




End-to-End Voting with Non-Permissioned and Permissioned Ledgers

Stefano Bistarelli · Ivan Mercanti ·
Paolo Santancini · Francesco Santini 

Received: 3 September 2018 / Accepted: 26 February 2019 / Published online: 20 March 2019
© Springer Nature B.V. 2019

Abstract We propose a decentralised *end-to-end* voting platform (from voter to candidate) based on the block-chain technology. In particular, we study and exploit both the non-permissioned ledger of Bitcoin, and the MultiChain permissioned ledger. We describe the main architectural choices behind the two implementations, including the pre-voting and post-voting phases. Similar approaches are not as decentralised as our application, where it is possible to directly cast a vote to the block-chain, without any intermediate level. Benefits and drawbacks of each implementation are explained. The Bitcoin block-chain consists in a large number of already available nodes in the related peer-to-peer network; moreover, its reliability and resistance to attacks are also well established. With *MultiChain* we instead exploit a fine-grained permission system: MultiChain is a *permissioned* public ledger. Hence, with it we can also satisfy two more properties of end-to-end voting systems: *uncoercibility and receipt-freeness* and *data confidentiality and neutrality*. Moreover, we can avoid costs and price fluctuations related to Bitcoin.

Keywords E-voting · Distributed ledger · Permissioned block-chain · Bitcoin · Coloured coin · MultiChain

1 Introduction and Motivations

In short, *electronic voting* [24] (also known as *e-voting*) is voting by using electronic systems to aid casting and counting votes. These systems often employ cryptographic methods to craft receipts that allow voters to verify that their votes have not been modified. We propose an *end-to-end (E2E) verifiable* system [15] by exploiting different block-chain technologies [37, 46, 47]. In particular, we study and implement a voting system whose engine is a *non-permissioned* ledger (Bitcoin [4, 37] in our case),¹ and one based on a *permissioned* [2] one (*MultiChain* [23], see Section 6). A permissioned ledger is a ledger where individuals need permission to access a ledger. Among all other possible protocols (other examples are *Ethereum*,² or the HyperLedger family of block-chains,³ MultiChain was selected in this study because of its similarity with respect to Bitcoin: it is directly derived from the *Bitcoin Core* software at the heart of Bitcoin (see Section 6). The goal in this paper is to show Bitcoin-related implementations of an end-to-end voting scheme.

In general, some advantages of e-voting versus paper-based voting come directly from managing all the information automatically: the main benefits are

S. Bistarelli · I. Mercanti · P. Santancini · F. Santini (✉)
Department of Mathematics and Computer Science,
University of Perugia, Perugia, Italy
e-mail: francesco.santini@unipg.it

¹In the paper we use “Bitcoin” to refer to the network or protocol, and “bitcoin” to the currency.

²Ethereum: <https://www.ethereum.org>.

³HyperLedger: <https://www.hyperledger.org>.

represented by reduced costs, automated tallying, means for an immediate comprehensive reporting, and a direct archiving of results. For the elector, two benefits are the improved ability to correct mistakes (before the final submission) and, the possibility of increasing the ease of voting. In the following we will consider the term “election” with its broad meaning, not necessarily restricting it to political elections.

In both the proposed solutions, electors anonymously authenticate online (multiple schemes can be used, e.g., *Anonymous Kerberos* [45, 49] or *Blind signature* [14]) and, as a consequence, receive a token to vote (i.e., a fraction of an “asset”) in their *wallet*. Afterwards, a voter can “spend” such a token by transferring it to the address of the desired candidate with a transaction from its wallet. This is recorded, once for all, in the block-chain, which consists in a distributed-ledger of sequential transactions. Such a procedure avoids the need to have a centralised database managed by a trusted third-party. Finally, the result can be verified by quickly counting in the block-chain the tokens transferred to each candidate. An implementation through a block-chain automatically inherits *transparency, decentralisation and immutability* from this technology.

In Section 4 we implement an E2E voting platform by using the Bitcoin block-chain [37]. In this case, the evident advantage consists in *i)* the immediate availability of the underlying infrastructure (the Bitcoin peer-to-peer network), *ii)* the reliability of Bitcoin, with a large number of full nodes deployed worldwide,⁴ and *iii)* the long-term history of Bitcoin (since 2009), proving its security (see Section 5). We also provide an estimation of costs when using Bitcoin and considering different numbers of electors.

In Section 6 we propose a solution where we enforce two properties, desirable in e-voting platforms (see Section 7): *i) uncoercibility and receipt-freeness* (voters should not be able to prove how they voted), and *ii) data confidentiality and neutrality* (votes must be protected from external reading during the voting process). Differently from Section 4, where such two properties are not satisfied, we adopt a different platform for building *permissioned* block-chains, i.e., *MultiChain*. Thanks to MultiChain permissions management, it is possible to prevent anyone from

reading votes in the block-chain, and grant instead this right to specific users at a given time. In addition to the above-mentioned properties, a further benefit in implementing a voting application by using MultiChain is that we become independent from bitcoin price and fees. In fact, in the MultiChain implementation, the voting token comes at null price, and fees are not required at all.

This paper extends [9], where only some results on Bitcoin are presented; in the following we will discuss about both Bitcoin and MultiChain implementations. Section 2 introduces the background on Bitcoin. Section 3 represents the core of the paper, adapting the decentralised payment-system to an e-voting system. Section 4 describes an implementation using the Bitcoin block-chain and an estimation of costs by considering different numbers of electors. Section 5 reports some known attacks to Bitcoin, and if/how they can be perpetrated also for the voting platform. Section 6 describes an implementation using a permissioned ledger, i.e., *MultiChain* with the purpose to satisfy more properties of voting systems, and be detached from Bitcoin costs. Section 7 reports which properties can be satisfied by the two proposed implementations. Section 8 collects related work, while Section 9 finally wraps up the paper with conclusions and future work.

2 Bitcoin

The white-paper on Bitcoin appeared in November 2008 [37], written by a computer programmer using the pseudonym “Satoshi Nakamoto”. His invention is an open-source, peer-to-peer digital currency (being fully electronic, with no physical manifestation). Money transactions do not require a third-party intermediary, with no traditional financial-institution involved in transactions. Therefore, the Bitcoin network is completely decentralised, with all the parts of transactions performed by the users of the system.

The buyer and seller directly interact (peer-to-peer), but without using their real identities, and no personal information is transferred from one to the other. However, unlike a fully anonymous transaction, there is a transaction record. A complete transaction record of every bitcoin and every Bitcoin user’s encrypted identity is maintained on a public ledger, called the *block-chain*. For this reason, Bitcoin transactions are

⁴A constantly updated map with full nodes in Bitcoin can be found at: <https://bitnodes.earn.com>.

thought to be *pseudonymous*, not anonymous: Bitcoin addresses are pseudonyms of real individuals (one can have several pseudonyms).

Bitcoin [4] is a currency with a finite supply: a cap of (slightly less than) 21 million bitcoins is set by default. Hence, Satoshi designed Bitcoin to eventually become a deflationary currency. The only way to create new bitcoins is through the *mining* process: *miners* are the nodes that verify the transactions and add them to the block-chain. The number of bitcoins created each time a miner discovers a new block represents a reward for its job, which consists of the computation of the *proof-of-work* (more details in the following of this section). The reward (12.5 bitcoins at the time of writing) is designed to be halved every 210,000 blocks, approximately four years.

Figure 1 shows the main actors in the Bitcoin network. Generic *users* own a *wallet* associated with couples of private/public cryptographic keys. In Bitcoin, a private key is a 256 bit random number, and by using the *Elliptic Curve Digital Signature Algorithm (ECDSA)* [27], a 512 bit public key can be obtained from it. Afterwards, from the public key it is possible to obtain a Bitcoin *address*, e.g., applying an hashing function on it. Users use these keys to sign the transactions they generate in order to transfer their money to other users; transactions are then broadcast to the Bitcoin peer-to-peer network. The miners update the block-chain, a public distributed data-structure that implements the database of every transaction ever executed.⁵

In the following of this section we detail transactions and how they are aggregated into the block-chain.

Transactions Transactions are the basic brick of Bitcoin: they represent the mechanism that allows a user to cede money to another user, e.g., from a buyer to a seller in Fig. 1. The new owner can prepare a new transaction referring to the ones through which she received money, called the (multiple) *inputs* of this new transaction. The *output* of a transaction describes the destination of bitcoins instead. There can be multiple *outputs*, allowing a owner to make multiple payments at once; one output often represents the change w.r.t. a previous transaction.

⁵Genesis block (block number #0) created on 3 January 2009, first transaction on 12 January 2009, from Satoshi Nakamoto to Hal Finney, a developer and cryptographic activist (block #170).

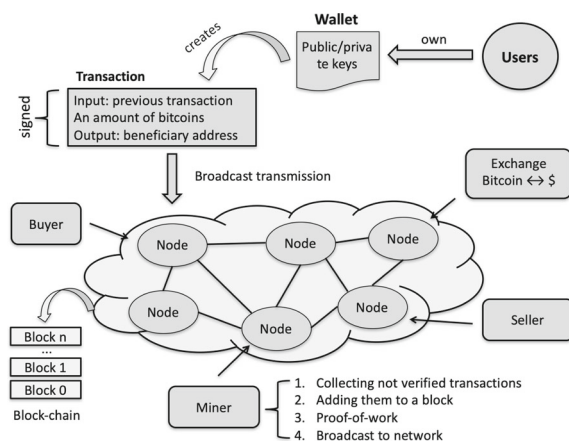


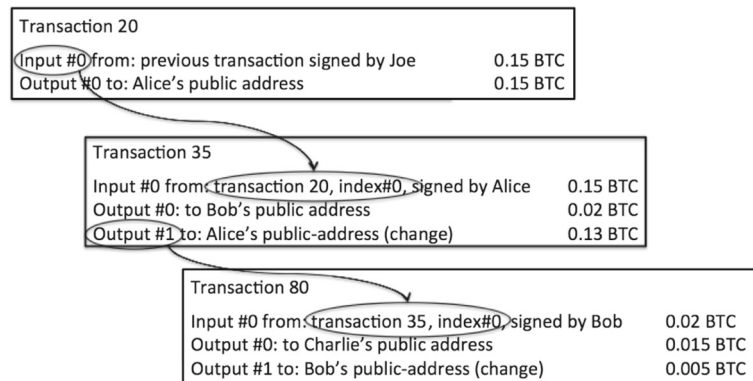
Fig. 1 The Bitcoin peer-to-peer network in a glimpse. Nodes can be buyers and sellers, exchange services that change bitcoins to other currencies and viceversa, and miners

The bitcoin transactions language *Script* is a Forth-like [41] stack-based execution language. Script requires minimal processing and it is intentionally not Turing-complete (no loops) to lighten and secure the verification process of transactions. An interpreter executes a script by processing each item from left to right in the script. Script is a stack-based language: data is pushed onto the stack, as well as operations, which can push or pop one or more parameters onto/from the execution stack, operate on them and possibly push their result onto the stack.

For example, the operator `OP_ADD` pops two items from the stack, add them, and finally push the resulting sum onto the stack [4]. There are also conditional operators as `OP_EQUAL`: it pops two items from the stack and pushes `TRUE` (represented by number 1) if operands are equal, or `FALSE` (represented by 0) if they are not equal. In Bitcoin, transaction scripts usually contain a final conditional operator, so that they can produce the result `TRUE`, which points to a valid transaction.

An input must store the proof it belongs to whom wants to reuse the money received in a previous transaction. We refer to the chain of transactions in Fig. 2 to explain this. Alice wants to send 0.02 bitcoins to Bob, thus she refers to transaction with id 20, where she has received 1 bitcoin from Joe; she prepares the transaction to Bob by adding the public key of Bob and then signing the whole transaction with her (Alice’s) private key. Hence, 0.02 bitcoins are irreversibly linked to Bob’s public-key, who is the only one possessing

Fig. 2 An example of a chain of three transactions between Joe, Alice, Bob, and Charlie



the corresponding private-key, as Alice is the only one who can reuse 1 bitcoins received from Joe. Transaction 35 in Fig. 2 has two outputs, one is the change ($0.15 - 0.02 = 0.13$ bitcoins).

To go more into the details, an example of transaction in a human-readable format is provided in Fig. 3. Line 1 shows the hash of the transaction, which uniquely identifies it. Line 2 shows the Bitcoin protocol version. Line 3 reports the time at which a particular transaction can be added to the blockchain. If it is less than 500 million it is interpreted as a block-height (miners wait until that block-height has been reached before adding it to a block), while if it is above 500 million it is converted to a Unix time-stamp. From line 4 to line 14 we find the (single, in this case) input of a transaction: in particular

line 5 refer to the previous transaction (the hash of it) from where money comes from, and line 6 specifies the reference output (i.e., 0) in that transaction. Lines 7-11 report the signature of the sender followed by her public-key; together can be used to check if she is the real owner of these bitcoins. Line 13 shows the address of the input transaction, while line 14 shows the amount of sent bitcoin (1 bitcoin). Lines 15-26 in Fig. 3 show the two outputs of this transaction. The second output is what transferred by this transaction, i.e., 0.67407245 bitcoins, while the first output represents the change between the input transaction and what paid, that is $1 - 0.67407245 = 0.32542755$ bitcoins. Line 18 and line 23 are statements in the Bitcoin scripting language. They define the conditions under which money can be unlocked; for instance,

```

1      {txid:"21ca709e800c216d73bc53b07e8a6ec8bdf3dfc6673400b4eb32c37fe50490e7",
2      version: 1,
3      locktime: 0,
4      vin:
5      [{"txid": "b18deaf7be159d3fac32d1d02b3bb8e9a906ca07ee06e26cf1bee86786caf184",
6      vout: 0,
7      scriptSig:
8      {hex:"47304402204d4734f8ede9c6b4c41cd38909057d79b652ae264717e90fd2468ae8f1c
9      6aae202200e24d93b2d9a9c8d48264202fe864df7b3b4094c266e5152fc09fe337fd550e901
10     4104ffc83a1835d9780d591c1ad98128b14afda69e0921ccfd139365f938b59564aea68c3dc
11     2fd2b2e989482ed9f1fa397a077b5f6c61abb05d6b97cfc9b4cd4fa2e"},
12     n: 0,
13     addr: "1NvKhobLdzd4Cxp7UVa9Tw9nj5iRA4XuES",
14     value: 1}],
15     vout:
16     [{"value": "0.32542755",
17     n: 0,
18     scriptPubKey: {asm: "OP_DUP OP_HASH160 f0704ce2b364c12c4a82faa428f0c16db33c150f OP_EQUALVERIFY
19     OP_CHECKSIG",
20     type: "pubkeyhash",
21     addresses: ["1NvKhobLdzd4Cxp7UVa9Tw9nj5iRA4XuES"]}],
22     {value: "0.67407245",
23     n: 1,
24     scriptPubKey: {asm: "OP_DUP OP_HASH160 8655298c2fac774d85084e67e9b3c7ea7473bd22 OP_EQUALVERIFY
25     OP_CHECKSIG",
26     type: "pubkeyhash",
27     addresses: ["1DFHZtM7mbicj442xC9G7HrSUDZWbUubzV"]}]}
28     fees: 0.0005}

```

Fig. 3 A real example of a Bitcoin transaction

8655298c2fac774d85084e67e9b3c7ea7473bd22 (line 23) is the destination address, while the other codes impose to present a signature created with the private key corresponding to the public-key of the receiver (i.e., OP_EQUALVERIFY). Line 26 reports the fee paid to miners (0.0005 bitcoins).

Note that OP_RETURN is a script opcode used to mark a transaction output as invalid. Since the data after OP_RETURN are irrelevant to Bitcoin payments, arbitrary data can be added into the output after an OP_RETURN, as a text message.

Finally Alice broadcasts her transaction to the Bitcoin network, and miners are in charge of adding it to the block-chain.

Block-Chain Miners keep the block-chain consistent, complete, and unalterable: they repeatedly verify and collect newly broadcast transactions into a new group of transactions, called a *block*. Mining is also the mechanism used to introduce bitcoins into the system (reward plus fees). This both serves the purpose of disseminating new coins in a decentralised manner as well as motivating people to provide security for the system.

The first step accomplished by a miner, after collecting transactions, is to perform a verification step on them. This implies to check a set of rules, e.g., if their format is syntactically correct w.r.t to the protocol, or to reject it if the sum of input values is less than sum of output values. Transactions are also checked w.r.t. the ones already in the block-chain: they are rejected in case they have been already registered there. A miner also relays transactions to the other peers in the network.

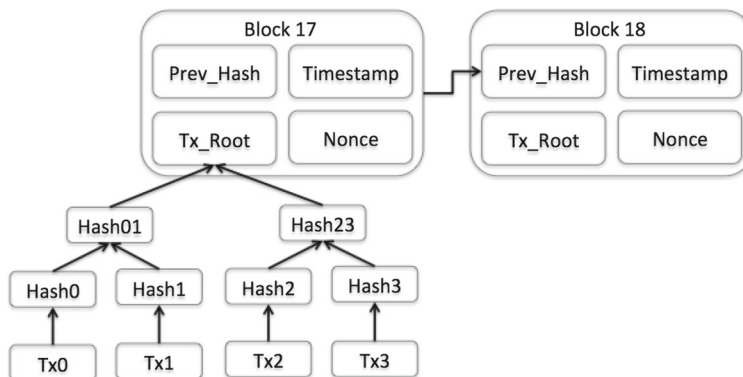
Valid transactions (all checks are passed) are added to a block. A block consists of a header and a list of

transactions (the block body). In Fig. 4 we show a fictitious excerpt of the block-chain (only two blocks), with a simplified structure (e.g., we show the main fields of the header). Inside a block, transactions are organised as a *Merkle tree* [34], which is a tree where every non-leaf node is labelled with the hash of the labels or values (in case of leaves) of its child nodes. Hash trees allow an efficient and secure verification of the contents of large data structures. In the header of a block, the Merkle root (*Tx_Root*) is the hash of all the hashes of all the transactions in the block. *Timestamp* in Fig. 4 is the current (Unix) time. Each block contains information that chains it to the previous block in the block-chain, that is a hash of the previous block (*Prev_Hash* in Fig. 4). Thank to this field, a block (and consequently the block-chain) is computationally impractical to be modified, since every block after it would also have to be regenerated. The remaining field of the header, i.e., *Nonce*, is obtained from the computation of the proof-of-work.

Simply speaking, with such a calculation a miner aims at finding a random *nonce* (a little random data) that becomes part of a block and makes it have a hash that starts with a given amount of zeroes. This nonce corresponds to a 32-bit field that, once inserted into the current block-header, makes its hash be less in value than the current *target*. The target is a 256-bit number (extremely large) that all Bitcoin nodes share. The most widely scheme for hashing is *SHA256* [21], similarly to Adam Back’s *Hashcash* [7].

This proof is easy to verify, but extremely time-consuming to generate: the demanded work is exponential in the number of “zero bits” required in the head of the block-hash, but it can be verified by other peers in a single hash. Such amount of effort is continuously adjusted: every 2016 blocks the target is

Fig. 4 An simplified model of the block-chain with only two blocks, one of which detailed with the structure of its Merkle tree



decreased by all the nodes, with the aim of keeping the average time between new blocks at 10 minutes.⁶ Additionally, the miner is awarded the transaction fees in a block.

2.1 Open Assets Protocol

In Section 3 (i.e., in the description of the voting phases) and Section 4 (i.e., in the implementation) we will use an evolution of the concept of *coloured coins*. This concept has been designed to be a logical layer on top of standard Bitcoin: in fact, it does not require any change to the existing Bitcoin protocol. The purpose is to create a new set of information about coins being exchanged: by using coloured coins, bitcoins can be “coloured” with specific attributes. This enhancement effectively turns coins into tokens, which can be then used to represent anything: coloured coins can represent, for instance, deeds for a house, stocks (which can be traded frictionlessly through the Bitcoin infrastructure), bonds or futures.

Such an extension of coloured coins is implemented in the *Open Assets Protocol*⁷ (OAP). It allows issuance and transfer of user-created assets: outputs can encapsulate a quantity of a user-defined asset on top of that Bitcoin amount. The transaction outputs have two main features:

- The ID is a RIPEMD-160 hash of the SHA-256 hash of the output script referenced by the first input of the transaction that initially issued that asset. It is used to uniquely identify the stored asset. An issuer can reissue more of an already existing asset, as long as the private key for that asset ID is retained.
- The quantity is an unsigned integer representing how many units of that asset are stored on the output.

Transactions relevant to OAP need to have a special output called *marker output*, which allows clients to recognise such transactions. OAP transactions can be used to issue new assets, or transfer ownership of assets. The marker output can have a zero or non-zero value, and it starts with the OP_RETURN opcode, and can be followed by any sequence of opcodes,

but it must contain a PUSHDATA opcode containing an OAP marker payload. The payload, as defined by OAP, has a format that includes, a marker (2 bytes), a version number (2 bytes), an asset quantity count (representing a number of items in the next file, 1-9 bytes), an asset quantity list (unsigned integers representing the asset quantity of every output in order, bytes variable), metadata length (1-9 bytes), and metadata, which consists of arbitrary metadata to be associated with a transaction (it can be empty).

Each output in the Block-chain can be either coloured or uncoloured: uncoloured outputs have no asset ID and no asset quantity (they are both undefined), while coloured ones have a strictly positive asset quantity, and a non-null asset ID.

By colouring bitcoin we can create special token dedicated to the election. This approach offers many desirable characteristics: for example, *i*) clients can identify coloured outputs simply by traversing the block-chain, *ii*) the cost of issuing or transferring an asset is completely independent from the quantity, and *iii*) the entire cryptographic infrastructure used by Bitcoin for securing the spending of outputs is reused for securing the ability to issue assets.

3 Voting with the Bitcoin Block-Chain

In this section we describe how to use the Bitcoin block-chain to implement an E2E voting-system. According to the high-level models of election systems [13], and to many proposals in the literature as the taxonomy in [43] recollects, e-voting has three phases with two/three sub-phases each:

- 1) Pre-voting Phase: (a) Candidate nomination and registration process, (b) Voter registration process.
- 2) Voting Phase: (a) Voter authentication, (b) Vote casting, (c) Vote transmission and confirmation.
- 3) Post-voting Phase: (a) Counting, (b) Result, (c) Audit administration.

Pre-Voting Phase Step 1(a) is the process of approving nominees as eligible candidates for certain positions in an election. A candidate in this context can be a named individual or a party. The aim is to retrieve a list of candidates that own a couple of asymmetric keys: the public one is associated with the identity

⁶The average current time between two consecutive blocks is 9’44” (July 2018).

⁷<https://github.com/OpenAssets/open-assets-protocol>.

of the candidate⁸ and it has to be freely available to all the voters, while the related private-key has to be kept secret by each candidate. Candidates' public-keys listed on (government) official Web-sites may suffer from phishing attacks (e.g., if the Web page is manipulated), driving the user to transfer bitcoins to a different wallet, instead of really voting. To prevent this, we require the user receives the list of verified public-keys at the same time she obtains the voting token, from the same token distribution-service used in phase 1(b). In this setting, we suppose candidates register themselves in person, by showing an ID and communicating their public key (only) to the authority. Otherwise, a digital registration can be implemented for candidates as well, similar to what proposed for voters in phase 1(b).

Step 1(b) concerns the process of approving voters instead of candidates. Due to its nature (e.g., the possibly large number of voters), such a step has to be fully digital. The public key of an approved voter will be charged with an amount of bitcoins, which represents the election token to be spent as a vote.⁹ The related private key will be instead used by the voter's wallet to cast a preference in the voting phase, by signing the transaction. Each voter generates her public-private keys, associated to her wallet.

However, a public key cannot be directly associated to the voter's identity, otherwise the anonymity property would be not guaranteed (see Section 7); clearly, anonymity is one of the main properties we need to satisfy (if not "the" property). In order to guarantee it we propose the Anonymous Kerberos authentication-protocol [45, 49]. Note that this is only one among different anonymous-authentication approaches that can be exploited to anonymise voters; in Remark 2 we propose other anonymous authentication schemes. In our implementation (Section 4) we have opted for a variant of the Anonymous Kerberos protocol, as explained in the following paragraph.

Anonymous Kerberos [45, 49] provides a mechanism for principals to authenticate to a remote service without disclosing their identity. We suppose the sequence is initiated by Alice (one of the voters), who logs

⁸A Bitcoin recipient *address* is the hash of the recipient's public-key.

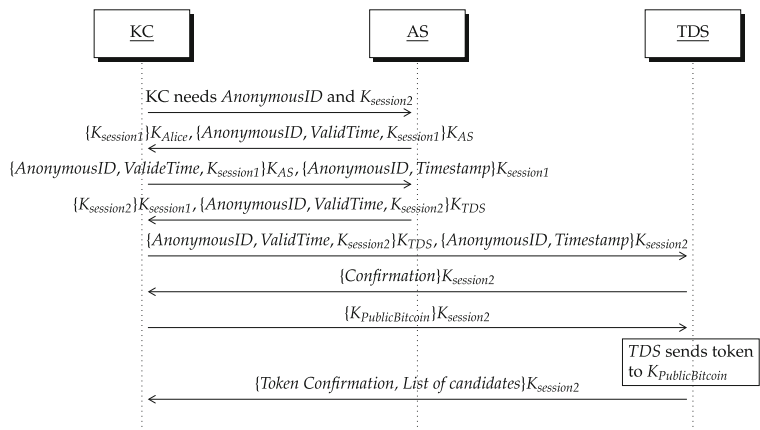
⁹In Section 4 we propose two different implementations of a token, based either directly on Bitcoin or the OAP (see Section 2.1), which lead to different election costs.

into the Kerberos Client (*KC*) with her *username* and *password* (or biometric features). The other two participants involved in the protocol are the *Authentication Server (AS)*, and the *Token Distribution Server (TDS)*. AS is in charge of authenticating Alice, while at the second step TDS transfers (via Bitcoin) the voting token to Alice by using her public key. In Fig. 5 we show the sequence diagram of the messages exchanged among these three entities. K_{Alice} , K_{AS} , and K_{TDS} in Fig. 5 are the secret keys of Alice, AS, and TDS respectively: see [45, 49] for a description about how they are created. In order to separate the authentication from a token (i.e., the "ballot"), it is important that AS releases an anonymous credential (i.e., *AnonymousID*) to Alice, used by her to access to TDS together with a session key ($K_{session2}$). In this way, AS will be aware of the identity of Alice without being able to associate it with her public key. On the other hand, TDS will not be aware of the identity of Alice. For the sake of duties separation, AS and TDS need to be implemented by distinct entities: for instance, a delegation of voters may (distributedly) implement AS, and a delegation of candidates may (distributedly) implement TDS. AS and TDS together prevent the same voter to register more than once: multiple requests with the same *AnonymousID* are denied by TDS.

In the last two messages, 7 and 8 in Fig. 5, first Alice sends (7) her public key ($K_{PublicBitcoin}$), which needs to be "charged" with a voting token. Message 8 confirms that such token has been accredited to the public key (sent by Alice), meaning that TDS has transferred one token to the wallet of Alice, and the related transaction is recorded in the block-chain.

Remark 1 (Multiple Kerberos) An obvious weakness of the protocol is that if the two entities AS and TDS collude, anonymity is immediately broken: AS can match the real identity with the anonymous one, and then match it with the $K_{PublicBitcoin}$ of Alice, with the help of TDS. Even if in our implementation (see Section 4) we opt for this simpler scheme, it is indeed possible to complicate it in order to have n entities (1 TDS and $n - 1$ ASs) and prevent this scenario if $n - 1$ (or less) entities collaborate to break anonymity. One solution is to ask Alice to authenticate $n - 1$ times to $n - 1$ different ASs: each of them returns a string of data, which Alice concatenates in the right order (AS may have a predefined sequence

Fig. 5 The sequence diagram of the Kerberos-based protocol: we have the Kerberos Client (KC), the Authentication Server (AS), and the Token Distribution Server (TDS)



order), and then hashes it (e.g., with SHA-1). The final string of 160 bits represents the *AnonymousID* that Alice presents to TDS in order to have its public key charged with one token. TDS is able to discover the real identity of Alice only if it somehow obtains all the sub-strings from each AS, and then, by hashing the whole string, TDS matches it with the *AnonymousID* given by Alice: TDS needs to collude with all the ASs, not with just one of them as in plain Anonymous Kerberos. TDS needs to store the hash of all the possible strings (however, in sequence) of the $n - 1$ ASs from the beginning.

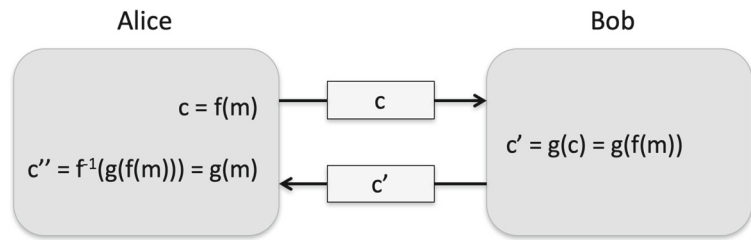
In order to avoid Alice to authenticate $n - 1$ times, the $n - 1$ ASs can be concatenated. In this way, Alice authenticates only to AS_1 , which then authenticates to AS_2 and so on, until AS_{n-1} is reached. Each AS encrypts its sub-string by using a public key associated with the real identity of Alice; then it passes the result to the next AS in the chain. Alice will decrypt the final message (obtained from AS_1) $n - 1$ times before applying the same hash function as before and send the result (i.e., the *AnonymousID*) to TDS. If only one AS is not maliciously collaborative, *AnonymousID* cannot be matched with the real identity of Alice.

Remark 2 (Other authentication schemes) Indeed other authentication schemes can be used to anonymously prove to be entitled to receive a voting token. For instance, *blind signature* [14] is a form of digital signature in which the content of a message is disguised (blinded) before it is signed: the sender is not aware of the message she signs. The resulting blind signature can be publicly verified against the original, unblinded message in the manner of a regular digital signature. Blind signatures are

typically employed in privacy-related protocols where the signer and message author are different parties. Typical examples include digital-cash schemes and cryptographic election-systems [29] (the cases in this paper). The Blind Signature scheme is visually represented Fig. 6: first Alice encrypts m , i.e., $f(m)$, thus obtaining a cyphered message $c = f(m)$. Alice sends c to Bob. Bob (blind-)signs c with a function $c' = g(c) = g(f(m))$: Bob does not know m . Bob sends back c' to Alice, who can remove her encryption (i.e., f^{-1}) and obtain $g(m)$. Blind signature schemes can be implemented using a number of common public key signing schemes, for instance *RSA* and *DSA* cryptographic schemes. Such a scheme can be used among a voter (Alice), AS, and TDS. After a first (non-anonymous) authentication between Alice and AS, Alice signs and sends her public key to AS, which then returns it to Alice after its turn of digital signing (c' in Fig. 6, i.e., AS is represented by Bob). Therefore, AS signs $K_{PublicBitcoin}$ without knowing it: AS only knows the identity of Alice, as with the Kerberos-based protocol. Now Alice can ask TDS to send her a token for voting. The credential is this time represented by $g(K_{PublicBitcoin})$: if $g^{-1}(K_{PublicBitcoin})$ corresponds to the public key of Alice, this means that AS has already authenticated her.

One more anonymous authentication step can be implemented by using *Zero Knowledge proofs* [22]. Zero-knowledge proofs are cryptographic protocols which do not disclose the information or secret itself during the protocol: knowledge possession is proved without revealing the information itself, or additional information. Clearly, this kind of interaction shows to be very useful to implement an anonymous authentication phase [31].

Fig. 6 A blind signature between Alice and Bob



Voting Phase After completing the pre-registration, a voter owns a voting-token in her wallet and she is ready to cast her preference. She transfers such token to the Bitcoin address of the candidate she likes. The sub-phase of authentication (2(a)) is performed by signing the transaction with the private key corresponding to the $K_{PublicBitcoin}$ charged in the pre-voting phase. Casting (2(b)) is performed by preparing a payment towards the public key (i.e., address) of the chosen candidate. Transmission (2(c)) corresponds to effectively executing the transaction, while confirmation (still 2(c)) can be done by simply self-checking if the transition is present in the block-chain (after the transaction has been mined).

Authentication, casting, and transmission are achieved by operating on the voter's wallet. When Alice wants to send bitcoins (2(b) casting) to Candidate, she uses her private key to sign a message with the input (the source transaction from TDS), an amount (the token), and an output (Candidate's address). The three-transaction chain is represented in Fig. 7. New transactions (votes) are broadcast to the Bitcoin network (2(c) transmission). Miners confirm all the transactions related to votes. All the transactions (i.e., all the votes) in a block are included in the block-chain (2(c) confirmation).

Post-Voting Phase This phase mainly covers tokens counting and result reporting. *Counting* (sub-phase 3(a)) is indeed the most interesting step. The possibility of recounting needs be considered as well, since it is one of the required properties (see Section 7): results need to be confirmed if requested.

So far we have referred to the amount of bitcoins transferred in the pre-voting and voting phases with the generic word “token” (i.e., a vote). In the simple case, a token corresponds to the smallest quantity possible of bitcoins that can be transferred in a transaction (one *satoshi*, i.e., 10^{-8} bitcoins), plus a *transaction*

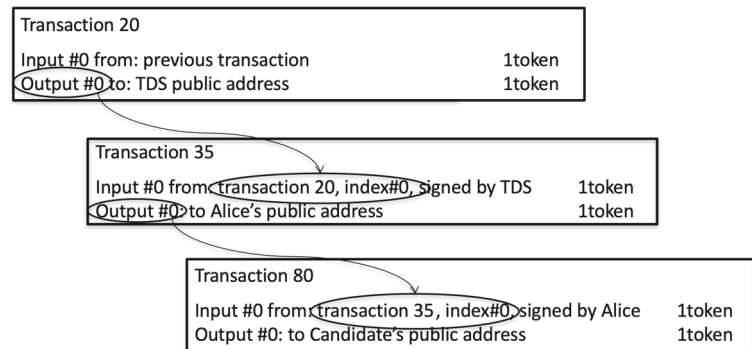
fee.¹⁰ An alternative is represented by the use of coloured coins, that is by using the OAP (see Section 2.1). In this way, it is possible to create assets on top of Bitcoin, in order to unambiguously mark money, or, here, to uniquely identify voting-tokens. In this case, the token amounts to a different quantity (6×10^{-6} bitcoins), while the fee is the same, in order of 10^{-4} bitcoins. We have implemented both these solutions, even if the implementation summarised in Section 4 describes the OAP alternative (more information and motivations behind using OAP can be found in Section 4).

Votes are counted by summing the tokens obtained by each candidate in the block-chain. As previously advanced, this can be implemented in two different ways: *i*) a token is a satoshi, or *ii*) a token is a digital asset coin (in case of using OAP). While the pre-voting and voting phases are marginally affected by choosing either *i* or *ii*, counting is more involved.

To be valid, a transaction needs to both originate from an authorised voter and end in the address of a registered candidate. In turn, a voter is authorised if she has previously received a token in a transaction from the public key of TDS (see the pre-voting phase). Therefore, for each token in the block-chain, the counting process needs to check if the sub-chain of transactions is identical to the last two transactions in Fig. 7. By remembering all the source addresses of confirmed votes we enforce that only one vote per authorised voter is counted. Since the block-chain can be accessed sequentially, this kind of search requires $n + 1$ scans of the block-chain (where n is the number of voters): one to count all the votes, while the other n scans to be sure that each vote has been cast by an

¹⁰Fees are required also for small transactions, in order for them to be processed without any delay. For transferring one satoshi, a fee of 10^{-4} bitcoins is considered as enough to avoid any delay, or risk to be discarded.

Fig. 7 The block-chain excerpt of a valid vote (cast by Alice to Candidate)



authorised voter. If we suppose that even the entity that is performing the count is aware of the list of authorised voters (which is known by TDS only), then the block-chain can be accessed only once.

The main reason behind using asset coins, i.e., *ii*, is that OAP is designed to attach metadata to a transaction and consequently expand Bitcoin functionalities. The token obtained in the pre-voting phase can be then unambiguously marked with the attached metadata signed by an administrator, that is the electoral commission. In this way, bitcoins really become votes, and it is possible to count single votes instead of authorised voters, as in *i* instead.

Therefore, there is not need to check if a voter is authorised or not and the count becomes easier: for instance, there is no need to keep track of voters' addresses to check they have voted only once. Without this check however, it is possible for a voter to send her asset coin to a different, authorised or unauthorised, voter, who can cast such a vote without being directly detectable. To avoid such misbehaviour, a possible solution is to use a *permissioned block-chain* [2] (see also Section 6), where access permissions are more tightly controlled, with rights to write (or even read) the block-chain state restricted to a few users. This leads to policies to arbitrarily censor transactions of such asset coins among voters, with the purpose to allow transactions only between an authorised voter and a candidate.

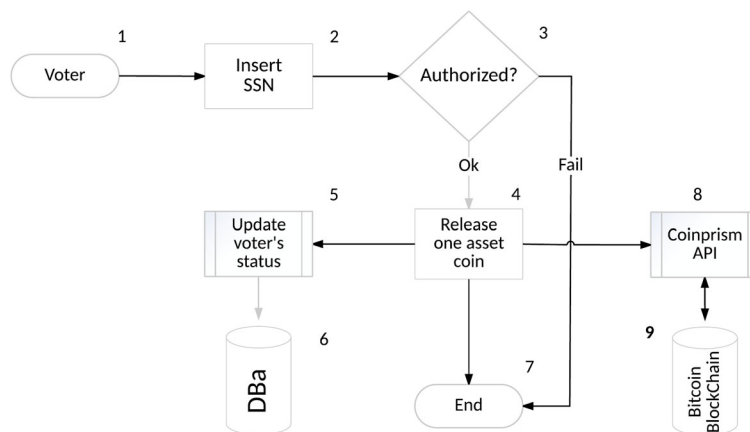
To count votes, not all the block-chain needs to be scanned: each block contains a Unix-time timestamp, so that it is possible to count as valid all the transactions in the blocks with a timestamp in the interval $Elections\ opening \leq timestamp \leq Election\ closing$. However, the mining process is not immediate: the interval between one block and another has an average

of 10 minutes, but not every block interval is exactly 10 minutes, and it follows a Poisson distribution. At the time of writing, in a 10 minute interval the probability of a block being mined is about 63%; in 30 minutes the probability raises to 95%.¹¹ For this reason and because *Election closing* is not synchronised with the mining process, we are sure to have counted all the votes arrived before the closing if we consider up to two blocks with $timestamp \geq Election\ closing$ in the block-chain.

In addition, as explained in Section 2, the last block of the election needs also to be confirmed (i.e., not orphaned due to a fork of the block-chain). Once the block-chain stores 8 blocks with $timestamp \geq Election\ closing$, counting can be launched: 2 blocks plus 6 blocks to avoid forks. The risk of losing a transaction to a reorganisation is low, and even then it will probably be re-included after the reorganisation occurs. However, it is better to wait for stable block-chain count only really confirmed transactions: note that counting votes can start as soon as there is a confirmed block with $timestamp \geq Election\ opening$. On the average, 80 minutes (10 minutes per block) after closing the election is necessary to start the counting. Note that block timestamps cannot be used in the practice. Due to synchronisation problems among the peers in the Bitcoin network, timestamps of mined sequential-blocks can be not ordered in time. However, it is possible to use their sequence number to understand how many of them have been mined after closing the election.

¹¹Data.bitcoinity.org: <https://tinyurl.com/yakxkvbb>.

Fig. 8 The workflow of the pre-voting stage



4 Implementation and Costs in the Bitcoin Block-Chain

The solution proposed in the previous section is architecturally distributed (Bitcoin is natively peer-to-peer), and no central authority is needed (TDS and AS can be distributed as well). In the following of this section, we describe how we have implemented voting with the OAP on top of the Bitcoin block-chain; moreover, we show how much an election costs by using either the bare block-chain, or OAP.

We have implemented a Web-interface to let a user vote without having a wallet. In this way, we let voters vote without having a wallet on their computer: only a browser is needed; however, a skilled voter can also opt for voting from her personal wallet. Therefore, more voters can vote through the same online application. First, we implemented the Authentication Server by using classical Anonymous Kerberos [45, 49]. In addition, some further technical requirements used during the development of our Web-based implementation are: *i*) a Web Server Apache, *ii*) a Mysql DBMS, *iii*) Perl CGI, and *v*) a digital-asset wallet compliant with OAP, as *Colu*¹² or *CoinPrism*.¹³ However, the same implementation can work with other OAP-compliant wallets, e.g., *CoinSpark*,¹⁴ or *SparkBit*.¹⁵

¹²<https://www.colu.com>.

¹³<https://en.bitcoin.it/wiki/Coinprism>.

¹⁴<http://coinspark.org>.

¹⁵<https://coinspark.org/sparkbit-wallet/>.

Figures 8, 9, and 10 respectively represent the workflow of the pre-voting, voting, and post-voting phases in our implementation using any coloured coin implementation. We authorise voters by checking their social security number (2 in Fig. 8). If the authorisation is successful, a voting token (i.e., an asset) is delivered to the address of the voter by using coloured coin API. Such a delivery is remembered by using an authentication database (6 in Fig. 8), in order to remember already authorised voters. In Fig. 9, a voter chooses a candidate from a database and, through the OAP-compliant Web-based wallet (“invisible to her”) cast her voting token to the selected candidate. After this stage, the voter receives a transaction ID, which represents a receipt to later check in the block-chain if her vote has been correctly assigned. Finally, in Fig. 10, a software scans the block-chain only for the coins coloured according to the considered election, and, through the same database of candidates (3 in Fig. 10), it counts the final result.

Costs After describing the implementation, in this paragraph we describe the cost of voting with either the bare block-chain of Bitcoin, or by using OAP.

To compute the cost of the election (in terms of bitcoins) by using OAP, we provide the following three calculations, where n represents the number of voters, and 10^{-4} is the magnitude order of the fee:

- $\alpha = 10^{-4} + (6 \times 10^{-6})$ represents the cost for transferring and issuing a new asset with CoinPrism. The cost of issuing or transferring an asset is completely independent from the quantity issued or transferred.

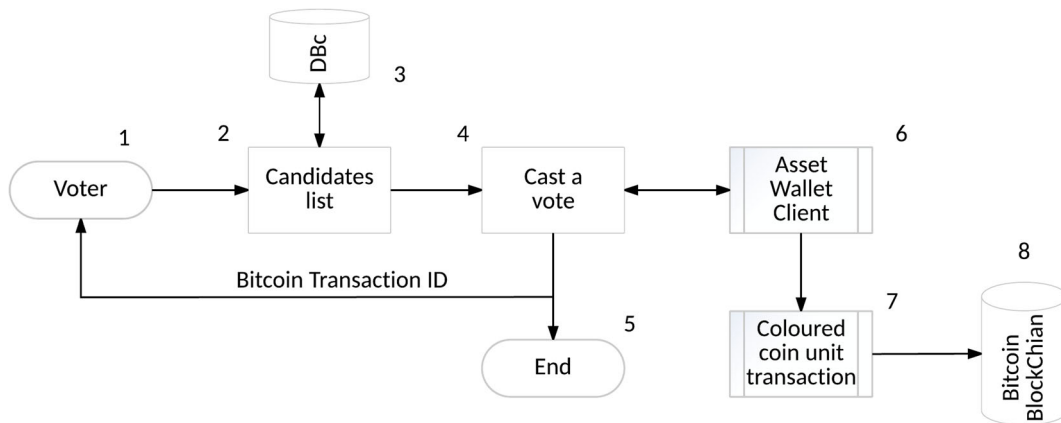


Fig. 9 The workflow of the voting stage

- b) $\beta = 2\alpha$, the cost to transfer the asset coin from TDS to the voter, and from a voter to a candidate.
 c) $T_{OAP} = \alpha + (\beta \times n) = \alpha \times (2n + 1)$, the total cost of the election.

Otherwise, without OAP, by voting with plain Bitcoin the cost becomes $T_{noOAP} = (2 \times 10^{-4} + 10^{-8}) \times n$, which is equal to the cost of a satoshi (i.e., 10^{-8} bitcoins, the token), two times the cost of the fee (i.e., 10^{-4} bitcoins, the cost to transfer the vote from TDS to a voter and the cost to vote for the voter), multiplied by n .

To exemplify the cost of an election, we consider as the current price 1 bitcoin = 10,000\$, and $n = 1000$ voters. In this case, by using the presented platform the total cost (for voting only) would have been around $T_{OAP} = 2,121\$$, and $T_{noOAP} = 2,000\$$ circa. To be even more precise, we can consider that all the used transactions in the voting platform have one input and one output, and thus their size is the following amount of bytes:¹⁶

$$\#inputs \times 180 + \#outputs \times 34 + 10 \pm \#inputs$$

If we consider 224 bytes, the amount of satoshis needed to be paid as fee for not having any delay in the block mining process is around 50,000 (i.e., 5×10^{-4}).¹⁷ Hence, the two costs for $n = 1000$ voters respectively become $T_{OAP} = 10,125\$$ and $T_{noOAP} = 10,000\$$.

¹⁶Public keys can be also compressed, for a total of 148 bytes instead of 180.

¹⁷Bitcoin fees: <https://bitcoinfees.earn.com>.

Therefore, the election cost with asset coins (using OAP and CoinPrism) is only marginally more expensive than simply voting with satoshis, and the benefits of having a clearly-marked token to represent a vote can improve the post-voting phase (see Section 3). Note a user has a very low incentive to spend the token otherwise instead of using it to vote: its cost is 10^{-4} bitcoins.

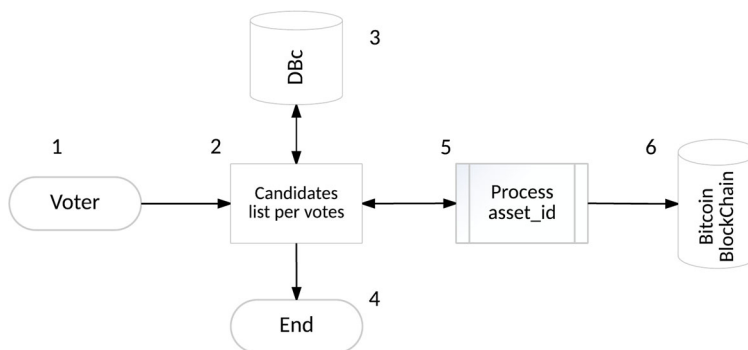
Note also that, given the price of a bitcoin fixed, overall the price is determined by the fees more than by the token (either a satoshi or an asset coin). Hence, while the same token can be reused for successive elections (the tokens are transferred to known candidates), the fee to transfer them has to be paid again at each election. To cut costs, fees can be reduced if waiting for even a few days from the time of the election end is negligible: however, going below a given threshold can result in waiting for more than 20 blocks to be mined before a vote appears in the block-chain, or also being completely dropped by some miners, thus not resulting in the final count.

In Table 1 we report the progression of one single election costs from a small (10 electors) to a large one (100,000 electors); of course not considering hardware and other expenses (i.e., fixed costs).

Finally, by studying the cost function we derive that the cost of an election is almost directly proportional to

- *i*) the amount of satoshis needed as fee;
- *ii*) the current price of a bitcoin (e.g., in dollars);
- *iii*) the number of electors;

Fig. 10 The workflow of the post-voting stage



with a very light preponderance in favour of *ii*. For instance, if both *i* and *ii* doubles, then the election globally cost quadruples.

Note that the fee to be paid usually follows the price of a bitcoin: when more transactions are requested, the price goes up, and during transaction peaks the fee increases, as explained in the following. Each node on the peer-to-peer network sets its own policies on fees. However, the most common implementation sorts pending transactions in reverse order of fee density d , where d is defined as

$$d = \frac{\text{fee paid for this transaction}}{\text{byte size of this transaction}}$$

Enough transactions are pulled from the memory pool to create a full block. When the memory pool grows too large, transactions are evicted starting from the bottom of the list. For this reason it is possible to reduce the fee costs by choosing off-peak periods, that is with a lower number of requested transactions; however, this is clearly not possible in case the date of an election has been already determined a long time earlier.

We now compare the obtained costs to general 2018 elections in Italy, in order to highlight the differences with real-world paper-based elections. With more than 30 million votes cast for the Parliament (i.e., the lower house),¹⁸ the total cost using Bitcoin would have been 300 million dollars, without considering any fixed cost due to the voting infrastructure. However, Italian elections in 2018 had a total cost of around 450 million dollars. Note that, in this example, we consider the price of a bitcoin to be 10,000 dollars. Due to the

¹⁸Less citizens have the right to vote for the higher house, and we can think of using just a single token for both the houses.

volatility of its price during the last years, clearly this comparison is deeply situational.¹⁹

5 Known Threats and Voting

We conclude these first sections using Bitcoin (or protocols on top of its block-chain) by describing possible threats to this technology and how they can impact on the security of voting through it. The aim of this section is to report some known attacks against the Bitcoin network, and understand if they can be applied also to the E2E voting-system proposed in this work.

As stated in [18] Bitcoin's network needs to be further investigated. Bitcoin purportedly offers three potential benefits to users: lower transaction costs, increased privacy, and no erosion of purchasing power due to inflation [18]. On the other hand, there are a number of factors that could discourage the widespread use of Bitcoin: first, not being a legal tender and being tied to a computer program could limit its widespread among less digitalised users. In addition, other impediments are its price volatility, and long-term deflationary bias: since the supply is limited in the long run, a widespread use would lead to a demand of Bitcoins that would likely outstrip supply, causing its price to steadily increase.

Clearly such negative issues do not interfere with an application to e-voting systems, since they are related to financial aspects. However, the last points raised in [18] is of our interest: Bitcoin's network security

¹⁹At the time of writing (January 2019), one bitcoin costs 3,600 dollars, and voting with Bitcoin would cost 107 million dollars instead.

Table 1 The costs in dollars of an election from $n = 10$ to $n = 100,000$ voters (not considering additional expenses, e.g., hardware)

	$n = 10$	$n = 100$	$n = 1,000$	$n = 10,000$	$n = 100,000$
T_{noOAP}	100\$	1,000\$	10,000\$	100,001\$	1,000,010\$
T_{OAP}	101\$	1,012\$	10,125\$	101,250\$	1,012,506\$

is uncertain.²⁰ Although counterfeiting purportedly is not possible, Bitcoin *i*) exchange services [17, 35] and *ii*) wallet services [16, 25, 48] have experienced some security breaches: so far, such two points of failure represent the main security issues we are going to better detail in the following of this section.

Exchange services allow customers to convert fiat money into bitcoins and vice versa. They have already suffered in the past from *Distributed Denial-of-Service* (DDoS) attacks on the servers dispensing the service [17]. The aim of attackers is to wait until the price of bitcoins reaches a certain value, then sell, destabilise the exchange, wait for everybody to panic-sell their bitcoins, wait for the price to drop to a certain amount, and finally stop the attack and start buying as much as they can. However, attacks to such services do not directly impact on the proposed e-voting application if implemented with Bitcoin; even less if implemented through a separate block-chain as proposed in Section 6.

Attacks on wallets (known attack *ii*) have been directed mostly to popular Internet wallet-providers so far, due to the chance of stealing larger amounts of money, instead of private wallets. These attacks (e.g., launched through compromising email accounts [25]) first access to the remote database, and then allegedly transfer money to fraudster Bitcoin-addresses. In an e-voting scenario this corresponds to penetrating the wallet of some candidate and fraudulently move votes to other candidates (or just users). Besides securing the machines where candidate wallets reside, this kind of misbehaviour can be detected by mining the block-chain, and related transactions can be invalidated in the counting process. In addition, also private wallets can experience software bugs (e.g., Android-based wallets [16]) that can be exploited in phishing or trojan attacks that force the beneficiary of a transaction be a different address. Indeed this kind of attacks could also affect our e-voting platform, even if implemented

on a separate block-chain. To reduce their impact, single nodes security needs to be improved.

Note that there is the possibility of attacks involving the misbehaviour of a pool of miners. There is the potential for a 51% attack, where a group of miners controlling over half of the network's computational power collude to rewrite a significant period of the block-chain's recent history. Some researchers claim that this is possible even with 33% of the network capacity [19]. The key idea behind this strategy, called *Selfish Mining*, is for a pool to keep its discovered blocks private, thereby intentionally forking the chain. The honest nodes continue to mine on the public chain, while the pool mines on its own private branch. If the pool discovers more blocks, it develops a longer lead on the public chain, and continues to keep these new blocks private. When the public branch approaches the pool's private branch in length, the selfish miners reveal blocks from their private chain to the public. This kind of -unlikely- attacks concern Bitcoin, while in a private block-chain as in Section 6 it is possible to limit this issue by setting a constraint on the number of blocks which may be created by the same miner within a given window. This enforces *mining diversity* directly in the configuration parameters of the block-chain. In general, the general premise behind the block-chain technology is that only a limited number of nodes in the network can be malicious.

Anonymity of Bitcoin transactions is questioned in several works, for example in [8, 28, 42]. Since the block-chain is public, this study investigates the possibility to associate Bitcoin addresses with external identifying-information (through a passive analysis). Different addresses can also be clustered together in a single wallet or user [3]: for instance, change addresses with input addresses. However, these threats are strongly mitigated in an e-voting application, since there is only one transaction per voter (one vote), and new keys can be generated before a new election. This holds for both the Bitcoin block-chain and private ones.

²⁰Of course, cash and traditional e-payment systems also have periodic security problems.

6 MultiChain

The advantage represented by voting with Bitcoin is that the infrastructure is already quite mature and tested in the large: currently there are more than 22 million wallet-users world-wide.

However, due to the considerable and well-known price fluctuations of bitcoin, the current and future cost of this crypto-currency (it is considered a “deflationary currency”), and the fees requested by miners to quickly validate transactions, using the Bitcoin block-chain can be expensive. In addition, the implementation presented in [9] was not able to satisfy two important properties, as *uncoercibility and receipt-freeness* and *data confidentiality and neutrality*. For these reasons, we propose a second implementation in order to overcome such limitations, and natively present a solution by using a different block-chain platform.

*MultiChain*²¹ [23] is a platform for the creation and deployment of private block-chains. Its original and main goal is to simplify the deployment of block-chain technology in the institutional financial sector, by providing more privacy and control. Like Bitcoin Core,²² from which it derives, MultiChain supports different operating systems as Windows, Linux and Mac. In addition, it provides a simple API and command-line interface.

MultiChain addresses the issues related to mining, privacy and openness via an integrated management of user permissions. The motivations are due to the following three aspects: first, *i*) to ensure that the block-chain activity is only visible to chosen participants, then *ii*) to introduce controls over which transactions are permitted, and finally *iii*) to avoid proof of work in the mining phase, still providing security and lowering associated costs.

Beyond controlling access to tokens, MultiChain enables any message to be signed by a user to prove that they own the private key corresponding to a particular address. MultiChain uses this property to restrict block-chain access to a list of permitted users. The “handshaking” process that occurs when two block-chain nodes connect is described by four steps:

- Each node presents its identity as a public address on the permitted list.

- Each node verifies that the other’s address is on its own version of the permitted list.
- Each node sends a challenge message to the other party.
- Each node sends back a signature of the challenge message, proving their ownership of the private key corresponding to the public address they presented.

If a MultiChain node is not satisfied with the results obtained during the previous steps, it aborts the peer-to-peer connection.

The connection of permissions to public addresses can be extended to many other operations on the network. An application, for example, consists in the right to send and/or receive transactions that can be restricted to a given list of addresses. Since transactions can have multiple senders and receivers, a transaction is only allowed if all of its senders and recipients are permitted on that list. Moreover, by adding a signature field to the coin-base transaction included by miners, also the MultiChain mining process can be similarly restricted.

Privileges can be granted and revoked by using network transactions containing special metadata. The miner of the first (i.e., “genesis”) block automatically receives all the possible privileges, including the administrator rights to manage the privileges of other users of the block-chain: in practice, it becomes the *Admin* of the current block-chain. This *Admin* grants privileges by using transactions whose outputs contain users’ addresses that receives such privileges, together with metadata denoting the granted privileges. The administrator can also grant the privilege of becoming *Admin* to other nodes: this avoids centralisation in the management of privileges. When changing the administration and mining privileges of other users, a minimum proportion of the existing *Admins* must vote to approve a change: an election needs to supports changes. These votes are registered by each *Admin* in a separate transaction, and the requested change is applied only once sufficient consensus is achieved.

In MultiChain, there are eight types of global permissions that can be granted on a per-address basis (addresses can either be public key hashes, or script hashes):

- *connect*: to connect to other nodes and see the block-chain contents.
- *send*: to send funds, i.e., inputs of transactions.

²¹MultiChain: <https://www.multichain.com>.

²²Bitcoin Core: <https://bitcoin.org/en/bitcoin-core/>.

- *receive*: to receive funds, i.e., outputs of transactions.
- *issue*: to issue assets, i.e., inputs of transactions which create new native assets.
- *create*: to create streams, i.e., inputs of transactions which create new streams.
- *mine*: to mine blocks, i.e., to sign the metadata of coinbase transactions.
- *activate*: to change *connect*, *send* and *receive* permissions for other users, i.e., sign transactions which change those permissions.
- *admin*: to change all permissions for other users, including *issue*, *mine*, *activate* and *admin*.

In general, permissions can be made temporary by limiting them in terms of a specific range of block numbers: in this way, they become available only to transactions which appear in this interval of blocks. The *Admin* can grant permissions to other addresses, including *admin* and *activate* permissions.

By restricting mining to a set of specific entities, MultiChain resolves the issue derived by private block-chains, in which only one participant can monopolise the process of mining. The proposed solution consists in constraining the number of blocks that can be created by the same miner, within a given interval in the block-chain. MultiChain implements this method by using a parameter called *mining diversity*, which takes value in the interval $[0..1]$. Thus, the validity of a block can be checked as described by the following steps:

- Apply all the permissions changes defined by transactions, following the order in a block.
- Count the number of permitted miners who are defined after applying those changes.
- Multiply miners by *mining diversity*, rounding the result up in order to get a *spacing* value.
- If the miner of the current block mined one of the previous *spacing* – 1 blocks, the block results as invalid.

This algorithm enforces a round-robin schedule, in which permitted miners create blocks in rotation. The mining diversity parameter also defines the proportion of permitted miners who would need to collude in order to undermine the network. A value of 1 ensures that every permitted miner is included in the rotation; a value of 0 represents no restriction at all. In general, higher values are safer, but a value too close

to 1 can cause the block-chain to freeze up if some miners become inactive for some reason: therefore, a reasonable compromise suggests this parameter to be assigned to 0.75.²³

Concerning security, the literature on MultiChain is clearly not as well developed as the one on Bitcoin (see Section 5). However, a major threat may consist in the presence of a (possibly) limited number of super-entities, i.e., the Admins, which also assign mining rights to other nodes: in the extreme case of a single Admin, by compromising its security the reliability of the overall network is disabled. Bitcoin can rely on a large number of miners with a global computational power of 40 million TH/s: achieving a 51% attack is very unlikely. Moreover, all these miners invested money in computational resources, and they are not interested in undermining the trust in Bitcoin. On the other hand, Admins in MultiChain are just “chosen” entities, hence the selection process needs to be carefully surveyed and it has to involve as more parties as possible, in order to increase the resilience of the network also with respect to DDoS attacks (see Section 5).

6.1 Implementation Details

In this section we describe the implementation of the voting process by using our implementation of a MultiChain block-chain.

In Fig. 11 we show the general architecture of the application,²⁴ which we simplify by only having one *Admin*. Clearly more Admins can be “created” by the first one, in order not to have a single administration point with full rights, and grant a fair and decentralised election process. To access MultiChain API in Fig. 11, the *multichain-cli* command-line tool or any other *JSON-RPC*²⁵ client needs to be used; MultiChain is compatible with any API library developed for Bitcoin Core. The proposed *Evote* application takes advantage of such an API.

The *Admin* node is in charge of setting the roles of the other participating nodes with respect of the e-voting block-chain. On this node, the following commands are executed:

²³<https://www.multichain.com/download/MultiChain-White-Paper.eps>.

²⁴<http://evote.dmi.unipg.it>.

²⁵JSON-RPC: <http://www.jsonrpc.org>.

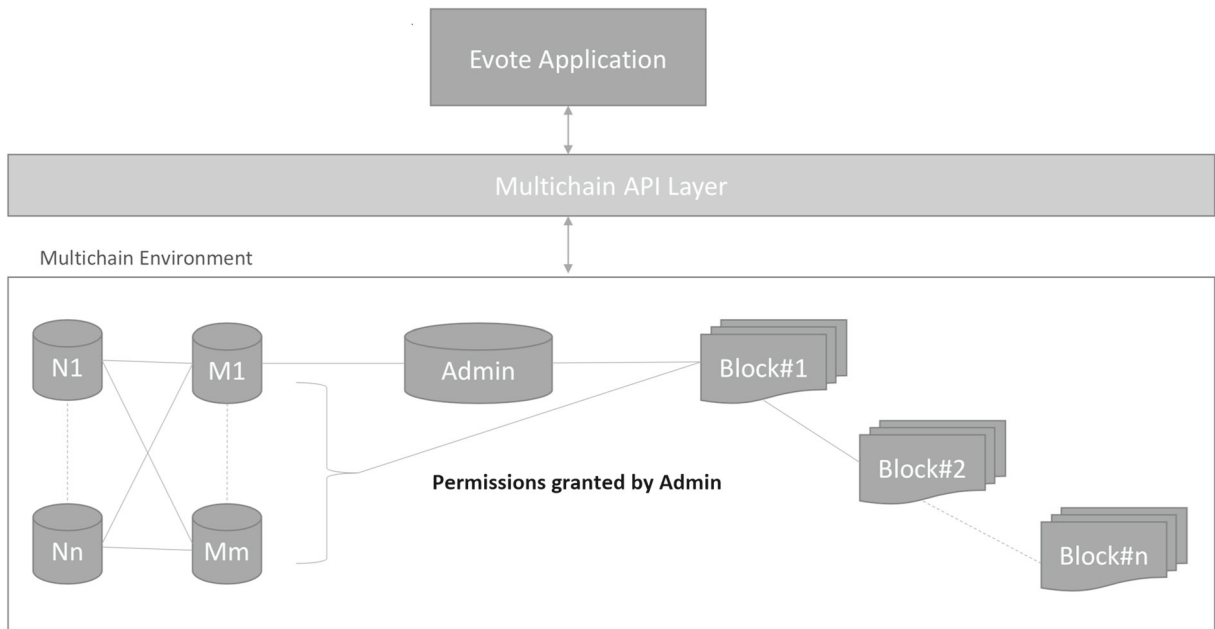


Fig. 11 The architecture of the MultiChain-based voting application

- `# multichain-util create evote;`
- `# multichainind evote -daemon;`
- `# multichain-util create aids;`
- `# multichainind aids -daemon.`

These commands are used to create and initialise (by creating their respective genesis block) two different block-chains: one to manage the voting process (i.e., *evote*), and the other to manage the *AnonymousID* of the users entitled to vote (i.e., *aids*).

Differently, in Section 3, we exploit the features of MultiChain to implement the authentication phase with a secondary block-chain, instead of Anonymous Kerberos or Blind Signature. Such a choice leads to a more compact implementation, avoiding to call additional external functionalities. To accomplish it, from *Admin* node we create a new *stream*. Streams provide a natural abstraction for block-chain use cases which focus on general data retrieval, time-stamping and archiving, rather than the transfer of assets between participants. Streams can be used to implement three different types of databases on a chain: *i*) a key-value database or document store, in the style of NoSQL, *ii*) a time series database, which focuses on the ordering of entries, and *iii*) an identity-driven database where entries are classified according to their author.

This new stream stores the *AnonymousID* related to users entitled to vote and a bit (true or false) of information showing whether the voting token has been already delivered to her or not: command `# create stream aidList false` is used for this, where “false” means that the stream can be managed only by its creator (i.e., *Admin*) or one of its delegates. A user presents its (real) id to an Authorisation Service (an AS, as it happens with Anonymous Kerberos), which checks if the user is entitled to vote or not; in case she is, with `# publish aidList aid 0`, the AS adds the *AnonymousID* aid to aidList, that is a voter with that *AnonymousID* is entitled to vote, but her token still needs to be delivered. In practice, this second block-chain is used to store the voting tokens added by an AS.

Once the corresponding aid is present in aidList (to check, command `# liststreamkeyitems stream aid` can be used), then a voter can receive the token to vote. For example, N1 in Fig. 11 (i.e., a TDS entity) is chosen by Admin to release tokens to voters; this can be accomplished by Admin by using commands:

- `# issue votes number_of_voters` (which creates all the tokens and terminated the pre-voting phase), and

- `# grant N1 send` (which grants N1 the right to transfer tokens to voters).

In the pre-voting phase, N1 uses the command `# sendwithdata '{"votes":1}' voter_asset_wallet_address` with the purpose to distribute a token to the wallet address of a voter. Similarly, during the voting-phase, each voter can use the command `# sendwithdata '{"votes":1}' candidate_asset_wallet_address` to vote the chosen candidate.

In the post-voting phase, in order to grant read access to a given *node*, from *Admin* it is necessary to use the command `# grant node receive`. Then, *node* can read the votes by using command `# listassets votes`. Of course this right can be granted and revoked by *Admin* at anytime.

It is worth to notice that the use of MultiChain does not lead to variable costs in supporting an election, as Bitcoin does instead. By developing an ad-hoc solution, only fixed-costs to set up machines need to be sustained (see Section 4), since tokens are inexpensive and fees are not considered. This choice could be preferred in case of general elections, where MultiChain Admins/miners may be represented by a mix of political parties, governmental and non-governmental organisations. However, less sensitive elections with a few hundred voters may take advantage of the already established infrastructure offered by Bitcoin.

7 Properties of E-Voting Systems

In the following we report some classical properties of e-voting systems [20, 36, 44], and we comment about to what extent they can be satisfied or not by the introduced E2E voting-systems.

Verifiability and Auditability: *It is possible to verify that all the votes have been correctly accounted for in the final tally, and there are reliable and demonstrably authentic election records.* Transactions in the block-chain implement such a public election record, which is public (Bitcoin implementation), or a transaction id can be received as receipt proving its inclusion (MultiChain implementation).

Uniqueness: *No voter is able to vote more than once.* Double-voting is prevented by the fact that double-spending is not possible with the block-chain technology [37, 38].

Integrity: *Votes should not be able to be modified, forged, or deleted without detection.* When a transaction is in a confirmed block, to modify that block is computationally hard by design [37, 38], since it is required to also modify all the successive blocks.

Vote Anonymity: *Neither election authorities nor anyone should be able to determine how any individual voted.* Public-keys of voters cannot be associated with their identity because the *Token Distribution Service (TDS)* is separate from the *Authorisation Service (AS)*, both in Sections 3 and 6, for instance by using a *Multiple Kerberos* service (see Section 3) or a different channel of the block-chain (see Section 6).

Counting and Recounting: *Voting system must provide easy functions for counting and recounting, in case there of any question about the final voting result.* Each valid transaction is permanently stored in the block-chain, where it is possible to repeat the counting phase when needed. Any node (Bitcoin), or authorised ones (MultiChain) can repeat this phase as needed. If we consider MultiChain, in order to secure counting and possible recounting, the rights on mining operations can be revoked by Admins at the end of elections, with the purpose to prevent block rewritings. In case of Bitcoin, the hash of the last block after terminating the elections represents an anti-tampering evidence of votes.

Eligibility and Authentication: *Only authorised voters are able to vote;* this is accomplished by the pre-voting phase (see Sections 3 and 6).

The last four properties are more influenced by the specific implementation, either Bitcoin or MultiChain.

Uncoercibility and Receipt-Freeness Voters should not be able to prove how they voted. With Bitcoin a voter can prove that she is the source of a transaction registered in the block-chain. Therefore, we can think of using this e-voting system in case the risk of coercibility is low. To prevent this, it is possible to adopt permissioned block-chains [2], where the right to read the block-chain can be granted only to some users. To mitigate this problem, we decided to use MultiChain. In such an implementation, each voter receives a receipt when her vote/transaction is correctly registered in the block-chain, in order to still

maintain the verifiability property. Only some official entities can be instead allowed to read the whole block-chain with the purpose to count votes. Such entities, for instance nodes managed by each political party, can access only at the end of the election, while before the access is denied (by Admins). Note that smart contracts can help to enforce receipt-freeness in non-permissioned block-chains (e.g., in Ethereum): a voter can use a voting contract acknowledging that it acquired the vote (still satisfying the verifiability property); then this vote will be later written on the block-chain by the contract itself.

Data Confidentiality and Neutrality Votes must be protected from external reading during the voting process. Once a vote has been confirmed in a non-permissioned block-chain, it is not confidential anymore, and can influence successive voters, who can freely read the block-chain. Therefore, this property is not enforced in the Bitcoin block-chain implementation (Section 4), but it is enforced by using a permissioned block-chain as in the MultiChain platform (Section 6), when the access to read the block-chain is granted to some given nodes at the end of the election, for counting purposes.

Accuracy Election systems should record the votes correctly, with an extremely small error-tolerance. The protocol reliability resistance resides in the presence of a (large) peer-to-peer network, in distributed consensus, and in cryptographic functions. If a transactions is altered by errors, it will not be mined in the block-chain or counted for the final results.

The last property we discuss is **Reliability**: *Elections systems should work robustly, without loss of any votes, even in the face of numerous failures, including voting machines and total loss of network communication.* Clearly, reliability depends on many factors and it is not easy to be measured (e.g., with a simulation). However, Bitcoin already proves to be a reliable and largely used infrastructure. Indeed, it is required to use transactions with a high fee in order to not lose votes; nevertheless, a voter can check if her votes has been included in the block-chain. Clearly, the size of the peer-to-peer network and the number of miners mitigate such problems. On the other end, using MultiChain requires to set up a new as large as possible peer-to-peer network from scratch, in order to be

resistant to DDoS attacks to miners, or manipulate the consensus (see Section 5).

8 Related Work

Nowadays, the most spread voting schemes consists in paper-based elections. However, paper-based systems are not completely secure and they may suffer from frauds, even in today's democratic countries,²⁶ where controversies are very frequent.²⁷ Estonia became the first nation to hold general elections over the Internet with a pilot project for the municipal elections in 2005. The e-voting system withstood the test of reality and was declared a success by Estonian election officials [1]. Despite this, e-voting systems have not experienced a breakthrough in Europe, since most of the diffidence resides in the general level of trust in government, but also the level of trust in the corporations that supply the machines use in the electoral process [30].

Some proposals have been already opened in the direction outlined by this paper. The most noticeable reference is the *Bitcongress.org project*,²⁸ which already offers a voting platform based on Bitcoin. However, the software is offered as a broker between the voter and Bitcoin. An evidence is the presence of a "Smart Contract Block-chain": quoting the project white-paper, "A vote token is sent by a legislation creation tool with combined cryptocurrency wallet. The vote is sent to a smart contract based election holding yay, nay and candidate addresses". On the contrary, in our implementation a vote is directly sent to the address of a candidate, without any intermediary. Moreover, still quoting the white-paper, "The election logs then changes, the vote count is recorded and displayed within Axiomity (a decentralised application) using Bitcongress onto the Smart Contract Block-chain". In our solution the counting is directly performed in the block-chain. Other commercial systems are *Follow my vote*²⁹ and *TIVI*.³⁰

Envisioning the use of block-chains for voting purposes has been already proposed in [39, 40, 46] for

²⁶http://news.bbc.co.uk/2/hi/uk_news/4410743.stm.

²⁷<http://news.bbc.co.uk/2/hi/europe/4904294.stm>.

²⁸<http://www.bitcongress.org>.

²⁹FollowMyVote.com: <https://followmyvote.com>.

³⁰TIVI: <https://tivi.io>.

example. In the following we present related scientific works. All of them propose solutions without the help of coloured coins or permissioned ledgers, which have been used to respectively simplify the counting process and satisfy further properties, as shown in Section 7: properties as data confidentiality and uncoercibility seem not to be addressed in all such proposals; with MultiChain, or in general permissioned ledgers, it is possible instead. The proposal in [6] simply consists in an electronic voting system based on the Bitcoin block-chain technology.

In [26] the author proposes an e-voting scheme, which is then implemented in the Ethereum block-chain. The implementation and related performance measurements are given in the paper along with the challenges presented by the block-chain platform to develop a complex application like e-voting. In general, special attention must be paid to the debugging and verification steps on (Ethereum) smart-contracts.

Even the execution of the protocol in [32] is enforced using the consensus mechanism that also secures the Ethereum block-chain. However, by using a permissionless block-chain, public verifiability does not provide any coercion resistance.

9 Conclusion

We have presented two E2E verifiable e-voting systems based on *i)* Bitcoin and *ii)* *MultiChain*. The underlying idea is the same for both of them: a transaction between a voter and a candidate represents a vote, which is broadcast to the peer-to-peer network and verified by miners. We describe a solution that is fully compliant with the current Bitcoin network. Moreover, we also suggest some possible modifications that can improve the performance in counting votes, by using OAP and a permissioned block-chain. We have an implementation of a voting system using OAP and a public Web-interface to it. The second implementation is based on MultiChain, and has the advantage of being lighter in terms of the consensus process, being independent from Bitcoin fees and costs, being permissioned and thus satisfying more properties in Section 7. However, an implementation in Bitcoin can immediately exploit the already existing network, and its largely tested reliability and security.

Depending on the requested features on the election to be performed, one solution can be preferred

instead of the other. For instance, for a quick election with a few tens of voters and no concern about neutrality and uncoercibility, voting with Bitcoin can be the best choice. On the contrary, for large (political?) elections, it would be better to set up a large peer-to-peer network in MultiChain; in such a way, voting costs are reduced (only due to the hardware) and more required properties are satisfied.

In [11] and [12] we study the Bitcoin block-chain from a different perspective, that is by visualising flows of money with the help of a forensics application, whose architecture is described in [10]. The works in [12] and [10] describe the tool we developed, while the work in [11] presents a specific case-study to show and to study all the ransoms paid to decrypt data due to *WannaCry* ransomware. Finally, in [33] we present an analysis of Bitcoin standard and non-standard transactions (mined by only some miners in the network).

In the future we would like to formally prove how the presented systems satisfy the properties in Section 7. To prove them we need a formal language and a verifier built on top of it. At the moment, no available and general framework to prove properties of distributed ledgers exists. Some first proposals have been introduced for the Bitcoin platform [5], thus we plan to extend such a framework to check also MultiChain behaviour. However, some of the properties are more quantitative than qualitative (e.g., reliability, and accuracy), and a formal verifier cannot easily help. In this case, a simulation with a scaling number of nodes will be used to measure this property.

References

1. Alvarez, R.M., Hall, T.E., Trechsel, A.H.: Internet voting in comparative perspective: The case of Estonia. *PS: Polit. Sci. Polit.* **42**(3), 497–505 (2009)
2. Androulaki, E., Cachin, C., De Caro, A., Sorniotti, A., Vukolic, M.: Permissioned blockchains and hyperledger fabric. *ERCIM News*, 2017(110) (2017)
3. Androulaki, E., Karame, G., Roeschlin, M., Scherer, T., Capkun, S.: Evaluating user privacy in Bitcoin. In: *Financial Cryptography and Data Security - 17th International Conference, FC, Volume 7859 of Lecture Notes in Computer Science*, pp. 34–51. Springer (2013)
4. Antonopoulos, A.M.: *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. O'Reilly Media, Inc. (2014)
5. Atzei, N., Bartoletti, M., Zunino, R.: A formal model of Bitcoin transactions. In: *To Appear in Financial Cryptography and Data Security - 22nd International Conference, FC, LNCS*. Springer (2018)

6. Ayed, A.B.: A conceptual secure blockchain-based electronic voting system. *International Journal of Network Security & Its Applications*, 93 (2017)
7. Back, A.: Hashcash - a denial of service counter-measure. <http://www.hashcash.org/papers/hashcash.pdf>. [Online; Accessed 28 Jan 2018] (2002)
8. Biryukov, A., Khovratovich, D., Pustogarov, I.: Deanonymisation of clients in bitcoin P2P network. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 15–29. ACM (2014)
9. Bistarelli, S., Mantilacci, M., Santancini, P., Santini, F.: An end-to-end voting-system based on bitcoin. In: Proceedings of the Symposium on Applied Computing, SAC, pp. 1836–1841. ACM (2017)
10. Bistarelli, S., Mercanti, I., Santini, F.: A suite of tools for the forensic analysis of bitcoin transactions: Preliminary report. In: Euro-Par 2018: Parallel Processing Workshops - Euro-Par 2018, Revised Selected Papers, Volume 11339 of Lecture Notes in Computer Science, pp. 329–341. Springer (2018)
11. Bistarelli, S., Parrocchini, M., Santini, F.: Visualizing bitcoin flows of ransomware Wannacry one week later. In: Proceedings of the Second Italian Conference on Cyber Security, Volume 2058 of CEUR Workshop Proceedings. CEUR-WS.org (2018)
12. Bistarelli, S., Santini, F.: Go with the -bitcoin- flow, with visual analytics. In: Proceedings of the 12th International Conference on Availability, Reliability and Security ARES, pp. 38:1–38:6. ACM (2017)
13. Borras, J., Webber, D.: Election Markup Language (EML) Specification Version 7.0. <http://docs.oasis-open.org/election/eml/v7.0/cs01/eml-v7.0-cs01.pdf>. [OASIS, online; Accessed 28 Jan 2016] (2011)
14. Chaum, D.: Blind signatures for untraceable payments. In: *Advances in Cryptology: Proceedings of CRYPTO '82*, Santa Barbara, California, USA, August 23–25, 1982, pp. 199–203. Plenum Press, New York (1982)
15. Chaum, D.: Secret-ballot receipts: True voter-verifiable elections. *IEEE Secur. Privacy* **2**(1), 38–47 (2004)
16. Chirgwen, R.: Android Bug Batters Bitcoin Wallets. http://www.theregister.co.uk/2013/08/12/android_bug_batters_bitcoin_wallets/. [The Register, online; Accessed 28 Jan 2018] (2013)
17. Clinch, M.: Bitcoin Hacked: Price Stumbles After Buying Frenzy. <http://www.cnn.com/id/100615508>. [CNBC, online; Accessed 28 Jan 2018] (2014)
18. Elwell, C.K., Murphy, M.M., Seitzinger, M.V.: Bitcoin: Questions, Answers, and Analysis of Legal Issues. <https://fas.org/sgp/crs/misc/R43339.pdf>. Congressional Research Service: prepared for members and committees of Congress. [Online; Accessed 28 Jan 2018] (2015)
19. Eyal, I., Sirer, E.: Majority is not Enough Bitcoin Mining is Vulnerable. CoRR, arXiv:1311.0243 (2013)
20. Fouard, L., Duclos, M., Lafourcade, P.: Survey on electronic voting schemes. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.295.7959&rep=rep1&type=pdf>. [Verimag technical report, online; Accessed 28 Jan 2018] (2007)
21. Gilbert, H., Handschuh, H.: Security analysis of sha-256 and sisters. In: *International Workshop on Selected Areas in Cryptography*, pp. 175–193. Springer (2003)
22. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. *SIAM J. Comput.* **18**(1), 186–208 (1989)
23. Greenspan, G.: Multichain private blockchain—white paper. <http://www.multichain.com/download/MultiChain-White-Paper.pdf> (2015)
24. Gritzalis, D.A.: *Secure Electronic Voting*, vol. 7. Springer Science & Business Media (2012)
25. Grubb, B.: Australian Bitcoin Bank Hacked: \$ 1 Million + Stolen. <http://www.brisbanetimes.com.au/it-pro/security-it/australian-bitcoin-bank-hacked-1m-stolen-20131108-hv2i.html>. [Brisbane Times, online; Accessed 28 Jan 2018] (2013)
26. Hardwick, F., Akram, R.N., Markantonakis, K.: E-Voting with Blockchain: An E-Voting Protocol with Decentralisation and Voter Privacy. CoRR, arXiv:1805.10258 (2018)
27. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ecdsa). *Int. J. Inf. Secur.* **1**(1), 36–63 (2001)
28. Juhász, P., Stéger, J., Kondor, D., Vattay, G.: A Bayesian approach to identify bitcoin users. *PloS One* **13**(12), e0207000 (2018)
29. Kremer, S., Ryan, M., Smyth, B.: Election verifiability in electronic voting protocols. In: *European Symposium on Research in Computer Security ESORICS*, Volume 6345 of LNCS, pp. 389–404. Springer (2010)
30. Loeber, L., Dutch Electoral Council: E-voting in the Netherlands; from general acceptance to general doubt in two years. *Electron Voting* **131**, 21–30 (2008)
31. Lu, L., Han, J., Liu, Y., Hu, L., Huai, J., Ni, L.M., Ma, J.: Pseudo trust: Zero-knowledge authentication in anonymous p2p. *IEEE Trans. Parallel Distrib. Syst.* **19**(10), 1325–1337 (2008)
32. McCorry, P., Shahandashti, S.F., Hao, F.: A smart contract for boardroom voting with maximum voter privacy. In: *Financial Cryptography and Data Security*, Volume 10322 of Lecture Notes in Computer Science, pp. 357–375. Springer (2017)
33. Mercanti, I., Bistarelli, S., Santini, F.: An analysis of non-standard bitcoin transactions. In: *Crypto Valley Conference on Blockchain Technology, CVCBT*, pp. 93–96. IEEE (2018)
34. Merkle, R.C.: A digital signature based on a conventional encryption function. In: *Advances in Cryptology - CRYPTO*, Volume 293 of LNCS, pp. 369–378. Springer (1987)
35. Moon, M.: Bitcoin Exchange Loses \$5 Million in Security Breach. <http://www.engadget.com/2015/01/06/bitstamp-bitcoin-exchange-hack/>. [Engadget, online; Accessed 28 Jan 2018] (2015)
36. Mote, C.D.: Report of the national workshop on internet voting: Issues and research agenda. In: Proceedings of the 2000 Annual National Conference on Digital Government Research, pp. 1–59. Digital Government Society of North America (2000)
37. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System. <http://www.hashcash.org/papers/hashcash.pdf>. [Online; Accessed 28 Jan 2018] (2008)
38. Okupski, K.: Bitcoin protocol specification. <http://www.enetium.com/resources/Bitcoin.pdf>. [Accessed 28 Jan 2018] (2014)

39. Omohundro, S.: Cryptocurrencies, smart contracts, and artificial intelligence. *AI Matters* **1**(2), 19–21 (2014)
40. Pilkington, M.: 11 Blockchain Technology: Principles and Applications. *Research Handbook on Digital Transformations*, 225 (2016)
41. Rather, E., Colburn, D.R., Moore, C.H.: The evolution of forth. In: *ACM Sigplan Notices*, vol. 28, pp. 177–199. ACM (1993)
42. Reid, F., Harrigan, M.: An analysis of anonymity in the bitcoin system. In: *PASSAT/SocialCom 2011, Privacy, Security, Risk and Trust*, pp. 1318–1326. IEEE (2011)
43. Sampigethaya, K., Poovendran, R.: A framework and taxonomy for comparison of electronic voting schemes. *Comput. Secur.* **25**(2), 137–153 (2006)
44. Schneider, A., Meter, C., Hagemester, P.: Survey on Remote Electronic Voting. arXiv:1702.02798 (2017)
45. Steiner, J.G., Neuman, B., Schiller, J.I.: Kerberos: An authentication service for open network systems. In: *Proceedings of the USENIX Winter Conference*, pp. 191–202. USENIX Association (1988)
46. Swan, M.: *Blockchain: Blueprint for a New Economy*. O'Reilly Media, Inc. (2015)
47. Tapscott, D., Tapscott, A.: *Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and the World*. Penguin (2016)
48. Weisenthal, J.: Bitcoin Service Instawallet: We've Been Hacked and are Suspending Service Indefinitely. <http://www.businessinsider.com/instawallet-suspended-2013-4>. [Business Insider, online; Accessed 18 Jan 2018] (2013)
49. Zhu, L., Leach, P., Hartman, S.: Anonymity Support for Kerberos. RFC 6112 (Proposed Standard) (2011)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.