How to Vote Privately Using Bitcoin

Zhichao Zhao(✉) and T.-H. Hubert Chan

The University of Hong Kong, Pokfulam, Hong Kong
zczhao@cs.hku.hk

Abstract. Bitcoin is the first decentralized crypto-currency that is currently by far the most popular one in use. The bitcoin transaction syntax is expressive enough to setup digital contracts whose fund transfer can be enforced automatically.

In this paper, we design protocols for the bitcoin voting problem, in which there are n voters, each of which wishes to fund exactly one of two candidates A and B. The winning candidate is determined by majority voting, while the privacy of individual vote is preserved. Moreover, the decision is irrevocable in the sense that once the outcome is revealed, the winning candidate is guaranteed to have the funding from all n voters. As in previous works, each voter is incentivized to follow the protocol by being required to put a deposit in the system, which will be used as compensation if he deviates from the protocol. Our solution is similar to previous protocols used for lottery, but needs an additional phase to distribute secret random numbers via zero-knowledge-proofs. Moreover, we have resolved a security issue in previous protocols that could prevent compensation from being paid.

1 Introduction

Private e-voting is a special case of secure multi-party (MPC) computation [9] which allows a group of people to jointly make a decision such that individual opinion can be kept private. However, the MPC framework only guarantees that the outcome is received by everyone, whose privacy is also protected. When the decision is financially related, it is not obvious how to ensure that the outcome is respected. For instance, a dishonest party may simply run away with his money.

Bitcoin [15] provides new tools for tackling this problem. Although it was originally intended for money transfer, surprisingly it can also be used to enforce a contract such that money transfer is guaranteed once the outcome is known, without the need of a trusted third party.

Our Problem. We study the *bitcoin voting problem*. There are n voters $P_1, \ldots, P_n$, each of which wishes to fund exactly one of two candidates A and B with 1฿[1]. The winning candidate is determined by majority voting (assuming n is odd) and receives the total prize n฿. The voting protocol should satisfy the following basic properties:

This research is partially funded by a grant from Hong Kong RGC under the contract HKU719312E.

[1] We use the latex code from [3] to generate the bitcoin symbol ฿.

S. Qing et al. (Eds.): ICICS 2015, LNCS 9543, pp. 82–96, 2016.
DOI: 10.1007/978-3-319-29814-6_8

– *Privacy and Verifiability.* Only the number of votes received by each candidate is known, while individual votes are kept private.
– *Irrevocability.* Once the final outcome of the voting is revealed, the winner is guaranteed to receive the total sum n฿.

In order to incentivize voters to follow the protocol, each voter needs extra bitcoins as deposit, which will be refunded if he follows the protocol, but will be used as compensation if he deviates from the protocol. The candidates A and B also need to participate in the protocol to collect the prize, however they do not need to own bitcoins initially.

Our Approach. In voting, each voter P_i has a private vote $O_i \in \{0, 1\}$ ($0, 1$ stands for A, B respectively), and the sum $\sum_i O_i$ reveals the winning candidate, where the O_i's must be kept private. Our voting protocol consists of two parts:

– *Vote Commitment.* The n voters generate n random numbers R_i's summing to 0 (mod N) using a distributed protocol, where R_i is kept secret and commitments to R_i and the *masked vote* $\widehat{O}_i = R_i + O_i$ are public known, together with a zero-knowledge proof which proves the values are generated correctly. By using zero-knowledge proofs we enforce the voters to obey the protocol.
– *Vote Casting.* This part of the protocol utilizes the bitcoin system for the voters to reveal their $\widehat{O}_i$'s, which also guarantees that the winner can receive the prize and any voter that does not reveal his masked vote is penalized. In Sect. 3.2, we use the *claim-or-refund* technique as in [8] to design a protocol in which the voters reveal their masked votes sequentially. As a result, this protocol takes $\Theta(n)$ rounds and $\Theta(n^2)$ bytes in the bitcoin network. In Sect. 3.3, improved upon the previous section, we use the idea of a *joint transaction* as in [4,13] to reveal the masked votes, where only constant number of rounds and $O(n)$ bytes in the bitcoin network are used.

Previous Security Issue and Our Fix. In vote casting, we use a similar idea as that of [4]. As pointed out in [4], the way timed-commitment is used presents a serious security flaw because the compensation is paid with a transaction depending on the hash of a (unconfirmed) joint transaction. In the protocol described in [4], an adversarial party could create an alternate joint transaction with a different hash by resigning it. The details are discussed in Sect. 3.3 of our full paper [16]. The cause of this issue is that the joint transaction is signed by each voter individually, who has the ability to produce a different signature. We resolve this issue by using a threshold signature scheme [10] in which a valid signature can only be produced by all voters together, which prevents the previous attack.

Our Constribution. In this paper, we propose a protocol that solves the bitcoin voting problem.

– We design a vote commitment scheme that hides individual votes in random numbers, where verifiability and privacy are guaranteed by using zero-knowledge proofs. This part can be used independent of the bitcoin network.

– We design vote casting schemes that place transactions in the bitcoin network in a carefully designed way. Such that any adversaries deviating from the protocol will be punished.
– We point out a security issue from previous protocol, which we fix by using threshold signature schemes.

The rest of the paper is organized as follows: In Sect. 2, we define Bitcoin and related cryptographic primitives. In Sect. 3, we present our protocol, which contains vote commitment and vote casting. In Sect. 4, we discuss implementation details and the performance of our protocols.

1.1 Other Related Work

The bitcoin protocol [15] has inspired many lines of research since its introduction in 2008. Some researchers have worked on identifying the protocol's weakness [5]. Other researchers have designed new crypto-currency with more features. For instance, zero-knowledge proofs and accumulators have been used to improve the currency's anonymity [6,14]. In [6], a new system is designed such that the *proof-of-work* to mine new coins is achieved by storage consumption, as opposed to computation power in bitcoin.

Another line of research, including this paper, is to design protocols that are compatible with the existing bitcoin system. Since bitcoin is still by far the most popular crypto-currency, protocols that can be deployed in the current bitcoin system have the most practical impact. For instance, a lottery protocol [4] was proposed, in which a group of gamblers transfer all their money to a randomly selected winner. General secure multi-party computation protocols [8,13] were also considered, in which bitcoin provides a way to penalize dishonest users.

Basic functionalities has been implemented in the bitcoin system that can serve as building blocks for more complicated protocols. In [8], the *claim-or-refund* mechanism is designed to enforce the promise made by party P that he will pay party Q a certain amount, provided that Q (publicly) reveals a certain secret before a certain time. As utilized in this paper, this mechanism allows a protocol in which parties reveal their secrets sequentially such that the first party that deviates from the protocol has to compensate the parties that have already revealed their secrets.

In [4], the *timed-commitment* mechanism is designed to enforce the promise made by party P that he will pay party Q a certain amount, unless he (publicly) reveals a certain secret before a certain time. In [13], the timed-commitment mechanism is extended for multiple-parties to reveal their secrets together. However, as mentioned in [4], their implementation of the timed-commitment mechanism has a security issue such that an adversary could prevent the compensation from being paid even when a party does not reveal his secret before the deadline.

2 Preliminaries

Bitcoin. We use the same terminology for Bitcoin [15] as in [4]. Bitcoin consists of a block of *transactions* known as the *blockchain*, which is maintained and

synchronized at each peer. New transactions are packed into block and linked at the end of the blockchain at a fixed time interval. Each transaction contains multiple *inputs*, *outputs* and an optional *locktime*, and is indexed by its hash called txid.

Validation of Transactions. An output of a transaction specify a program indicating how the coin (unspent output) can be redeemed. Each input must refer to an unspent output which is specified by its txid and the output index. The input contains an input-script which servers as parameters of the referred output (as a program). For the transaction to validate, the referred output (as a program) must evaluates to true on all the inputs, all referred outputs must be unspent, and the current time must be no earlier than the locktime.

For a detailed description of the Bitcoin network, please see our full paper [16].

Zero-Knowledge Proof. We utilize the zero-knowledge Succinct Non-interactive ARgument of Knowledge (zk-SNARKs) [7]. Zero-knowledge proof allows a party to convince others that he knows a secret witness w such that $C(x, w) = 1$, where x is known by all parties. E.g., "I know an w such that $\mathsf{sha256}(w) = x$". The use of zk-SNARKs is to guarantee that the voters cannot deviate from the protocol. We use the definition for zk-SNARKs from [7]. Informally, zk-SNARKs is a triple of (randomized) algorithms (G, P, V), where G is the key generator which runs only once universally and outputs the key-pairs (pk, vk), $P(pk, x, w)$ is the prover and $V(vk, x, \pi)$ is the verifier.

We informally summarize the properties satisfied by zk-SNARKs.

- *Completeness.* Prover P can produce a proof that is accepted by V.
- *Soundness and Proof of Knowledge.* No polynomial-time adversary can fake a proof for x, without knowing its witness w.
- *Efficiency.* The algorithms runs in time polynomial in the sizes of their inputs and the given security parameter.
- *Zero-knowledge.* The proof leaks no information other than the statement itself.

Commitment Schemes. Commitment schemes allow one party to hide to a secret value which is opened in a later phase to another party, where the other party is convinced that the opened value is the original one. As it turns out, the Bitcoin protocol is restricted to certain operations, which limits the choice of commitment schemes. In Sect. 2.3 of our full paper [16], we give the formal definition and discuss commitment schemes that is compatible with the Bitcoin protocol.

Security Model. In this paper, we assume that the blockchain is unique (no branching) and one block is grown at a fixed time interval known as *round*. The only ways to interact with the blockchain are submitting transactions and reading transaction histories. No one can affect the blockchain by other "non-standard" ways. We assume that the blockchain is always publicly accessible. However, we do not assume such access is private. At the time a transaction is submitted, it is publicly known. A submitted transaction may or may not appear

on the blockchain, depending on its validity. If a transaction is valid, it will be *confirmed* one round later, otherwise it will be *rejected*. If conflicting transactions are submitted in the same round, only one of them will be confirmed. Hence, if an adversary is able to create a different input-script for a (newly submitted) unconfirmed transaction. He could submit another transaction with the modified input-script within the same round. In such case, either transaction could appear in the final blockchain.

A type of attack called signature *malleability* needs to be considered. Briefly, it means creating a valid signature from an existing one, without the corresponding plain text. Many protocols [4,13] suffers from this attack especially when a joint transaction is concerned. For a detailed discussion of malleability attack, please see our full paper [16]. We tackle this issue by using a threshold signature scheme, which we describe in details through the protocols.

Peers (voters and candidates) need to communicate in the protocol. We assume there exists a secure private channel between any pair of participants. We also assume there exists a public broadcast channel among all participants.

3 Our Protocols

We present our protocol for bitcoin voting. Apart from a (universal) one-time setup using zk-SNARKs [7], our protocol works in a peer-to-peer fashion without a centralized server. Suppose N is the least power of 2 that is greater than the number n of voters. We use $\mathbb{Z}_N$ to denote the group of integers modulo N. We choose N to be a power of 2 to simplify the implementation of modulo arithmetic.

On a high level, our protocol consists of two components.

- **Vote Commitment.** In this phase, each voter P_i has a private vote $O_i \in \{0, 1\}$, where 0 indicates candidate A and 1 indicates candidate B. Each voter P_i receives a secret random number R_i, which is constructed in a distributed fashion such that $\sum_j R_j = 0$.
  At the end of this phase, each voter P_i makes commitment C_i to R_i, and commitment $\widehat{C}_i$ to his masked vote $\widehat{O}_i := O_i + R_i$. The commitments C_i and $\widehat{C}_i$ are broadcast publicly, while the underlying values and opening keys remain secret.
  Every participant convinces others that he follows the protocol using zero-knowledge proofs. In particular, everyone is convinced that $\sum_i R_i = 0$ and $\widehat{O}_i - O_i \in \{0, 1\}$ with respect to the commitments. This part is described in Sect. 3.1.
- **Vote Casting.** In this phase, the votes are cast using transactions in the bitcoin protocol, which are responsible for revealing the outcome and guaranteeing money transfer to the winning candidate. After each voter P_i reveals his masked vote $\widehat{O}_i$, the outcome $\sum_i \widehat{O}_i$ (the number of votes supporting B) is known, and the winning candidate is guaranteed to receive nɃ.
  Moreover, parties that deviate from the protocol are penalized. We have two versions for vote casting, which have different consequences for the penalty and the funding outcome for the candidates when a voter deviates from the protocol.

(a) The first version is based on the lottery protocol in [8] using a *claim-or-refund* functionality. The voters reveal their masked votes in the order: $P_1, P_2, \ldots, P_n$. If voter P_i is the first to deviate from the protocol, he pays a penalty to each voter that has already revealed his masked vote. Everyone else gets his deposit back, and the protocol terminates while neither candidate A nor B gets any money.

(b) The second version is an improvement over other protocols [4,13] using *joint transaction* to incentivize fair computation via the bitcoin system. Each voter P_i places $(1+d)$฿ into the bitcoin system, where 1฿ is for paying the winning candidate and d฿ is for deposit. If a voter reveals his masked vote $\widehat{O}_i$, he can get back the deposit d฿. For each voter that does not reveal his masked vote within some time period, his deposit d฿ will be used as compensation. For instance, with $d = 2n$, the deposit can be shared between the candidates A and B.

The two versions are described in Sects. 3.2 and 3.3, respectively.

3.1 Vote Commitment

Recall that we wish to uniformly generate random numbers R_i's that sum to 0 such that for each i, only voter P_i receives R_i. Moreover, the commitments to R_i and $\widehat{O}_i = (R_i + O_i)$ are public.

We first give a high level idea of the procedure. Imagine that there is an $n \times n$ matrix $[r_{ij}]$ whose entries contain elements from $\mathbb{Z}_N$. The protocol can be described in terms of the matrix as follows.

1. For each i, voter P_i generates the i-th row whose sum $\sum_j r_{ij}$ is zero. This is done by generating $n-1$ random numbers and derive the last.
2. For each i and j, voter P_i sends r_{ij} to P_j via the secret channel.
3. For each i, voter P_i knows the i-th column. Hence, he computes and commits to both $R_i := \sum_j r_{ji}$ and the masked vote $\widehat{O}_i := O_i + R_i$.

The trick here is that commitment schemes and zero-knowledge proofs are used to ensure that every party follows the protocol, while maintaining the secrecy of the random numbers. For a step-by-step description of the algorithm please see Fig. 2.

Security Analysis. The security of the vote commitment protocol follows readily from the security of commitment schemes [11] and zero-knowledge proofs from zk-SNARKs [7]. Observe that as long as at least one party generate his random numbers uniformly at random, the resulting R_i's will still be uniform. Hence no private information is leaked by revealing $\widehat{O}_i$'s. Using commitment schemes and zero-knowledge-proofs, each P_i commits to both R_i and the masked vote $\widehat{O}_i := R_i + O_i$, and convinces everyone that $\widehat{O}_i - R_i \in \{0, 1\}$, while keeping them secret.

3.2 Vote Casting via Claim-or-Refund

This version of the vote casting protocol is based on the lottery protocol in [8] that makes use of bitcoin transactions to guarantee money transfer. The protocol

is not symmetric among the voters, and the voters are supposed to reveal their masked votes in the order: $P_1, P_2, \ldots, P_n$. In order to participate in the protocol, for $i \in [n-1]$, voter P_i needs to place $(i+1)$฿ into the system, and voter P_n needs to place $(3n-1)$฿ into the system. This protocol requires a linear number of bitcoin rounds, as opposed to constant number of rounds in the protocol that is given in Sect. 3.3. The protocol in this section guarantees the following:

- If every voter reveals his masked vote, the net effect is that each voter pays 1฿ to the winning candidate.
- If voter P_i is the first that does not reveal his masked vote, the net effect is that he pays 1฿ to each voter that has already revealed his masked vote. Neither candidate receives any money in this case.

We make use of the claim-or-refund (COR) functionality which is first introduced in [8]. It can be used for "conditionally transfer of coins". Briefly, COR protocol between two parties X and Y guarantees that, after a deposit phase, either X reveals some secret (predetermined by X and Y) to Y, or Y is compensated after some time interval. We assume the readers are familiar with it and the description is omitted from here.

Vote Casting via COR. We show how the vote casting protocol can be implemented with a sequence of COR instances in Figs. 5 and 6. The idea is similar to that in [13]. For n voters, there are $2n$ COR instances. The deposit transactions of the COR instances are placed in the reversed order as the claim transactions are broadcast to reveal the masked votes.

Correctness. The correctness of the protocol is analyzed in the same way as [13]. The voters are supposed to reveal their masked votes in the order: $P_1, P_2, \ldots, P_n$. At the moment just after voters $P_1, P_2, \ldots, P_i$ have revealed their masked votes in the corresponding claim transactions, the net effect is that P_{i+1} has paid each of the previous voters 1฿. Hence, if P_{i+1} does not reveal his masked vote, eventually all outstanding COR instances will expire and the protocol terminates.

On the other hand, if everyone follows the protocol, then at the end all the masked votes can be summed up to determine the winner, who can collect n฿ from P_n.

3.3 Vote Casting via Joint Transaction

The protocol in Sect. 3.2 requires a linear number of bitcoin rounds. In this section, we give an alternative protocol that only needs constant number of bitcoin rounds. Also, our protocol has the advantage of small total transaction size. The total size of the transactions in the protocol is $\Theta(n)$ bytes, instead of $\Theta(n^2)$ from previous protocol.

Loosely speaking, we achieve this by locking all bitcoins involved in a transaction that is jointly signed by all voters. The protocol is symmetric among the n voters. In the protocol, each voter P_i needs $(1+d)$฿, of which 1฿ is to be paid to the winning candidate if everyone reveals his masked vote and the remaining

d฿ is for deposit that will be used for compensation if P_i does not reveal his masked vote. The *timed-commitment* technique in [4] can be used to handle the deposit and compensation.

The protocol guarantees the following:

- If a voter reveals his masked vote, he can get back the deposit d฿.
- If every voter reveals his masked vote, the sum $\sum_i \widehat{O}_i$ determines the winner who receives n฿.
- If at least one voter does not reveal his masked vote, the n฿ originally intended for the winner will be locked. For each voter that does not reveal his masked vote, his deposit will be used for compensation. Here are several options:
  (a) For $d = 2n$, the deposit can be shared between candidates A and B. We shall concentrate on this option in this section.
  (b) For $d = n$, the deposit can be distributed among all voters in a similar way.

There is a potential problem in the "joint transactions" described in [4,13]. We fix this problem by using threshold signature schemes. The detail of the problem is discussed in our full paper [16].

Description of Vote Casting Protocol.

Key Setup. We use the threshold signature scheme [10] for ECDSA. The n voters jointly generate a group address such that voter P_i learns the group public key $\widehat{\mathsf{pk}}$ and his share $\widehat{\mathsf{sk}}_i$ of the private key. No party knows the underlying secret key $\widehat{\mathsf{sk}}$, which could be reconstructed from all parties' secret shares. Each party also has his own bitcoin address, whose key pairs are denoted by $(\mathsf{pk}_i, \mathsf{sk}_i)$.

Coin Lock. Eventually, the n voters will sign some transaction JOIN together, whose inputs are contributed by the n voters. We introduce a protocol that locks the contribution from each voter in a state such that only with all voters' permission can it be redeemed. This ensures that only one version of the JOIN transaction can use these coins later. On the other hand, if the protocol ends prematurely and the JOIN transaction is not created successfully, we wish to let each voter P_i get back his contribution with the transaction BACK_i. The coin lock protocol is described in Fig. 3.

Joint Transaction. In the next step, the n voters shall jointly sign a transaction JOIN using the threshold signature scheme, each with his private key share $\widehat{\mathsf{sk}}_i$. The JOIN transaction has n inputs referring to the LOCK_i's, each of which contributes $(1+d)$฿. It has $(n+1)$ outputs, of which out-prize delivers n฿ to the winning candidate, while each $\mathsf{out\text{-}deposit}_i$ of the remaining n outputs handles the deposit d฿ of each voter.

Using the timed-commitment technique as in [4], the output $\mathsf{out\text{-}deposit}_i$ can be redeemed by a transaction that either (1) reveals the masked vote $\widehat{O}_i$ and is signed with the key associated with P_i, or (2) is signed with the group key. Hence, before JOIN takes effect (by appearing in the blockchain), a transaction PAY_i

with some timelock that can redeem $\mathsf{out\text{-}deposit}_i$ needs to be created and signed using the threshold signature scheme, in case P_i does not reveal his masked vote and his deposit is used for compensation. The details are in Fig. 4.

Outcome Revealing Phase. After JOIN appears on the blockchain, each voter P_i can collect his deposit d฿ (from the output $\mathsf{out\text{-}deposit}_i$) by submitting a CLAIM_i transaction that provides the opening key $\widehat{K}_i$ to reveal his masked vote $\widehat{O}_i$. If all voters have submitted their transactions CLAIM_i's, the winning candidate is determined and can redeem n฿ from out-prize with his signature.

On the other hand, if some voter i does not reveal his masked vote, then the n฿ from out-prize cannot be accessed anymore. However, since PAY_i is publicly known, after time t_2, the d฿ from $\mathsf{out\text{-}deposit}_i$ can be redeemed by PAY_i as compensation.

Correctness. After the coin lock protocol, all the transactions LOCK_i's remain secret, while their hashes and the BACK_i's are publicly known. Observe that before the transaction JOIN appears on the blockchain, any voter can terminate the whole protocol without losing any money by submitting BACK_i to the bitcoin system. On the other hand, once JOIN has appeared on the block chain, no voter can terminate the protocol without either revealing his masked vote or losing his deposit d฿.

4 Experiment

We describe our implementation of the proposed protocols below.

Vote Commitment. We have implemented vote commitment protocol Sect. 3.1. For zk-SNAKRs [7], we choose snarkfront [12]. We translated the required relation into circuits. We run the program using a computer with 4G RAM and Intel Core i5-3570 CPU. The key generator typically takes 5 times longer than the time to generate proofs. However, since it only needs to be run once universally, we omit its running time here as it is not a performance concern. In Fig. 1, we report the time to generate proofs for different number of users. We consider three kinds of proofs: (1) to prove n numbers sum to 0, (2) to prove n numbers sum to the $(n+1)$-st number, (3) to prove the subtraction of a number from another is either 0 or 1.

In zk-SNARKs, the time for verification is only linear in the size of the input (and the security parameter). Typically, it takes less than 0.1 s.

Vote Casting. As a proof of concept, we have executed the protocols in bitcoin (testnet) network [2]. We use bitcoinj Java library to create and send the transactions.

Below we present txid of the transactions. There are 9 voters in our protocol. One may read the full transaction data on chain.so website.

For the protocol using claim-or-refund in Sect. 3.2, we first create a transaction with multiple outputs, each of which acts as the source address of each claim-or-refund transaction. The source of each claim-or-refund transaction can be found with index $0-17$ at:

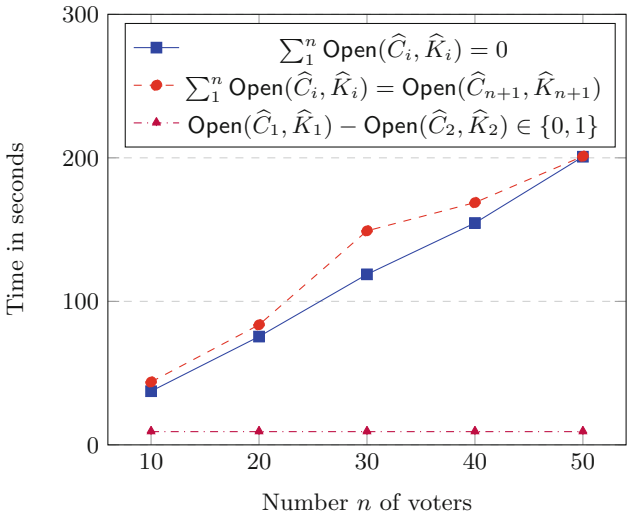

Fig. 1. Performance of zk-SNARKs

https://chain.so/tx/BTCTEST/d3f62d6dfd9722699938a3d7457e23ba786a3e8d14615d128847ad7ca56b7a1a *d3f62d6dfd9722699938a3d7457e23ba786a3e8d14615d128847ad7ca56b7a1a.*

All following transactions can be found following the outputs. Another execution in which an adversary terminated the protocol and was punished is here:

https://chain.so/tx/BTCTEST/8d4031dfa71bf9b1a296b6f67c3cb1d801e899d4ff7d1ee6dd6751622032b60f *8d4031dfa71bf9b1a296b6f67c3cb1d801e899d4ff7d1ee6dd6751622032b60f.*

For the protocol using joint transaction in Sect. 3.3, the JOIN transaction of a successful execution is here:

https://chain.so/tx/BTCTEST/ca42f58d2a7eadc4360029ea31e6f2224c9b7f47c18ef985a9b477ac869822c7 *ca42f58d2a7eadc4360029ea31e6f2224c9b7f47c18ef985a9b477ac869822c7.*

All claim transactions can be found by following the outputs.

A JOIN transaction of an unsuccessful (terminated by adversary) execution is here:

https://chain.so/tx/BTCTEST/6506857a75b1f25b930a923e6bd8274cbccb42339425e21f29cf5ba2ce389738

6506857a75b1f25b930a923e6bd8274cbccb42339425e21f29cf5ba2ce389738.

All CLAIM_i and PAY_i transactions can be found by following the outputs.

5 Conclusion

We present protocols that solve the bitcoin voting problem and run directly on the current bitcoin network. Our protocols consist of two phases. The first phase generates a masked vote for each voter. We guarantee that the generated masks are publicly verifiable by the use of zero-knowledge proofs. The second phase takes the masked votes and put transactions on the blockchain, so that

the winner is guaranteed to receive the prize digitally. The protocols of the two phases are of independent interest and we hope our work will inspire other applications of bitcoin.

Appendix

A Figures

Vote Commitment Protocol

This protocol runs among n voters, where for $i \in [n]$, party P_i has secret vote $O_i \in \{0, 1\}$. We assume the proving and verification keys for zk-SNARKs are already generated and distributed to all voters. For each $i \in [n]$, the procedure for P_i is as follows.

1. Generate n secret random numbers $r_{ij} \in \mathbb{Z}_N$, for $j \in [n]$, such that they sum to 0.
   For $j \in [n]$, commit $(c_{ij}, k_{ij}) \leftarrow \mathsf{Commit}(r_{ij})$, where k_{ij} is the opening key to the commitment c_{ij}.
2. Generate zero-knowledge proofs that shows $\sum_j r_{i,j} = 0$. Specifically, the circuit C takes two components. The input component is the n commitments, while the witness component is the n corresponding opening keys. The circuit C evaluates to 1 if the opened values sum to 0.
   Broadcast the commitments and zero-knowledge proofs to all voters.
3. Receive commitments and verify the zero-knowledge proofs from all other parties generated in Step 2.
4. For all $j \in [n] \setminus \{i\}$, send to P_j the opening key k_{ij}.
   For $j \in [n] \setminus \{i\}$, wait for the opening key k_{ji} from P_j, and check that $r_{ji} = \mathsf{Open}(c_{ji}, k_{ji}) \neq \perp$.
5. Compute $R_i \leftarrow \sum_j r_{ji}$ and $\widehat{O}_i \leftarrow R_i + O_i$, and commit $(C_i, K_i) \leftarrow \mathsf{Commit}(R_i)$ and $(\widehat{C}_i, \widehat{K}_i) \leftarrow \mathsf{Commit}(\widehat{O}_i)$, where K_i,$\widehat{K}_i$ are the opening keys. Broadcast the commitment C_i and $\widehat{C}_i$ publicly.
6. Generate and broadcast publicly the zero-knowledge proofs for the following:
   (a) "$R_i = \sum_j r_{ji}$". This is similar to Step 2.
   (b) "The committed value in $\widehat{C}_i$ minus that in C_i is either 0 or 1." The input part of the circuit is the two commitments C_i and $\widehat{C}_i$, and the witness part is their opening keys. The circuit evaluates to 1 if the opened values differ by 0 or 1 as required.
7. Receive and verify all proofs from other parties generated in Step 6. The protocol terminates.

Fig. 2. Vote commitment protocol

Coin Lock Protocol

Each voter locks $(1+d)$Ƀ into the system, where 1Ƀ is to fund the winner, and dɃ is for deposit; here, we set $d := 2n$. Each voter P_i does the following:

1. P_i creates a (secret) transaction LOCK_i. Its input is $(1+d)$Ƀ owned by P_i, and its output is the address of the group public key $\widehat{\mathsf{pk}}$.
   P_i also creates a simplified transaction BACK_i that transfers the money from LOCK_i back to an address pk_i owned by P_i. Note that $\mathsf{hash}(\mathsf{LOCK}_i)$ is embedded in BACK_i, but LOCK_i remains secret.
   P_i broadcasts (simplified) BACK_i to all other voters.
2. On receiving BACK_j for $j \in [n] \setminus \{i\}$, P_i checks that the hash value referred to by its input is not $\mathsf{hash}(\mathsf{LOCK}_i)$. At this point, P_i has only contributed coins to $\widehat{\mathsf{pk}}$ through the transaction LOCK_i, and hence, he can sign anything else using $\widehat{\mathsf{sk}}_i$ without losing money.
3. For each $j \in [n]$, P_i participates in the threshold signature scheme to sign BACK_j using his secret key share $\widehat{\mathsf{sk}}_i$.
4. On receiving the correct signature for BACK_i, P_i is ready to submit LOCK_i to the bitcoin network later.

Fig. 3. Coin lock protocol

Joint Transaction Protocol

Assume that the Coin Lock Protocol has been run, and each P_i has created the (secret) transaction LOCK_i, whose hash is publicly known. Suppose $t_1 < t_2$ are times far enough in the future. Each voter runs the following protocol.

1. Each voter generates the same simplified transaction JOIN as follows.
   - It has n inputs, each of which refers to LOCK_i that contributes $(1+d)$Ƀ.
   - It has $n+1$ outputs:
     $\mathsf{out\text{-}deposit}_i$, $i \in [n]$: each has value dɃ, and requires either (1) the opening key $\widehat{K}_i$ (revealing $\widehat{O}_i$) and a signature verifiable with P_i's public key pk_i, or (2) a valid signature verifiable with the group's public key $\widehat{\mathsf{pk}}$.
     $\mathsf{out\text{-}prize}$: has value nɃ, and requires all opening keys $\widehat{K}_i$'s (revealing the masked votes $\widehat{O}_i$'s) and a signature from the winning candidate (which can be determined from the sum $\sum_i \widehat{O}_i$).
2. The voters jointly sign JOIN using the threshold signature scheme, each with his private key share $\widehat{\mathsf{sk}}_i$. Observe that JOIN has n inputs, each of which requires its own group signature. (See [1] for details.) The signed JOIN is ready to be submitted.
3. Each voter generates, for each $i \in [n]$, the same simplified transaction PAY_i with timelock t_2 whose input refers to $\mathsf{out\text{-}deposit}_i$. The output handles the compensation dɃ if voter P_i does not reveal his masked vote by time t_2. For instance, with $d = 2n$, the compensation can be shared between candidates A and B. The n voters jointly sign PAY_i using the threshold signature scheme.
4. Each voter P_i verifies that the above steps have been completed, and submit LOCK_i to the bitcoin system.
5. After all LOCK_i's have appeared on the blockchain, JOIN is submitted to the blockchain.
6. As long as JOIN has not appeared on the blockchain, say by time t_1, any voter P_i can terminate the whole protocol by submitting BACK_i to get back $(1+d)$Ƀ.

Fig. 4. Joint transaction

Vote Casting: Deposit Phase

Assume that the commitments $\widehat{C}_i$'s to the masked votes $\widehat{O}_i$'s are publicly known, and each P_i knows the opening key $\widehat{K}_i$ for $\widehat{C}_i$. Assuming that n is odd, the winner is B if $\sum_i \widehat{O}_i > \frac{n}{2}$.
Assume that the times $\tau_1 < \tau_2 < \ldots < \tau_n < \tau_{n+1}$ are spaced sufficiently wide apart, for they will be used as locktimes.
The protocol runs as follows.

1. P_n submits the deposit transactions of the following COR instances to the bitcoin network:
$$P_n \xrightarrow[n,\tau_{n+1}]{\widehat{O}_1,\ldots,\widehat{O}_n:A \text{ wins}} A$$
$$P_n \xrightarrow[n,\tau_{n+1}]{\widehat{O}_1,\ldots,\widehat{O}_n:B \text{ wins}} B$$
2. Simultaneously for each $i \neq n$, P_i verifies that the deposit transactions broadcast in the previous step are on the block chain, and broadcasts the deposit transaction of the following COR instance to the bitcoin system:
$$P_i \xrightarrow[2,\tau_n]{\widehat{O}_1,\ldots,\widehat{O}_n} P_n$$
3. Sequentially for i from n down to 2:
P_i verifies that all deposit transactions broadcast previously have appeared in the blockchain, and broadcasts the deposit transaction of the following COR instance to the bitcoin system:
$$P_i \xrightarrow[i-1,\tau_{i-1}]{\widehat{O}_1,\ldots,\widehat{O}_{i-1}} P_{i-1}$$

Fig. 5. Deposit phase of vote casting

Vote Casting: Claim/Refund Phase

- For $i \neq n$, if before time τ_i, all previous secrets $\widehat{O}_1, \ldots, \widehat{O}_{i-1}$ are revealed, then P_i reveals his secret $\widehat{O}_i$ and use the claim transaction to receive iɃ from P_{i+1}.
- If before time τ_n, all secrets $\widehat{O}_i$ for $i \neq n$ are revealed, P_n reveals his secret $\widehat{O}_n$ and use the claim transactions to receive 2Ƀ from each P_i for $i \neq n$.
- If before time τ_{n+1} all secrets are revealed, the winner is determined and he can use the corresponding claim transaction to receive nɃ from P_n.
- At any time when the locktime of a COR instance has passed, the sender can immediately use the corresponding refund transaction to get his amount back.

Fig. 6. Vote casting: claim/refund Phase

References

1. Checksig - bitcoin wiki. https://en.bitcoin.it/wiki/OP_CHECKSIG (2015). Accessed 10 May 2015
2. Testnet - bitcoin wiki. https://en.bitcoin.it/wiki/Testnet (2015). Accessed 10 May 2015
3. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: How to deal with malleability of bitcoin transactions (2013). CoRR, abs/1312.3230
4. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: IEEE Symposium on Security and Privacy, SP, pp. 443–458. Berkeley, 18–21 May 2014
5. Barber, S., Boyen, X., Shi, E., Uzun, E.: Bitter to better — how to make bitcoin a better currency. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 399–414. Springer, Heidelberg (2012)
6. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: decentralized anonymous payments from bitcoin. In: IEEE Symposium on Security and Privacy, SP, pp. 459–474. Berkeley, 18–21 May 2014
7. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg (2013)
8. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 421–439. Springer, Heidelberg (2014)
9. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: Proceedings of the 20th Annual ACM Symposium on Theory of Computing, pp. 11–19. Chicago, 2–4 May 1988
10. Goldfeder, S., Gennaro, R., Kalodner, H., Bonneau, J., Kroll, J.A., Felten, E.W., Narayanan, A.: Securing bitcoin wallets via a new DSA/ECDSA threshold signature scheme (2015). http://www.cs.princeton.edu/stevenag/threshold_sigs.pdf
11. Goldreich, O.: Foundations of Cryptography, vol. 1. Cambridge University Press, New York (2006)
12. Carlsson, J.: Snarkfront: a c++ embedded domain specific language for zero knowledge proofs. https://github.com/jancarlsson/snarkfront
13. Kumaresan, R., Bentov, I.: How to use bitcoin to incentivize correct computations. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pp. 30–41. Scottsdale, 3–7 Nov 2014
14. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: anonymous distributed e-cash from bitcoin. In: IEEE Symposium on Security and Privacy, SP, pp. 397–411. Berkeley, 19–22 May 2013
15. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). http://bitcoin.org/bitcoin.pdf
16. Zhao, Z., Hubert Chan, T-H.: How to vote privately using bitcoin. Cryptology ePrint Archive, Report 2015/1007 (2015). http://eprint.iacr.org/