Blockchain based election

Guillaume A. Khayat

guill.khayat@gmail.com

March 13, 2022

1 Introduction

This note describes a blockchain designed for use in elections, coded in Python 3.10.2 and published on the following GitHub repository.

Advantages

An election using this blockchain offers the following advantages:

- **Voter's anonymity:** no one can identify who created an eligible vote from a pool of N randomly chosen eligible electors (N to be decided by the government before the start of the election)
- **Transparency:** anyone can verify the validity of every step of the process including the below
  - Any valid vote originates from an eligible voter
  - Only valid votes are counted in the final step
  - All valid votes are counted in the final step
  - Why each of the non-valid votes was rejected?
  - Vote counting and result calculation

Limitations

An election using this blockchain includes the following limitations:

- **Attacks:** flooding the system with false votes is the main attack that could jeopardize the election process
- **Vulnerabilities:** this blockchain design is vulnerable to read vs. read/write rights. If a malicious user is able to write in the vote's file directly instead of the internal functions of the blockchain, this user can change the result of the election despite not being able to identify the details of each vote
- **Centralized block creation:** this blockchain design is compatible with a decentralized block creation, but additional checks needs to be added to account for forks. The advantage for any voter to be able to check the validity of every step in the election process is significant, but the added value of the decentralization of the block creation is not clear and adds different type of potential attacks to the election process. Therefore, I chose to maintain the centralized block creation and verification feature and the blockchain design to allow for a possible extension to decentralized verification and block creation

1.1 Technical implementation

This blockchain uses secp256k1 as a curve in the Elliptic Curve Digital Signature Algorithm implementation and Elliptic Curve nonce creation used in the ring signature. The SHA256 function is used whenever a hash is needed.

Below is a brief description of the implementation code. Unless specified differently, all files are Python files (.py).

- Root:
  - impPackages.bat: installs the packages needed for the blockchain
  - globImp: imports all needed packages in the blockchain
  - globParams: sets the main parameters used in the blockchain
- Gvt: a folder where the central government's databases and codes are stored
  - Db: folder where databases which should be accessed to by only the central government are stored
  - a01genKeys: generates all secret information needed by eligible voters to create a vote
  - a02voteRights: creates voting rights used in the election process
  - a03RSAkeys: creates secret and public RSA keys used to ensure the secrecy of the vote message when a vote is created
  - a04genBckChain: creates the initial block in the blockchain (only non-child block)
  - a05verifElecResult: verifies the blockchain and calculate the election result
- Voter: a folder where all databases and codes of the voter V are stored
  - Db: folder where databases which should be accessed to by only the voter V are stored
  - vote: creates eligible votes of different voters
  - falseVotes: creates non-valid votes for testing purposes
- Verif: a folder where all public information and codes are stored
  - PubParams: folder where databases which stores public parameters are stored
  - VoteBckChain: folder where databases which stores votes and the blockchain are stored
  - verifVoteCrtBck: verifies votes and creates a block of valid votes
- Util: a folder where useful functions and classes are stored
  - Cls: a folder where useful classes are stored
  - Fcts: a folder where useful functions are stored

2 Election preparation

The central government decides on N, the length of the ring signature. The anonymity of each eligible voter is guaranteed among N eligible voters. The central government also produces the below:

- Eligible voters' secret information
- Voting rights
- RSA secret and public keys

2.1 Voters' secret information

For each eligible voter V, the central government generates the below and shares it only with V (at the exception of the public key which will be publicly shared):

- Secret and public keys
- Two nonces:
  - First nonce to be used in the ring signature to identify that the vote is generated by V
  - Second nonce to be used in generating the voting right ID, the vote's vote right hash and concatenated to the vote message. This ensures that the message in the vote is provided by the V
- (N-1) randomly generated numbers to be used as signatures in the ring signature

2.2 Voting rights

The central government generates the list of voting rights and publishes it. Each voting right of the eligible voter V includes:

- Voting right hash: the hash of V's 2^{nd} nonce
- Signatures: the signatures used in the ring signature that verifies the identity of V
- Public keys: N compressed public keys that include V's public key and (N-1) randomly chosen eligible voters to be included in the ring signature
- EC points: N nonce EC points used in the ring signature

2.3 RSA secret and public keys

RSA secret and public keys (number of bits: 1500). The public key to be shared before opening the ballots will be used to encrypt the vote message (concatenated to V's 2^{nd} nonce) when V creates his vote and the secret key to be shared after closing the ballots is used to decrypt all vote messages when calculating the election result.

3 Voting

V follows the below steps to create his vote:

- V fetches his voting right
- V creates the ring signature (see Gibson, 2018 for more details on ring signatures) that identifies him. The message to be included in the ring signature is the concatenation of the compressed EC point K and V's 1^{st} nonce
- The list of the messages eL identifies the validity of the vote and since V's 1^{st} nonce is included to the message before it's hashed, only V can generate eL
- V publishes his vote that includes the below information. It is necessary that some of the below is generated automatically by the internal functions and not by V to avoid potential attacks:
  - The vote's ID: the hash of the hash of V's 2^{nd} nonce concatenated to the time stamp
  - The vote's voting right: the hash of V's 2^{nd} nonce
  - The list of the messages eL from the ring signature
  - The time stamp when the vote is created

4 Vote verification

Vote verification and election result calculations can be done as a single step after the closure of the ballots, but I keep the blockchain design to allow the extension of the below to decentralized verification and possibly ongoing incremental verification by the government while the ballots are open.

As writing in the JSON where votes are saved should be strictly through the internal functions of the blockchain, the first verification layer is sufficient to guarantee the validity and origin of the vote. But I add a second layer of identity verification given the importance of this validity and the cheap computation cost of the second verification layer.

4.1 First verification layer: the ring signature

- V's 1^{st} nonce (which only the voter V has access to) is concatenated to the message to be signed in each of the ring signature steps
  => this guarantees that if a vote is created with the messages list eL that validates the ring signature
- As all public keys and all EC nonce points included in the ring signature are part of the voting right, it is impossible to identify which eligible voter V created the valid vote only from the voting right and eL
- Potential attack:
  - Copy any vote that is created and change only the vote message, this is prevented by the additional step below and the second verification layer
  - Each created vote includes a time stamp of its creation time, from all valid votes related to the voting right VR keep the vote with the earliest time stamp (V must create his vote so others could copy it)

4.2 Second verification layer: vote message

The below steps add an additional verification step that guarantees that the vote message in the vote is created by the eligible voter V:

1. The hash of V's 2^{nd} nonce identifies his voting right
2. This nonce is concatenated to the vote message before its RSA encryption
3. After the closure of the ballots, vote that pass the 1^{st} validation step are selected
4. Messages from the valid votes are RSA decrypted
5. The concatenated nonce is hashed and a verification that it is equal to the voting right hash is made

5 Block creation & verification

While the ballots are open, the central government selects only valid votes (1^{st} step verification only), creates a block and appends the blockchain by the creates block. The block includes the below:

- Block hash: the hash of the hash of the parent block concatenated to the vote IDs added to this block
- Parent block's hash: the hash of the parent block
- Vote IDs: the IDs of the votes added to this block

To verify a given blockchain the below steps are followed:

1. Reorder the blockchain (initial block to the only non-parent block in the blockchain)
2. Verify that all votes included in each block are valid, if not keep only the first n valid blocks (the block n + 1 being the first non-valid block)

6 Result calculations

After the ballots are closed, the below steps are done to calculate the election result:

1. Extract vote IDs from valid blocks from the blockchain
2. RSA decrypt the message in each of the above votes
3. Verify that the hash of the concatenated nonce (V's 2^{nd} nonce) is identical to the voting right hash
4. After selecting only votes that pass step (3), calculate the election result

References

Gibson, A. (2018). An investigation into confidential transactions. `https://github.com/AdamISZ/ConfidentialTransactionsDoc/blob/master/essayonCT.pdf` (visited on February 24, 2022).