig) is a piece of code written the domain specific language that provides
 5 the information required to satisfy the locking script criteria. For example, it may contain Bob’s signature. Unlocking scripts appear in the input 202 of transactions.

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

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

$$\text{<Sig } P_A \text{> } \text{< } P_A \text{> } || [\text{Checksig } P_A]$$

30 where “||” represents a concatenation and “<...>” means place the data on the stack, and “[...]” is a function comprised by the locking script (in this example a stack-based language). Equivalently the scripts may be run one after the other, with a common stack, rather than

concatenating the scripts. Either way, when run together, the scripts use the public key P_A of Alice, as included in the locking script in the output of T_{X_0} , to authenticate that the unlocking script in the input of T_{X_1} contains the signature of Alice signing the expected portion of data. The expected portion of data itself (the “message”) also needs to be
5 included in order to perform this authentication. In embodiments the signed data comprises the whole of T_{X_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).

The details of authentication by public-private cryptography will be familiar to a person
10 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, thus enabling any holder of the public key to authenticate the signature. Note therefore
15 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.

If the unlocking script in T_{X_1} meets the one or more conditions specified in the locking script
20 of T_{X_0} (so in the example shown, if Alice’s signature is provided in T_{X_1} and authenticated), then the blockchain node 104 deems T_{X_1} valid. This means that the blockchain node 104 will add T_{X_1} to the ordered pool of pending transactions 154. The blockchain node 104 will also forward the transaction T_{X_1} to one or more other blockchain nodes 104 in the network 106, so that it will be propagated throughout the network 106. Once T_{X_1} has been validated and
25 included in the blockchain 150, this defines $UTXO_0$ from T_{X_0} as spent. Note that T_{X_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_1} 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_0} is already spent (i.e. whether
30 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.

5 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.

10 Note that in UTXO-based transaction models, a given UTXO needs to be spent as a whole. It cannot “leave behind” a fraction of the amount defined in the UTXO as spent while another fraction is spent. However the amount from the UTXO can be split between multiple outputs of the next transaction. E.g. the amount defined in $UTXO_0$ in T_{X_0} can be split between multiple UTXOs in T_{X_1} . Hence if Alice does not want to give Bob all of the amount
15 defined in $UTXO_0$, she can use the remainder to give herself change in a second output of T_{X_1} , or pay another party.

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,
20 T_{X_0} may be rejected by the blockchain nodes 104, and hence although technically valid, may not be propagated and included in the blockchain 150 (the node protocol does not force blockchain nodes 104 to accept transactions 152 if they don’t want). In some protocols, the transaction fee does not require its own separate output 203 (i.e. does not need a separate UTXO). Instead any difference between the total amount pointed to by the input(s) 202 and
25 the total amount of specified in the output(s) 203 of a given transaction 152 is automatically given to the blockchain node 104 publishing the transaction. E.g. say a pointer to $UTXO_0$ is the only input to T_{X_1} , and T_{X_1} has only one output $UTXO_1$. If the amount of the digital asset specified in $UTXO_0$ is greater than the amount specified in $UTXO_1$, then the difference may be assigned (or spent) by the node 104 that wins the proof-of-work race to create the block
30 containing $UTXO_1$. Alternatively or additionally however, it is not necessarily excluded that a transaction fee could be specified explicitly in its own one of the UTXOs 203 of the transaction 152.

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.

- 5 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.

10

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
15 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.

20

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
25 (and thus fixed at the time of signing).

30

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.

3. SIDE CHANNEL

- 5 As shown in Figure 1, the client application on each of Alice and Bob’s computer equipment 102a, 120b, respectively, may comprise additional communication functionality. This additional functionality enables Alice 103a to establish a separate side channel 107 with Bob 103b (at the instigation of either party or a third party). The side channel 107 enables exchange of data separately from the blockchain network. Such communication is
- 10 sometimes referred to as “off-chain” communication. For instance this may be used to exchange a transaction 152 between Alice and Bob without the transaction (yet) being registered onto the blockchain network 106 or making its way onto the chain 150, until one of the parties chooses to broadcast it to the network 106. Sharing a transaction in this way is sometimes referred to as sharing a “transaction template”. A transaction template may lack
- 15 one or more inputs and/or outputs that are required in order to form a complete transaction. Alternatively or additionally, the side channel 107 may be used to exchange any other transaction related data, such as keys, negotiated amounts or terms, data content, etc.
- 20 The side channel 107 may be established via the same packet-switched network 101 as the blockchain network 106. Alternatively or additionally, the side channel 301 may be established via a different network such as a mobile cellular network, or a local area network such as a local wireless network, or even a direct wired or wireless link between Alice and Bob’s devices 102a, 102b. Generally, the side channel 107 as referred to anywhere
- 25 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
- 30 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

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

- 5 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 point to obtain a new point. Multiplying point P by scalar a forms a new point on the EC, P^* , defined by:

$$P^* = aP$$

10

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

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

- 15 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 compute P , given P^* and scalar a . This is done by performing point multiplication of the inverse of a , as such:

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

20

Where, the inverse scalar is defined by:

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

- 25 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} .

- 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:

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

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

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

- 5 Homomorphic encryption describes the property that encrypted data can be further 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.

10

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.

15

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 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.

20

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:

25

1. Interaction outline and element mapping:
 - a. Agree to the rules,
 - 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 ,

30

- 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.
- 5
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',
- 10
- b. We consider this random as no player can know what any of the encrypted values represent,
4. Decryption:
- a. After selecting elements, each party reveals their decryption keys: a^{-1} , b^{-1} and c^{-1} respectively,
- 15
- b. Now each player can reveal their original public keys: P_1 , P_2 and P_3 .
5. Outcome:
- a. After decrypting the element each party can reveal which element/card they have: 'Ace', 'King or 'Queen'.

20 5. SET SHUFFLING

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 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.

30 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 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, such as make a vote.

- 5 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.
- 10 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
- 15 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
- 20 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

25 private key may be chosen as the x-coordinate of the shuffle key. A public key is then 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

30 the shuffle key. Preferably, the method also allows the shuffle key to be derivable from the 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 some examples, in addition to or instead of sending and/or publishing the mapping itself,
- 5 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.
- 10 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 keys is generated for the second data item in the ordered set of data items which is mapped
- 15 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.
- 20 Each other party (e.g. the second party Bob 103b) performs the same actions as Alice 103a to generate a set of shuffle keys (one per data item), a set of private keys (one per data item), and a set of public keys (one per data item). Each public key is mapped to a particular data item. As there are multiple parties, this means there are multiple public keys mapped to the same data item. For example, if there are three parties, each data item is mapped to
- 25 three public keys. The multiple publics mapped to the same data item are used to generate a combined public key that is mapped to that data item. For instance, the public keys may be combined via point addition. Alice 103a may generate the combined public keys. That is, Alice 103a may receive, for each data item, a respective public key from each respective party 103. Alternatively, another party 103 (e.g. Bob 103b) may generate the combined
- 30 public keys. The combined public keys may be independently generated by each party 103. The combined public keys are made available to each party 103, e.g. Alice 103a may send them to each party 103.

Preferably, each party 103 publishes their mappings of public keys to data items. This guarantees that they can all check the combined mapping is correct, e.g. to prevent cheating. Alice may collect all of the mappings and perform the checks.

5

Each other party (e.g. Bob 103b and Claire 103c) encrypts each of their shuffle keys with a respective scalar value, as described in section 4.1 above. Each shuffle key thus becomes an encrypted shuffle key, encrypted with the same scalar value. That is, Bob 103b encrypts each of his shuffle keys with a second scalar value (the same value for each shuffle key).

10 These encrypted shuffle keys may be referred to as “first encrypted shuffle keys”, referring to the fact that they correspond to the shuffle keys that have been encrypted by the second party Bob 103b. Claire 103c performs similar actions with a different, third scalar value (the same value for each shuffle key).

15 Each other party then sends to Alice 103a an ordered set of encrypted shuffle keys, where the order is based on the data item mapped to the shuffle keys. That is, the encrypted shuffle key mapped to the first data item is placed first in the ordered set, then the shuffle key mapped to the second data item, and so on.

20 Alice 103a therefore receives an ordered set of encrypted shuffle keys from each other party, e.g. one set from Bob 103b, one set from Claire 103c, etc. Alice 103a then generates a respective sequence of elements, one sequence per data item. The first element in each sequence is the shuffle key generated by Alice 103a for that data item. The second element in each sequence is the encrypted shuffle key generated by Bob 103b for that data item. The
25 third element in each sequence is the encrypted shuffle key generated by Bob 103b for that data item (if Claire 103c is present in the group). Thus the number of elements in each sequence corresponds to the number of parties 103 in the group. Each element (shuffle key, or encrypted shuffle key) in a given sequence corresponds to (i.e. is mapped to) the same data item. The sequences of elements have an initial order. The first sequence
30 corresponding to the first data item, the second sequence corresponding to the second data item, and so on. Although Alice 103a has knowledge of which encrypted element corresponds to which data item, she cannot know the actual decrypted values. As these

elements will be further encrypted she will not be able to deduce the associations between the encrypted elements and the data items once encrypted and shuffled by the other parties.

5 Alice 103a encrypts each sequence of elements with the same first scalar value. That is, the first sequence is encrypted with the first scalar value, the second sequence is encrypted with the first scalar value, and so on. Each sequence is thus a respective “first encrypted sequence). Alice 103a does this so as to mask the elements from the other parties. Then, the order of the first encrypted sequences is shuffled (i.e. changed) such that the positions
10 of one or more of the sequences differs before and after encryption with the first scalar value. The shuffled sequence of encrypted keys is referred to as the first ordered set of first encrypted sequences. Again, “first” here refers to the sequences being generated by the first party, Alice 103a. Alice 103a then sends the first ordered set of first encrypted sequences to Bob 103b.

15

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

25 The second party, e.g. Bob 103b, receives the first ordered set of first encrypted sequences from Alice 103a. Bob 103b uses an inverse of the second scalar value to decrypt each of the first encrypted sequences. Note that this does not fully decrypt the sequences as they are encrypted with Alice’s scalar value (and some of the shuffle keys that form the elements are also encrypted with respective scalar values used by other parties). Bob 103b then encrypts
30 each of the resulting encrypted sequences with a different respective scalar value. That is, first, the same scalar value is used to decrypt each sequence, and then different scalars value are used to encrypt the resulting sequences. The first encrypted sequences, once

decrypted with the second scalar value and encrypted with the difference respective scalar values will be referred to as second encrypted sequences (having been encrypted by the second party Bob 103b). Bob 103b shuffles the order of the second encrypted sequences to obtain a second ordered set of second encrypted sequences. The order of the second encrypted sequences differs from the order of the first encrypted sequences.

If Alice 103 and Bob 103b are the only parties in the group, Bob 103b sends the second ordered set of second encrypted sequences to Alice 103a. If the group includes one or more additional parties, e.g. Claire 103c, Bob 103b sends the second ordered set of second encrypted sequences to Claire 103c. Claire 103c then performs the same actions as Bob 103b except for using a third scalar value for the decryption stage. The process of decrypting with a single scalar, encrypting with different scalars, shuffling and sharing the encrypted sequences continues until each party 103 has performed this process. This includes Alice 103a. That is, Alice 103a receives an ordered set of encrypted sequences from the last party 103 in the group. This ordered set may be referred to as a “penultimate ordered set of penultimate encrypted sequences”, referring to the fact that Alice 103a will undergo the final round of encryption and shuffling of the sequences. Alice 103a encrypts each of the respective sequences with a different respective scalar value to obtain a set of final encrypted sequences. She then shuffles the order of the final encrypted sequences to obtain a final ordered set of final encrypted sequences, and sends the final ordered set to at least Bob 103b, and optionally to each other party 103. After this final round of encryption, each encrypted sequence has been encrypted with n different scalar values, where n is the number of parties in the group.

Note that the respective scalar values used by the other respective parties 103 to encrypt their individual shuffle keys, and the scalar value used by Alice 103a to initially encrypt the sequences of elements, may be referred to as “first-round scalar values”. In contrast, the different scalar values used by each respective party 103 to encrypt the sequences of elements (i.e. after decrypting with the respective first-round scalar value) may be referred to as “second-round scalar values”. This illustrates that there are two rounds of encryption as part of the shuffling process.

Alice 103a then selects one of the final encrypted sequences of elements 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 selection involves hashing the selected encrypted sequence 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.

As described above, each final encrypted sequence is encrypted with a plurality of different scalar values, one for each party 103. To decrypt the final encrypted sequence selected by Alice 103a, Alice 103a requests from the other parties 103 inverse values of the scalar values that those parties 103 used to encrypt the final encrypted sequence. Alice 103a uses the received inverse values, and the inverse value of the scalar value used by herself to encrypt the final encrypted sequence, to decrypt and reveal a sequence of shuffle keys - "a selected sequence of shuffle keys". Each selected shuffle key is thus revealed in its raw form, i.e. as an x-coordinate and y-coordinate of an elliptic curve point. From each selected shuffle key, Alice 103a then derives the corresponding private key (e.g. the x-coordinate of the shuffle key). Alice 103a then derives a public key from each private key. Alice 103a generates a combined public key from each of the derived public keys and, using the mapping of combined public keys to data items, Alice 103a determines which data item is mapped to the combined public key. Alice 103a has therefore determined which object, etc. she has selected.

When another party (e.g. Bob 103b) commits to a selection of one of the final encrypted sequences, Alice 103a sends to Bob 103b an inverse of the scalar value used by Alice 103a to encrypt the selected final encrypted sequence. The other parties 103 perform similar actions to Alice 103a in order to decrypt a selected final encrypted sequence and determine which object, etc. has been selected.

In some examples, each of the combined 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, Alice 103a uses the private keys derived from the selected shuffle keys to generate a combined private key. Alice 103a may then use
5 combined private key to unlock one of the UTXOs locked to the combined public key derived from the selected shuffle keys, e.g. by generating a signature using the combined private key.

Each UTXO may also be locked to a public key of a different entity outside of the group, e.g.
10 an oracle or a trusted third party (TTP), the TTP not being the same as the third party, Claire 103c. In these examples, a signature from the TTP or equivalent entity is required to unlock the UTXO.

In some examples, Alice 103a may commit a secret value to the blockchain prior to making a
15 selection of the final encrypted sequence. Then, when unlocking the UTXO locked to the selected combined public key, Alice 103a may include a commitment of the secret value and a salt (e.g. random) value in the transaction that unlocks the UTXO. Alice 103a may later reveal the salt value and the secret value to prove that it was Alice 103a that unlocked the UTXO.

20 As described in more detail below, each party may generate a respective attestation transaction (also referred to as a “fair-play transaction”) to attest to the shuffle keys which each party 103 has generated. For instance, Alice 103a may generate a first attestation transaction which, when executed, can be used to verify the first set of shuffle keys
25 generated by Alice 103a. To verify the shuffle keys, an unlocking script that attempts to unlock the locking the script of the first attestation transaction includes, for each respective data item, a respective candidate x-coordinate, a respective candidate y-coordinate, and a respective candidate scalar value. When executed, the locking script verifies that each respective candidate x-coordinate derives to (i.e. can be used to generate) a respective one
30 of the public keys corresponding to a respective one of the shuffle keys generated by Alice 103a. The locking script also verifies that each pair of respective candidate x-coordinate and respective candidate y-coordinate derives to a respective one of the shuffle keys generated

by Alice 103a. The locking script also verifies that each of the shuffle keys derived from a pair of candidate coordinates, together with the candidate scalar value, derives to one of the encrypted shuffle keys generated by Alice 103a. Alice 103a may use Bob's attestation transaction to verify Bob's shuffle keys. In generate Alice 103a may verify any party's shuffle
 5 keys.

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
 10 protocol will be described with reference to Bitcoin and the secp256k1 elliptic curve, but other blockchains and other elliptic curves may be used instead.

6.1 Outline and element mapping

Figure 5 illustrates the architecture of an example mental poker protocol. The initial stages
 15 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 interactioninteractions. 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
 20 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 interaction. One option is to represent real-world objects in string format where, for example, the three of hearts card could be
 25 represented as a string, as such:

'3Hearts'

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
 30 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:

$$P_i = s_i \cdot G$$

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.
- 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 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 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 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.

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.

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

5

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.

10

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

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

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

15 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.

25 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:

30

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

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 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 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 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 private-public key pair with a security reduction of factor 2.

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 have this agreement represented on the blockchain. The 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 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 rules using the stated element mappings.

5 In this protocol, transactions that represent 'bets' will be tied to public key addresses relating to private keys that will be revealed during the interaction. The addresses are formed from multiple private keys, so that no one party can unlock the funds before the interaction is played out.

10 If Alice 103a and Bob 103b both generate a private key, s_A and s_B , respectively, funds can be tied through the point addition of the public keys, P_A and P_B . Neither party can sign for the address $(P_A + P_B)$ without the other's private key. This forms the basis of an on-chain settlement mechanism.

15 To enable an on-chain settlement mechanism, one party cannot have knowledge of the entire private-public key mappings for each object. In this setup, objects within a set are made up of multiple private-public key pairs, with a contribution from each party involved in the interaction. This is referred to herein as many-to-one element mapping.

20 Consider a group: $A, B, C, \dots, Z$, who wish to participate in a set shuffle with on-chain settlement, each party 103 creates an element mapping for the set, as such:

1. Randomly select shuffle points P'_i for the set,
2. Extract the x-coordinates of these shuffle points to use as private keys, s_i ,
3. Determine public keys P_i for these private keys.

25 It is important that they do not reveal their shuffle points or private keys. However, public keys will be shared.

30 The group then share their public key mappings for each element in the set, each committing their mapping to a blockchain transaction. It is beneficial that all members within the group can see all element mappings. This will allow them to ensure that single address formed for each element, from all individually contributions, is the correct value.

From the shared public key mappings, the group can then form a list of associated public keys for each element, with one contribution from each member of the group, as such:

$$P_{A1}, P_{B1}, P_{C1}, \dots, P_{Z1} = 'Element_1'$$

$$P_{A2}, P_{B2}, P_{C2}, \dots, P_{Z2} = 'Element_2'$$

...

$$P_{An}, P_{Bn}, P_{Cn}, \dots, P_{Zn} = 'Element_n'$$

Using this many-to-one public key element map, the group can begin to create transactions with locking scripts based on the combined public key representation of elements. For example, funds could be locked to combined public key for $Element_1$, as such:

Example Transaction for Element 1, $TxID_{E1}$			
Inputs		Outputs	
Value	Script	Value	Script
1 BSV	$\langle Sig(P_A, Tx_{FA}) \rangle$ $\langle P_A \rangle$	2 BSV	OP_DUP OP_HASH160 $\langle P_{Element\ 1} \rangle$ OP_EQUALVERIFY OP_CHECKSIG
1 BSV	$\langle Sig(P_B, Tx_{FB}) \rangle$ $\langle P_B \rangle$		

Where, the public key address for $Element_1$, $P_{Element\ 1}$, is defined as the EC point addition of all public key maps from the group, as such:

$$P_{Element\ 1} = P_{A1} + P_{B1} + P_{C1} + \dots + P_{Z1}$$

This ensures that a party can only unlock this transaction if all the associated private keys are revealed for that element. These private keys will be exchanged as shuffle keys during

the shuffle phase and revealed at the end of the interaction to enable one party to unlock the transaction associated with their revealed element.

Each party can check that the single address, $P_{Element\ 1}$ for example, has been correctly
5 calculated using all of the on-chain element mappings that have been committed to.

6.2 Shuffling and element encryption

As described each party has an individual element mapping, containing private, public and shuffle key per each element. During the shuffle phase, each party will encrypts their shuffle
10 key points using point multiplication by some secret scalar.

Each set of encrypted shuffle keys will be sent to one party (e.g. Alice 103a), who will collect and organise all the shuffle points, as shown in Figure 8:

15 Note than when collecting the encrypted values, Alice:

1. Maintains the 'vertical' sequence, so that shuffle points align to their mapped elements,
2. Maintains the 'horizontal' sequence, so that the order in which they are shuffled is the order in which they are organised.

20

After collating the groups shuffle points into one set, Alice will then encrypt the rest of the set with her chosen encryption key a . She will then reorder the set, thus shuffling them, as shown in Figure 9.

25 Alice then sends this encrypted and shuffled set to the next party in the order she sequenced the shuffle points, e.g. Bob 103b. Bob will then encrypt the other parties shuffle points with his chosen encryption key b , and reorder the set. Bob will then send this set to Claire 103c, who will do the same. This is repeated until the final party member in the sequence, Zach, has completed the process. The process is summarised in Figure 10.

30

The set is considered shuffled, where no one party can determine the sequence of elements. Form this point, the order of the elements will remain fixed. This is because the

next stage involves group members encrypting elements (rows) with distinct keys, which will be defined by the index/order.

The group now decrypt the entire set, removing their single encryption keys $a, b, c, \dots, z$.

- 5 Then encrypt each of the shuffle points representing one element with a discrete encryption key, as shown in Figure 11.

This will enable individual element selection and reveals. This is necessary as each element corresponds to a set of private keys to unlock funds associated to a public key. It would not
10 be beneficial to utilise this method for the reveal phase as it would mean that anyone could decrypt for any of the elements, thus revealing all private keys. This completes the multi-deck shuffle phase, providing the group with a common shuffled and distinctly encrypted set.

15 6.3 Settlement

Once the set has been shuffled and encrypted, the group can proceed to the interaction, vote, etc.. As discussed, such interactions are usually based around group members selecting encrypted elements at random. Once elements have been selected, they request the relevant decryption keys to reveal the shuffle point. Here the revealed shuffle points
20 correspond to a set of private keys which can be used to sign for transactions related to the combined public key for that element.

6.4 Worked example: 2-player 3-card game

Suppose Alice 103a and Bob 103b want to play a mental poker game in which bets can be
25 placed and automatically settled P2P using blockchain transactions. The game considers a highest card winner random selection, where whoever selects the highest of the 3 cards $\{'Ace', 'King', 'Queen'\}$ wins all the funds.

The first stage would be to outline the game and associated rules for the game.

- 30 For this game, Alice and Bob agree that they will each select a card from the combined deck and whoever has the highest will win all the associated funds. The following object rules apply:

$$Ace > King > Queen$$

Which can be represented as:

$$P_1 > P_2 > P_3$$

- 5 1. Any player who selects the *Ace* card wins and should receive all funds.
 2. If no player selects the *Ace* card, we assume that the winning card is the
 King card and the player with that card receive all funds.
 3. Any player who selects the *Queen* card loses and should receive no funds.
- 10 Alice and Bob also determine the amount of funds: they are each willing to bet 1 BSV on this game.

After agreeing these rules Alice and Bob would proceed to create an initial transaction committing funds to their game and signing their agreement if the game rules.

- 15 Alice and Bob combine their ‘buy-ins’ into a single 2-of-2 multi-signature output, referred to as the *funding transaction*, as such:

Funding Transaction, $TxID_F$				
Inputs		Outputs		
Value	Script	Value	Script	
1 BSV	$\langle Sig(P_A, Tx_{FA}) \rangle$ $\langle P_A \rangle$	2 BSV	OP_2 $\langle P_A \rangle \langle P_B \rangle$ OP_2 OP_CHECKMULTISIG	
1 BSV	$\langle Sig(P_B, Tx_{FB}) \rangle$ $\langle P_B \rangle$	0 BSV	OP_0 OP_RETURN [Agreed game rules]	

- This transaction will be used as an input for the betting transaction that both Alice and Bob
- 30 sign prior to the game’s commencement. As a precaution Alice and Bob will also have to create a *refund transaction* for the funding transaction, to ensure these funds are not lost if either Alice and Bob becomes inactive after some time, T_R .

The *refund transaction* is signed and stored offline by both parties prior to the commencement of the game. It can only be redeemed after T_R , if no other spending/betting transactions have been made prior.

5

The outputs will be a standard P2PKH, returning the funds to their original address as such:

10

15

20

Refund Transaction, $TxID_R$			
Inputs		Outputs	
Value	Script	Value	Script
2 BSV	$\langle \text{Sig}(P_A, Tx_F) \rangle$ $\langle P_A \rangle$ $\langle \text{Sig}(P_B, Tx_F) \rangle$ $\langle P_B \rangle$	1 BSV	OP_DUP OP_HASH160 $\langle P_A \rangle$ OP_EQUALVERIFY OP_CHECKSIG
		1 BSV	OP_DUP OP_HASH160 $\langle P_B \rangle$ OP_EQUALVERIFY OP_CHECKSIG
nLocktime: T_R			

After the creation of the funding and refunding transactions, Alice and Bob can begin the mapping system for the objects, where $\{s_{A1}, s_{A2}, s_{A3}\}$ and $\{s_{B1}, s_{B2}, s_{B3}\}$ are the private keys derived by Alice and Bob respectively from their generated shuffle keys. As mentioned, the corresponding public keys formed from these private keys can be shared without revealing any information about the private key needed for signing.

25

After committing and sharing their respective public key mappings, Alice and Bob can now formulate the many-to-one mapping for each card:

30

$$\begin{aligned}
 (s_{A1} \cdot G), (s_{B1} \cdot G) &= 'Ace' \\
 (s_{A2} \cdot G), (s_{B2} \cdot G) &= 'King' \\
 (s_{A3} \cdot G), (s_{B3} \cdot G) &= 'Queen'
 \end{aligned}$$

This process is illustrated in Figure 12.

Both parties can now form a combined address scheme for each card value

{'Ace', 'King', 'Queen'} by point addition as such:

$$\begin{aligned} P_1 &= (s_{A1} + s_{B1}) \cdot G = (s_{A1} \cdot G) + (s_{B1} \cdot G) \\ P_2 &= (s_{A2} + s_{B2}) \cdot G = (s_{A2} \cdot G) + (s_{B2} \cdot G) \\ P_3 &= (s_{A3} + s_{B3}) \cdot G = (s_{A3} \cdot G) + (s_{B3} \cdot G) \end{aligned}$$

{ s_{A1}, s_{A2}, s_{A3} } and { s_{B1}, s_{B2}, s_{B3} } are secret to Alice and Bob respectively. Only someone who has both values for each private key, related to one address (P_1, P_2 and P_3) can create a digital signature for said address. This interaction is summarised in Figure 13.

Now that the public keys have been derived for each of the three cards, Alice and Bob can exchange and sign two transactions.

15

The first transaction has as a P2PKH address output where the public key is defined as:

$$P_1 = (s_{A1} \cdot G) + (s_{B1} \cdot G)$$

This address represents the Ace, which is the best card and should be immediately redeemable. The transaction will spend the funding transaction, as such:

20

Bet Transaction 1, $TxID_1$			
Inputs		Outputs	
Value	Script	Value	Script
2 BSV	$\langle Sig(P_A, Tx_F) \rangle$ $\langle P_A \rangle$ $\langle Sig(P_B, Tx_F) \rangle$ $\langle P_B \rangle$	2 BSV	OP_DUP OP_HASH160 $\langle P_1 \rangle$ OP_EQUALVERIFY OP_CHECKSIG
nLocktime: 0			

25

30 They then generate and sign a transaction related to a P2PK address where the public key is:

$$P_2 = (s_{A2} \cdot G) + (s_{B2} \cdot G)$$

As this address represents the King card, it is the second-best card and should only be redeemable if the Ace card is not. We therefore allow it to be redeemed after some time, t , as such the corresponding transaction will appear as follows:

5

10

Bet Transaction 2, $TxID_2$			
Inputs		Outputs	
Value	Script	Value	Script
2 BSV	$\langle Sig(P_A, Tx_F) \rangle$ $\langle P_A \rangle$ $\langle Sig(P_B, Tx_F) \rangle$ $\langle P_B \rangle$	2 BSV	OP_DUP OP_HASH160 $\langle P_2 \rangle$ OP_EQUALVERIFY OP_CHECKSIG
nLocktime: t			

15 The lock time t of the second transaction is intended to give sufficient time for the first transaction to be submitted and confirmed if that card (Ace) were to have been revealed during the game.

If that card (Ace) is not revealed during the game, then either Alice or Bob should be able to
 20 submit the second transaction for P_2 (King), after time t .

Any bet transaction nLocktime, t in this case, should be less than the specified refund transaction nLocktime T_R . It will always be the case that the refund transaction should not be redeemable before any bet transactions:

25
$$t < T_R$$

Alice and Bob both have signed versions/copies of each 'bet' transaction. Once cards have been revealed, Alice and Bob can submit the bet transaction that corresponds to their card. From here they can then unlock this transaction with the appropriate private keys and send
 30 the associated funds to any address.

Alice and Bob will each define some secret encryption keys, which are scalar values used for EC point multiplication. Alice will define 1 set scalar encryption key a and 3 distinct element encryption keys a_1, a_2 and a_3 . Bob will define 1 set scalar encryption key b and 3 distinct element encryption keys b_1, b_2 and b_3 . From here Alice and Bob are ready to begin the combined shuffling and distinct key multiplication, as shown in Figure 14.

From here Alice and Bob both have access to the following:

$$\begin{aligned} & a_1 b_1 \{P'_{A2}; P'_{B2}\} \\ & a_2 b_2 \{P'_{A3}; P'_{B3}\} \\ & a_3 b_3 \{P'_{A1}; P'_{B1}\} \end{aligned}$$

Neither party can derive any of the values without the other's inverse multiplication key. From here Alice selects element $a_2 b_2 \{P'_{A3}; P'_{B3}\}$ and asks Bob for inverse multiplication key b_2^{-1} . Bob then selects element $a_1 b_1 \{P'_{A2}; P'_{B2}\}$ and asks Alice for inverse multiplication key a_1^{-1} . This is shown in Figure 15.

As shown, Alice can derive that she has:

$$(s_{A3} \cdot G), (s_{B3} \cdot G) = 'Queen'$$

And Bob can derive that he has:

$$(s_{A2} \cdot G), (s_{B2} \cdot G) = 'King'$$

Having agreed that:

$$Ace > King > Queen$$

Bob can determine he has won the game. Further, now Bob has both private key values to sign and submit Transaction 2 after time t , collecting all associated funds.

25

The method outlined above offers at least the following advantages:

- Multi-deck shuffle – the protocol allows for shuffling multiple sets into a common encrypted set. This allows multiple parties to define their own element mapping using shuffle keys, which are then combined and shuffled into one common deck. This method can be extended to as many parties as

30

required and allows each party to contribute a private key to a single mapped object common to all parties.

- Fund locking to multiple keys – The multi-deck shuffle allows each party to contribute a private key to a mapped object. Funds can then be tied to mapped objects as the point addition of the respective public keys from each party. These public keys are revealed prior to the shuffling, but the private keys are attached to the elements which are encrypted and shuffled during the interactions. As objects are revealed at the end of the game, the associated private keys are also revealed allowing any party with them to unlock associated funds. This provides an efficient mechanism for settlement using blockchain transactions based on set shuffling and private key reveals as is only possible through the representation of private keys as shuffle keys combined with the multi-deck shuffle technique.

15 6.5 On-chain enforcement mechanisms

The on-chain settlement mechanism described above assumes a level of trust between Alice and Bob in the following two ways:

- Firstly, it assumes Alice and Bob both submit the correct private keys during shuffle phase, preventing the other party from unlocking any transactions related to betting. As the private keys are exchanged during the shuffle phase as multiplied shuffle points, it is not trivial for either party to verify they contain the correct private keys. Further, neither party will know whether these are the correct private keys until the game is completed, and they are given the inverse multiplication key to reveal the private keys.
- Secondly, it assumes that Alice and Bob both send the inverse multiplication key at the end of the game. It may be possible that Alice or Bob know they have lost the game after receiving the inverse multiplication key for their card, but before revealing the inverse multiplication key for the other player's card. At which point they could choose not to reveal the inverse multiplication key, to prevent the other player unlocking the winning funds.

This level of trust could be guaranteed using some off-chain verification platform, where participants have ratings attached to their public key profiles for the number of 'fair' shuffle interactions they have completed. Any form of cheating could be submitted to this platform
 5 to disincentivise players from doing so. Further, utilising blockchain enabled betting can allow players to commit very small amount of funds, micro-bets, as a further mitigation against cheating parties.

However, described below are two methods for guaranteeing fair play for the on-chain
 10 settlement mechanism described above.

6.5.1 Trusted third party – multi-signature output

The first enforcement mechanism requires a partially trusted third party, whose intervention is only required if there is some disagreement between the involved parties.
 15 This mechanism may be utilised as an additional signee for all transactions as an n-of-(m+1) multi-signature scheme. In the worked example described above, Alice and Bob would include some trusted 3rd party as a signee for all potential spending transactions, in a 2-of-3 multi-signature scheme.

20 As before, P_A and P_B represent the public addresses of Alice and Bob respectively. P_1 and P_2 represent the public addresses the potential winning object addresses, derive from secret keys for each of Alice and Bob, as such:

$$P_1 = (s_{A1} \cdot G) + (s_{B1} \cdot G) = 'Ace'$$

$$P_2 = (s_{A2} \cdot G) + (s_{B2} \cdot G) = 'King';$$

25

Finally, P_C represents the address of a trusted 3rd party. It is worth noting that this address could be made publicly available by a service provider. It could be included in any betting transaction without the service providers approval. The trusted 3rd party would only need to act in the event of a dispute.

30

In this protocol all bet transactions would require the unlocking address related to the respective card address for the bet, but also 2 signatures out of the possible 3 for P_A , P_B or P_C , as such:

5	Bet Transaction 1, $TxID_1$			
	Inputs		Outputs	
	Value	Script	Value	Script
10	2 BSV	$\langle Sig(P_A, Tx_F) \rangle$ $\langle P_A \rangle$ $\langle Sig(P_B, Tx_F) \rangle$ $\langle P_B \rangle$	2 BSV	OP_IF OP_DUP OP_HASH160 $\langle P_1 \rangle$ OP_EQUALVERIFY OP_CHECKSIG OP_ELSE OP_3 $\langle P_A \rangle \langle P_B \rangle \langle P_C \rangle$ OP_2 OP_CHECKMULTISIG OP_ENDIF
15	nLocktime: 0			

	Bet Transaction 2, $TxID_2$			
	Inputs		Outputs	
	Value	Script	Value	Script
20	2 BSV	$\langle Sig(P_A, Tx_F) \rangle$ $\langle P_A \rangle$ $\langle Sig(P_B, Tx_F) \rangle$ $\langle P_B \rangle$	2 BSV	OP_IF OP_DUP OP_HASH160 $\langle P_2 \rangle$ OP_EQUALVERIFY OP_CHECKSIG OP_ELSE OP_3 $\langle P_A \rangle \langle P_B \rangle \langle P_C \rangle$ OP_2 OP_CHECKMULTISIG OP_ENDIF
25	nLocktime: t			

The locking output can be summarised as follows:

1. Party can claim the funds if:
 - a. They can sign for winning address P_1/P_2 ,
 - b. Or, 2 out of the 3 sign from P_A , P_B and P_C .

These transactions allow Alice and Bob to play their game without any external intervention from the trusted 3rd party. If both parties interact fairly, they can sign and unlock funds for each game without needing the trusted 3rd parties' signature. However, should Alice or Bob believe the other party has cheated during the game, they can instate the help of the

5 trusted 3rd party. This will prevent either Alice or Bob from claiming the funds associated with the bet.

From here both parties can seek to come to a settlement with the trusted 3rd party's intervention. Alice and Bob could communicate with the trusted 3rd party, evidencing the

10 entire game using the irrefutable commitments to the blockchain made at the end of each phase of their game and interaction. The trusted 3rd party could therefore determine if a party had cheated and who should win the funds using these blockchain commitments as evidence.

15 This method assures that Alice and Bob play fairly, as any attempt to cheat would involve the 3rd party signing the funds to the other party, as well as taking some small fee for the intervention.

6.5.2 Fair-Play Transaction

20 The second enforcement mechanism is P2P and does not require any involvement from a trusted 3rd party. It involves parties committing to the public keys and encrypted values they send to one another in a separate transaction, referred to as a *fair-play transaction*. These values are derived in-script from the private keys committed. This ensures that the correct values are used in the game, otherwise the party will lose the funds associated with the *fair-*

25 *play transaction*.

After generating the funding and refund transactions, Alice and Bob will then each generate *fair-play transactions*, committing to their private-public key values. These *fair-play transactions* are generated, signed, and submitted to the blockchain after set shuffling and

30 encryption, but before the game and reveal takes place. These transactions ensure that Alice and Bob can derive both the public key addresses and encrypted shuffle keys from the same set of private keys. If either party attempts to cheat by exchanging shuffle keys that

don't correspond with public keys, they will lose any funds associated with the *fair-play transaction*.

So, Alice has received the public key addresses $\{P_{B1}, P_{B2}, P_{B3}\}$ from Bob, for bet

5 transactions, where:

$$P_{B1} = (s_{B1} \cdot G) = 'Ace'$$

$$P_{B2} = (s_{B2} \cdot G) = 'King'$$

$$P_{B3} = (s_{B3} \cdot G) = 'Queen'$$

10 Alice has also received the following encrypted shuffle keys, after shuffling and distinct key encryption with Bob, whose sequence, and values she does not know:

$$b_1\{P'_{B2}\}$$

$$b_2\{P'_{B3}\}$$

$$b_3\{P'_{B1}\}$$

Alice wants to ensure that the secret keys $\{s_{B1}, s_{B2}, s_{B3}\}$, associated with the public keys
15 $\{P_{B1}, P_{B2}, P_{B3}\}$, correspond to the encrypted shuffle keys $\{b_1P'_{B2}, b_2P'_{B3}, b_3P'_{B1}\}$ sent by Bob.

She can request Bob create a *fair-play transaction* prior to any bet transactions, committing funds to the derivation of the above values from their respective private keys. The following
20 script ensures that the public addresses and encrypted shuffle keys Bob has sent to Alice can be derived by the same single set of private keys, ensuring Bob has not cheated and that he sends the values to recoup these funds.

As Bob does not wish to reveal which encrypted shuffle key corresponds to which public key
25 address, it is important to utilise nested if conditions. A private key that derives for P_{B1} or P_{B2} can then derive for anyone of: $b_1\{P'_{B2}\}$, $b_2\{P'_{B3}\}$ or $b_3\{P'_{B1}\}$.

EC Point Multiplication:

- Inputs: EC Point P and scalar a ,
- 30 • Output: EC Point $P^* = a \cdot P$.

A further in-script function is used for outputting the shuffle point, P'_i , from the secret key, s_i , derived as:

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

5

Shuffle Point Calculation:

- Inputs: Secret key, s_i and $\sqrt{s_i^3 + 7}$
- Output: Shuffle key, P'_i .

10 Locking script: OP_DUP OP_TOALTSTACK OP_TOALTSTACK OP_DUP OP_DUP OP_DUP
OP_MUL OP_MUL <7> OP_ADD < p > OP_MOD OP_FROMALTSTACK OP_DUP OP_MUL
OP_EQUALVERIFY OP_FROMALTSTACK OP_SWAP

Unlocking script: < $+\sqrt{s_i^3 + 7}$ > < s_i >

15

The above script requires that the shuffle point y coordinate e.g., the scalar value for $+\sqrt{s_i^3 + 7}$, be passed into the script. It then checks that that the x coordinate, e.g. the secret key s_i , can derive to this value. The above script checks that the square of the square roots e.g., $s_i^3 + 7$, corresponds to the secret key derived.

20

The transaction would look as follows:

< s_{B1} >< $+\sqrt{s_{B1}^3 + 7}$ > < b_3 >< s_{B2} >< $+\sqrt{s_{B2}^3 + 7}$ > < b_1 >< s_{B3} >< $+\sqrt{s_{B3}^3 + 7}$ >
< b_2 >< $Sig(P_B, Tx_{FPB})$ >

25

Fair-Play Bob Transaction, $TxID_{FPB}$			
Inputs		Outputs	
Value	Script	Value	Script
2 BSV		2 BSV	[OP_TOALTSTACK]*6 OP_DUP [OP_FROMALTSTACK]*2 OP_ROT OP_FROMALTSTACK OP_DUP

	$\langle \text{Sig}(P_B, Tx_B) \rangle$ $\langle P_B \rangle$	OP_ROT[OP_FROMALTSTACK] OP_ROT*2 [OP_FROMALTSTACK] OP_ROT*2 OP_FROMALTSTACK OP_DUP OP_TOALTSTACK OP_ROT*2 OP_FROMALTSTACK OP_2DUP OP_NOTEQUAL OP_VERIFY OP_ROT OP_2DUP OP_NOTEQUAL OP_VERIFY OP_ROT OP_NOTEQUAL OP_VERIFY OP_DROP OP_DUP $\langle G \rangle$ [EC Point Multiplication] $\langle P_{B1} \rangle$ OP_EQUALVERIFY [Shuffle Point Calculation] [EC Point Multiplication] OP_DUP $\langle b_1 \cdot P'_{B2} \rangle$ OP_EQUAL OP_IF OP_VERIFY OP_ELSE OP_DUP $\langle b_2 \cdot P'_{B3} \rangle$ OP_EQUAL OP_IF OP_VERIFY OP_ELSE $\langle b_3 \cdot P'_{B1} \rangle$ OP_EQUAL OP_VERIFY OP_ENDIF OP_ENDIF OP_DUP $\langle G \rangle$ [EC Point Multiplication] $\langle P_{B2} \rangle$ OP_EQUALVERIFY [Shuffle Point Calculation] [EC Point Multiplication] OP_DUP $\langle b_1 \cdot P'_{B2} \rangle$ OP_EQUAL OP_IF OP_VERIFY OP_ELSE OP_DUP $\langle b_2 \cdot P'_{B3} \rangle$ OP_EQUAL OP_IF OP_VERIFY OP_ELSE $\langle b_3 \cdot P'_{B1} \rangle$ OP_EQUAL OP_VERIFY
--	--	--

5			OP_ENDIF OP_ENDIF OP_DUP <G> [EC Point Multiplication] < P_{B3} > OP_EQUALVERIFY [Shuffle Point Calculation] [EC Point Multiplication] OP_DUP < $b_1 \cdot P'_{B2}$ > OP_EQUAL OP_IF OP_VERIFY OP_ELSE 10 OP_DUP < $b_2 \cdot P'_{B3}$ > OP_EQUAL OP_IF OP_VERIFY OP_ELSE < $b_3 \cdot P'_{B1}$ > OP_EQUAL OP_VERIFY OP_ENDIF 15 OP_ENDIF OP_DUP OP_HASH160 < P_B > OP_EQUALVERIFY OP_CHECKSIG
---	--	--	---

20 The unlocking scripts for the above fair-play transaction would look as follows:

$$\langle s_{B1}a \rangle \langle +\sqrt{s_{B1}^3 + 7} \rangle \langle b_3 \rangle \langle s_{B2} \rangle \langle +\sqrt{s_{B2}^3 + 7} \rangle \langle b_1 \rangle \langle s_{A3} \rangle \langle +\sqrt{s_{A3}^3 + 7} \rangle \langle b_2 \rangle \langle \text{Sig}(P_B, Tx_{FPB}) \rangle$$

Where $\{s_{B1}, s_{B2}, s_{B3}\}$ represent Bob's two potential winning private keys, $\{b_2, b_1, b_3\}$

25 represent the respective scalar multiplication keys Bob used in the shuffle and

$\text{Sig}(P_B, Tx_{FPB})$ represents Bob's signature.

The locking output can be summarised as follows:

1. Check that the same secret keys haven't been submitted twice (a potential hack
30 to not reveal all keys),
 - a. If not equal, continue,

- b. If equal, reject unlocking script,
 2. Check if private key s_1 derives to the in script public key P_{B1} using EC point multiplication:
 - a. If yes, continue,
 - 5 b. If no, reject unlocking script.
 3. Convert same private key s_1 , to shuffle key P'_1 .
 4. Check if P'_1 multiplied by some scalar b_{check1} ,
 - a. Option 1: If equal to $b_1 \cdot P'_{B2}$, verify and exit 3)
 - b. Option 2: If equal to $b_2 \cdot P'_{B3}$, verify and exit 3)
 - 10 c. Option 3: If equal to $b_3 \cdot P'_{B1}$, verify and exit 3)
 - d. If not equal to any of the above, reject unlocking script.
 5. Repeat 2-4) for P_{B2} with inputs s_2 and some scalar b_{check2} .
 6. Repeat 2-4) for P_{B3} with inputs s_3 and some scalar b_{check3} .
 7. Check for Bob's signature.

15

This transaction would ensure that the private keys Bob has committed to, as part of any bet transaction with Alice, are the same as those encrypted and exchanged with Alice during the shuffle and game phase. If not, Bob will not be able to redeem the fair-play transaction and will lose more funds than he has committed to the bet transaction, thus guaranteeing

20 fair play for the on-chain settlement mechanism. The above would be repeated from Alice to Bob in the same way:

Fair-Play Alice Transaction, $TxID_{FPA}$			
Inputs		Outputs	
Value	Script	Value	Script
2 BSV		2 BSV	$[OP_TOALTSTACK]*6$ OP_DUP $[OP_FROMALTSTACK]*2$ OP_ROT $OP_FROMALTSTACK$ OP_DUP $OP_ROT[OP_FROMALTSTACK]$ OP_ROT*2 $[OP_FROMALTSTACK]$ OP_ROT*2 $OP_FROMALTSTACK$ OP_DUP $OP_TOALTSTACK$ OP_ROT*2 $OP_FROMALTSTACK$ OP_2DUP $OP_NOTEQUAL$

	$\langle \text{Sig}(P_A, Tx_A) \rangle$ $\langle P_A \rangle$	OP_VERIFY OP_ROT OP_2DUP OP_NOTEQUAL OP_VERIFY OP_ROT OP_NOTEQUAL OP_VERIFY OP_DROP OP_DUP <G> [EC Point Multiplication] $\langle P_{A1} \rangle$ OP_EQUALVERIFY [Shuffle Point Calculation] [EC Point Multiplication] OP_DUP $\langle a_1 \cdot P'_{B2} \rangle$ OP_EQUAL OP_IF OP_VERIFY OP_ELSE OP_DUP $\langle a_2 \cdot P'_{A3} \rangle$ OP_EQUAL OP_IF OP_VERIFY OP_ELSE $\langle a_3 \cdot P'_{A1} \rangle$ OP_EQUAL OP_VERIFY OP_ENDIF OP_ENDIF OP_DUP <G> [EC Point Multiplication] $\langle P_{A2} \rangle$ OP_EQUALVERIFY [Shuffle Point Calculation] [EC Point Multiplication] OP_DUP $\langle a_1 \cdot P'_{A2} \rangle$ OP_EQUAL OP_IF OP_VERIFY OP_ELSE OP_DUP $\langle a_2 \cdot P'_{A3} \rangle$ OP_EQUAL OP_IF OP_VERIFY OP_ELSE $\langle a_3 \cdot P'_{A1} \rangle$ OP_EQUAL OP_VERIFY OP_ENDIF OP_ENDIF
--	--	--

5			OP_DUP <G> [EC Point Multiplication] < P_{A3} > OP_EQUALVERIFY [Shuffle Point Calculation] [EC Point Multiplication] OP_DUP < $b_1 \cdot P'_{B2}$ > OP_EQUAL OP_IF OP_VERIFY OP_ELSE OP_DUP < $b_2 \cdot P'_{B3}$ > OP_EQUAL OP_IF OP_VERIFY OP_ELSE < $b_3 \cdot P'_{B1}$ > OP_EQUAL OP_VERIFY OP_ENDIF OP_ENDIF
10			
15			OP_DUP OP_HASH160 < P_A > OP_EQUALVERIFY OP_CHECKSIG

The unlocking scripts for the above fair-play transaction would look as follows:

20 $\langle s_{A1} \rangle \langle +\sqrt{s_{A1}^3 + 7} \rangle \langle a_3 \rangle \langle s_{A2} \rangle \langle +\sqrt{s_{A2}^3 + 7} \rangle \langle a_1 \rangle \langle s_{A3} \rangle \langle +\sqrt{s_{A3}^3 + 7} \rangle$
 $\langle a_2 \rangle \langle \text{Sig}(P_A, Tx_{FPA}) \rangle$

Where $\{s_{A1}, s_{A2}, s_{A3}\}$ represent Alice's two potential winning private keys, $\{a_3, a_1, a_2\}$ represent the respective scalar multiplication keys Alice used in the shuffle and

25 $\text{Sig}(P_B, Tx_{FPB})$ represents Alice's signature.

The above fair-play transactions may be generated and committed after shuffling, but before submitting any bet transactions. They guarantee that the public-private key pair submitted for element mapping and on-chain settlement addressing, correspond to the
 30 encrypted keys sent during the shuffle phase. This ensures that a party must send the correct private keys during the shuffling process. If they do not, then they will not be able to

redeem their *fair-play transaction*. If the *fair-play transaction* commits more funds they the bet transaction, we can assume parties will not cheat.

Further, as the scalar multiplication keys used to encrypt shuffle key are submitted to unlock
5 these *fair-play transactions* we can use their unlocking to guarantee parties can calculate the inverse multiplication key and decrypt their object values.

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
10 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
15 goals.

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 tokens to voting public key addresses, in a large transaction, called an *issuance transaction*.
20 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, where in this example there are 100 board members:

- 25 1. Creating a many-to-one map for 100 potential voting addresses, where each board member:
 - a. Selects 100 random shuffle points,
 - b. Extracts the x-coordinates from these shuffle points to use as private keys,
 - 30 c. Generate the public keys from these private keys,
 - d. Shares and broadcast the public keys, committing them to transaction outputs.

2. One board member would then combine a public key contribution from each board member for each voting address, using EC point addition.
3. The board members would then follow the multi-deck shuffle protocol outlined above.
- 5 4. Once completed, each board member would select an element and request the relevant decryption keys.
5. From these decryption keys each board member could reveal the 100 shuffle keys and therefore private keys contributions that would allow them to sign for one of the 100 combined public keys.
- 10 6. From here voting can be carried out by spending a UTXO locked to the combined public key.

This protocol allows members to vote without the risk of the one party having the ability to sign for any voting address.

15

8. FURTHER REMARKS

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.

20

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, 25 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 30 104 as described above.

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

In other embodiments of the invention, the blockchain network 106 may not be the bitcoin network. In these embodiments, it is not excluded that a node may perform at least one or some but not all of the functions of creating, publishing, propagating and storing blocks 151 of the blockchain 150. For instance, on those other blockchain networks a “node” may be used to refer to a network entity that is configured to create and publish blocks 151 but not store and/or propagate those blocks 151 to other nodes.

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

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 for enabling pseudorandom selection of one or more data items between a group of two or more parties, each party having a respective index, and wherein the method is performed by a first party of the group and comprises:

generating an ordered set of first party shuffle keys, wherein each first party 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 first party shuffle key

based on a respective first private key, or generating the respective first party shuffle key and deriving the respective first private key based on the respective first party shuffle key,

for each respective data item, deriving, based on the respective first private key, a respective first party public key, and mapping the respective first party public key to the
5 respective data item;

obtaining, from each other respective party, a respective party public key mapped to each respective data item, wherein a respective combined public key is generated based on each respective party public key mapped to that respective data item, the respective combined public key also being mapped to that respective data item;

10 obtaining, from each other respective party, a respective ordered set of respective party encrypted shuffle keys, each respective party encrypted shuffle key being generated by encrypting a respective party shuffle key with a respective scalar value;

for each respective data item, generating a respective sequence of elements by placing the respective first party shuffle key corresponding to that respective data item at a
15 first position in the respective sequence and placing each respective party encrypted shuffle key corresponding to that respective data item at a respective position in the respective sequence based on the respective index of the respective party, wherein the respective sequences of elements are placed in an initial ordered set;

encrypting each respective sequence of elements with a first scalar value to generate
20 a respective first encrypted sequence;

generating a first ordered set of respective first encrypted sequences of elements, wherein the order of the respective first encrypted sequences of elements differs compared to the initial ordered set of respective sequences of elements; and

sending the first ordered set of respective first encrypted sequences of elements to
25 at least the second party.

The second party is configured to decrypt each element using an inverse value of the respective scalar used by the second party to generate the respective (second) party encrypted shuffle keys, then encrypt each respective sequence of resulting elements with a
30 different respective scalar value; then shuffle the order of the resulting elements to produce a second ordered set of respective second encrypted sequences of elements. The second

party shares the second ordered set of respective second encrypted sequences 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
5 key comprises selecting a respective x-coordinate of the respective first party shuffle key as the respective private key.

Statement 3. The method of statement 1, wherein said deriving of the respective private
10 key comprises deriving the respective private key based on a hash of the respective first party 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 first party shuffle keys in a first
blockchain transaction, and sending the first blockchain transaction to one or more nodes of
15 a blockchain network;
b) storing a commitment of each respective first encrypted sequence of elements in
a second blockchain transaction, and sending the second blockchain transaction to one or
more nodes of a blockchain network;
c) storing a commitment of the first ordered set of respective first encrypted
20 sequences of elements 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 4, wherein the first, second, and third blockchain
25 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:
30 sharing the mapping of each respective first party public key to the respective data
item with the group of parties.

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

storing a commitment of the mapping of each respective first party 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.

5

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

generating the respective combined public key for each data item.

Statement 9. The method of any preceding statement, comprising:

10 sending each of the respective combined public keys to one or more of the group of parties.

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

15 obtaining a final ordered set of respective final encrypted sequences of elements, wherein each final encrypted sequence of elements is generated by each party encrypting each respective sequence of elements with a different respective scalar value, with each different party using a different respective scalar value for each sequence of elements.

Statement 11. The method of statement 10, comprising:

20 committing to a selection of one of the final encrypted sequences of elements;
for each other party, obtaining a respective inverse value of the respective scalar value used by that party to encrypt the selected final encrypted sequence of elements;
using the obtained respective inverse values and a respective inverse value used by the first party to encrypt the selected final encrypted sequence of elements, decrypting the
25 selected final encrypted sequence of elements to obtain a selected sequence of elements, each element being a respective shuffle key;
for each respective shuffle key in the selected sequence, deriving a respective selected private key and therefrom deriving a respective selected public key;
deriving a selected combined public key from the each of the respective selected
30 public keys; and
determining a selected data item based on the mapping of the respective combined public keys to the respective data items.

Statement 12. The method of statement 11, wherein said committing to the selection of one of the final encrypted sequences of elements comprises:

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

Statement 13. The method of any preceding statement, wherein each respective combined public key is used to lock a respective unspent transaction output.

10

Statement 14. The method of statement 13, comprising:

locking one or more of the respective unspent transaction outputs to the respective combined public key.

15 Statement 15. The method of statement 13 or statement 14 when dependent on any of statement 11 or statement 12, comprising:

deriving a combined private key from each of the respective selected private keys;
and

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

Statement 16. The method of statement 15, wherein a sixth blockchain transaction is used to unlock the respective unspent transaction output locked to the selected combined public key, and wherein the method comprises:

25 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.

30

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.

Statement 17. The method of any preceding statement, wherein each respective data item is a respective token.

- 5 Statement 18. The method of statement 17, wherein each respective token represents a respective vote.

- Statement 19. The method of statement 13 or any statement dependent thereon, wherein each respective unspent transaction output is locked to a public key of an entity other than
10 a party of the group, e.g. an oracle or other trusted third party.

Statement 20. The method of any preceding statement, comprising:

- generating an attestation transaction, wherein the attestation transaction comprises a locking script configured to, when executed alongside an unlocking script of a response
15 transaction comprises, for each respective data item, an input set comprising a respective candidate x-coordinate, respective candidate y-coordinate of a corresponding respective candidate shuffle key, and a respective candidate scalar value:

for each input set:

- verify that the respective candidate x-coordinate derives to one of the first
20 party public keys,
verify that the respective candidate x-coordinate and the respective candidate y-coordinate derive to one of the respective first party shuffle keys; and
verify that the derived one of the respective first party shuffle keys and the respective candidate scalar value derive to one of the respective first party encrypted
25 shuffle keys.

Statement 21. Computer equipment comprising:

- memory comprising one or more memory units; and
processing apparatus comprising one or more processing units, wherein the memory
30 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 configured so as, when run on one or more processors, to perform the method of any of statements 1 to 20.

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

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, each party having a respective index, and wherein the method is performed by a first party of the group and comprises:

generating an ordered set of first party shuffle keys, wherein each first party 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 first party shuffle key based on a respective first private key, or generating the respective first party shuffle key and deriving the respective first private key based on the respective first party shuffle key,

for each respective data item, deriving, based on the respective first private key, a respective first party public key, and mapping the respective first party public key to the respective data item;

obtaining, from each other respective party, a respective party public key mapped to each respective data item, wherein a respective combined public key is generated based on each respective party public key mapped to that respective data item, the respective combined public key also being mapped to that respective data item;

obtaining, from each other respective party, a respective ordered set of respective party encrypted shuffle keys, each respective party encrypted shuffle key being generated by encrypting a respective party shuffle key with a respective scalar value;

for each respective data item, generating a respective sequence of elements by placing the respective first party shuffle key corresponding to that respective data item at a first position in the respective sequence and placing each respective party encrypted shuffle key corresponding to that respective data item at a respective position in the respective sequence based on the respective index of the respective party, wherein the respective sequences of elements are placed in an initial ordered set;

encrypting each respective sequence of elements with a first scalar value to generate a respective first encrypted sequence;

generating a first ordered set of respective first encrypted sequences of elements, wherein the order of the respective first encrypted sequences of elements differs compared to the initial ordered set of respective sequences of elements; and

sending the first ordered set of respective first encrypted sequences of elements to at least the second party.

2. The method of claim 1, wherein said deriving of the respective private key comprises selecting a respective x-coordinate of the respective first party 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 first party shuffle key.

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

a) storing a commitment of the ordered set of first party 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 respective first encrypted sequence of elements in a second blockchain transaction, and sending the second blockchain transaction to one or more nodes of a blockchain network;

c) storing a commitment of the first ordered set of respective first encrypted sequences of elements 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 first party 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 first party 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:
generating the respective combined public key for each data item.
9. The method of any preceding claim, comprising:
sending each of the respective combined public keys to one or more of the group of parties.
10. The method of any preceding claim, comprising:
obtaining a final ordered set of respective final encrypted sequences of elements,
wherein each final encrypted sequence of elements is generated by each party encrypting
each respective sequence of elements with a different respective scalar value, with each
different party using a different respective scalar value for each sequence of elements.
11. The method of claim 10, comprising:
committing to a selection of one of the final encrypted sequences of elements;
for each other party, obtaining a respective inverse value of the respective scalar
value used by that party to encrypt the selected final encrypted sequence of elements;
using the obtained respective inverse values and a respective inverse value used by
the first party to encrypt the selected final encrypted sequence of elements, decrypting the
selected final encrypted sequence of elements to obtain a selected sequence of elements,
each element being a respective shuffle key;
for each respective shuffle key in the selected sequence, deriving a respective
selected private key and therefrom deriving a respective selected public key;
deriving a selected combined public key from the each of the respective selected
public keys; and
determining a selected data item based on the mapping of the respective combined
public keys to the respective data items.
12. The method of claim 11, wherein said committing to the selection of one of the final
encrypted sequences of elements comprises:

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

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

14. The method of claim 13, comprising:
locking one or more of the respective unspent transaction outputs to the respective combined public key.

15. The method of claim 13 or claim 14 when dependent on any of claim 11 or claim 12, comprising:

deriving a combined private key from each of the respective selected private keys;
and
using the combined private key to unlock the respective unspent transaction output locked to the selected combined public key.

16. The method of claim 15, wherein a sixth blockchain transaction is used to unlock the respective unspent transaction output locked to the selected combined 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.

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

18. The method of claim 17, wherein each respective token represents a respective vote.

19. The method of claim 13 or any claim dependent thereon, wherein each respective unspent transaction output is locked to a public key of an entity other than a party of the group.
20. The method of any preceding claim, comprising:
generating an attestation transaction, wherein the attestation transaction comprises a locking script configured to, when executed alongside an unlocking script of a response transaction comprises, for each respective data item, an input set comprising a respective candidate x-coordinate, respective candidate y-coordinate of a corresponding respective candidate shuffle key, and a respective candidate scalar value:
for each input set:
verify that the respective candidate x-coordinate derives to one of the first party public keys,
verify that the respective candidate x-coordinate and the respective candidate y-coordinate derive to one of the respective first party shuffle keys; and
verify that the derived one of the respective first party shuffle keys and the respective candidate scalar value derive to one of the respective first party encrypted shuffle keys.
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.