How to deal with malleability of BitCoin transactions

Marcin ANDRYCHOWICZ*, Stefan DZIEMBOWSKI**, Daniel MALINOWSKI*** and ŁUKASZ MAZUREK†

University of Warsaw

Abstract. BitCoin transactions are malleable in a sense that given a transaction an adversary can easily construct an equivalent transaction which has a different hash. This can pose a serious problem in some BitCoin distributed contracts in which changing a transaction’s hash may result in the protocol disruption and a financial loss. The problem mostly concerns protocols, which use a “refund” transaction to withdraw a deposit in a case of the protocol interruption. In this short note, we show a general technique for creating malleability-resilient “refund” transactions, which does not require any modification of the BitCoin protocol.

Applying our technique to our previous paper “Fair Two-Party Computations via the BitCoin Deposits” (Cryptology ePrint Archive, 2013) allows to achieve fairness in any Two-Party Computation using the BitCoin protocol in its current version.

1 Malleability of BitCoin transactions

We assume that the reader is familiar with the BitCoin protocol and in particular with non-standard transaction scripts (used e.g. in so-called distributed contracts). For general description of BitCoin, see e.g. [4] or BitCoin wiki page http://en.bitcoin.it/ For the description of non-standard transaction scripts, see [2] or Contracts page http://en.bitcoin.it/wiki/Contracts.

BitCoin transactions are malleable in a sense that given a transaction $T$ it is easy to create a functionally identical transaction $T'$ ($T$ and $T'$ differs only in the input scripts) which has a different hash. This gives an adversary an opportunity to slightly change the transaction sent by a user before it is included in the blockchain. It strongly affects the distributed contracts which use the hashes of the transactions before broadcasting them.

The source of the malleability is the fact that in the current version of the BitCoin protocol, each transaction contains a hash of the whole transaction it spends, while the signatures are taken over the simplified version of the transaction (excluding the input scripts).

The most common scenario in which the malleability of transactions is a problem is the following. Suppose that there is a transaction Deposit, which should be redeemed by a transaction Fuse with time-lock $t$, but for some reason Fuse has to be created and signed before Deposit is broadcast. In the above scenario a problem arises if the Deposit transaction is maliciously changed and its version included in the blockchain has a different hash than expected, what invalidates the transaction Fuse.

In our recent paper [1] we proposed a modification of BitCoin which eliminates the malleability problem. The idea of this modification was to identify the transactions by the hashes of their simplified versions (excluding the input scripts). With this modification one can of course still modify the input script of the transaction, but the modified transaction would have the same hash. We used this improvement of BitCoin to guarantee the correctness of the Fuse transactions, which had to be sign before broadcasting its input transaction. In this short note we present another approach to achieving the correctness of Fuse transactions which does not need any modification of the BitCoin protocol.

---

* marcin.andrychowicz@crypto.edu.pl
** stefan.dziembowski@crypto.edu.pl
*** daniel.malinowski@crypto.edu.pl
† lukasz.mazurek@crypto.edu.pl

1 See http://en.bitcoin.it/wiki/Transaction_Malleability
2 This can be done e.g. by adding push and pop commands to the input script
3 Transactions of this kind are sometimes called refund transactions.
4 See e.g. examples 1, 5 and 7 on http://en.bitcoin.it/wiki/Contracts
2 New technique

Our technique uses a BitCoin-based timed commitment scheme introduced in [2]. We briefly describe this commitment scheme in Sec. 2.1. Later in Sec. 2.2 we show how to construct Fuse transactions, which are resistant to malleability and in Sec. 2.3 we apply it to SCS protocol from [1], what leads to a general fair Two-Party Computation protocol, which is secure in the current version of the BitCoin protocol (in particular, even if transactions are malleable). In Sec. 2.4 we list other protocols, which can be made resistant to malleability using our technique.

2.1 BitCoin-based timed commitment scheme

![Diagram](image)

Pre-conditions:

1. The protocol is executed between the Committer C holding the key pair C and the Recipient R holding the key pair R.
2. The Committer knows the secret string s.
3. The blockchain contains an unredeemed transaction T with value d, which can be redeemed with the key C.

The CS. Commit(C, R, d, t, s) phase:

1. The Committer computes \( h = H(s) \) and broadcasts the transaction Commit. This obviously means that he reveals h, as it is a part of the transaction Commit.
2. The Committer waits until the transaction Commit is confirmed. Then, he creates the body of the transactions Fuse, signs it and sends the signature to the Recipient.
3. If the Recipient does not receive the signature or the signature is incorrect, then he quits the protocol.

The CS. Open(C, R, d, t, s) phase:

4. The Committer broadcasts the transaction Open, what reveals the secret s.
5. If within time t the transaction Open does not appear on the blockchain then the Recipient broadcasts the transaction Fuse and gains d.

Fig. 1. The CS protocol. The scripts’ arguments, which are omitted are denoted by \( \bot \).
makes a deposit of $d \frac{1}{2}$, which is returned to him if he opens the commitment before time $t$ or taken by the Recipient otherwise.

We follow the notation from [21] in which the transactions are represented as boxes. The graph of transactions and the full description of the CS protocol is presented on Fig. 1. Refer to [22] for more details. Notice that even if the transaction Commit is maliciously changed before being included in the block, the protocol still succeeds because the transaction Fuse is created after Commit is included in the blockchain, so it always contains the correct hash of Commit. Therefore, the CS protocol is resistant to transaction malleability.

The execution of the commitment phase with C as the Committer and R as the Recipient will be denoted by CS.Commit($C, R, d, t, s$), where $d$ is the size of the deposit and $t$ is the time until which $C$ should reveal the secret $s$.

2.2 **Fuse** transactions resistant to malleability

Suppose that in the execution of some protocol between the parties $A$ and $B$ there is a transaction Deposit, which should be redeemed to an address controlled by $A$ at the time $t$ if it is not spent earlier. The typical solution would be to create a transaction Fuse with time-lock $t$, which is signed by both parties and redeems Deposit. Moreover, Deposit has to be claimable using signatures of both parties. The graph of transactions for this situation is presented on Fig. 2.

![Fig. 2. The typical solution with the Fuse transaction vulnerable to malleability.]

We will now present a technique for creating Fuse transactions, which are resistant to malleability. The general idea is to use a timed commitment instead of using a time-lock directly. More precisely, the transaction Deposit should be claimable with a signature of $A$ and a random secret $r$, which is known only to $B$. It means that $A$ can claim Deposit using the Fuse transaction as soon as the secret string $r$ is revealed by $B$. In our situation we would like the secret to be revealed at the time $t$. It can be achieved by executing at the very beginning (before broadcasting the Deposit transaction) the CS.Commit($B, A, d, t, r$) protocol. In this case at the time $t$ either: the CS commitment was opened, the secret $r$ is known and $A$ can broadcast the Fuse transaction (assuming that Deposit was not spent earlier) or the CS commitment was not opened and $A$ gets $d \frac{1}{2}$ from the Fuse transaction in the CS execution. The graph of transactions is presented on Fig. 3.

![Fig. 3. The solution with the Fuse transaction resistant to malleability. CS($B, A, d, t, r$) denotes the transactions in the appropriate execution of the CS protocol.]

---

5 See e.g. examples 1, 5 and 7 on [http://en.bitcoin.it/wiki/Contracts](http://en.bitcoin.it/wiki/Contracts)
The key difference, which makes a new construction resistant to malleability is that the Fuse transaction does not need to be signed by B, so it can be created and signed by A after the Deposit transaction is confirmed and its hash is known. A drawback of this construction is that the other party (B in our case) also has to make a deposit.

2.3 Fair Two-Party Computation protocol

The simultaneous BitCoin-based timed commitment scheme (SCS) described in [1] is an extended version of the CS protocol in which two parties simultaneously commit to their secret strings. The pivotal property of this protocol is that after the commitment phase either: both parties are committed or none of them is committed (the latter is only possible if one of the parties misbehaved). The graph of transactions is presented on Fig. 4. Refer to [1] for more details.

The main application of the SCS protocol is the FairComputation protocol from [1], which is a general fair Two-Party Computation protocol (refer to [1] for more details). However, in contrast to CS, the SCS protocol is vulnerable to transaction malleability, because it requires the Fuse transactions to be created and signed before broadcasting their input transaction. Therefore, in [1] it was assumed that the BitCoin protocol is modified in such a way, that the transactions are no longer malleable.

In this section we present a modified version of SCS protocol called NewSCS, which is resistant to transactions malleability and does not require any change of the BitCoin protocol. Combining it with the FairComputation protocol from [1] it gives the general fair Two-Party Computation protocol.

![Fig. 4. The graph of transactions for the original version of the SCS protocol.](image)

The NewSCS protocol is a result of a straightforward application of the technique from Sec. 2.2 to the SCS protocol. The graph of the transactions and the full description of the NewSCS protocol are presented on Fig. 5 and Fig. 6.

2.4 Other applications

In this section we list some other protocols, which can be made resistant to malleability using our technique:

- [http://en.bitcoin.it/wiki/Contracts](http://en.bitcoin.it/wiki/Contracts), Example 1: Providing a deposit. Although, this protocol could be fixed using our technique, the resulting protocol would be rather impractical as it would require the server to also make a deposit.
Example 5: Trading across chains. Example 7: Rapidly-adjusted (micro)payments to a pre-determined party. Back and Bentov’s lottery protocol from [3].

References

1. M. Andrychowicz, S. Dziembowski, D. Malinowski, and Ł. Mazurek. Fair Two-Party Computations via the BitCoin Deposits. Cryptology ePrint Archive, 2013.
2. M. Andrychowicz, S. Dziembowski, D. Malinowski, and Ł. Mazurek. Secure Multiparty Computations on BitCoin. Cryptology ePrint Archive, 2013. [http://eprint.iacr.org/2013/784](http://eprint.iacr.org/2013/784).
3. Adam Back and Iddo Bentov. Note on fair coin toss via bitcoin, 2013. [http://www.cs.technion.ac.il/~idddo/cointossBitcoin.pdf](http://www.cs.technion.ac.il/~idddo/cointossBitcoin.pdf).
4. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
Pre-conditions:

1. A holds the key pair A and B holds the key pair B.
2. A knows the secret $s_A$, B knows the secret $s_B$, both players know the hashes $h_{s_A} = H(s_A)$ and $h_{s_B} = H(s_B)$.
3. There are four unredeemed transactions $T^A_1, T^A_2, T^B_1, T^B_2$, which can be redeemed with the keys A and B respectively, each having the value of $d_B$.

The NewSCS.Commit$_R(A, B, d, t, s_A, s_B)$ phase:

1. A draws a random string $r_A$ and B draws a random string $r_B$.
2. The parties execute CS.Commit(A, B, d, t, $r_A$) and CS.Commit(B, A, d, t, $r_B$) using $T^A_1$ and $T^B_1$ respectively. The former execution will be denoted CS$^A$ and the latter CS$^B$. Recall that the parties quit the whole NewSCS protocol if they detect the misbehavior of the other party during one of the CS.Commit executions.

The NewSCS.Commit$_S(A, B, d, t, s_A, s_B)$ phase:

3. Both players compute the body of the transaction Commit using $T^A_2$ and $T^B_2$ as inputs.
4. A signs the transaction Commit and sends the signature to B.
5. B signs the transaction Commit and broadcasts it.
6. Both parties wait until the transaction Commit is confirmed.
7. If the transaction Commit does not appear on the blockchain until the time $t - 3\max_{BB}$, where $\max_{BB}$ is the maximal possible delay between broadcasting the transaction and including it in the blockchain, then A immediately redeems the transaction $T^A_2$ and after $T^A_2$ is redeemed she opens her CS$^A$ commitment and quits the protocol. Analogously, if A did not send her signature to B until the time $t - 3\max_{BB}$, then B opens his CS$^B$ commitment and quits the protocol.

The NewSCS.Open(A, B, d, t, s_A, s_B) phase:

8. A and B broadcast the transactions Open$^A$ and Open$^B$ respectively, what reveals the secrets $s_A$ and $s_B$.
9. After the transactions Open$^A$ and Open$^B$ are confirmed, A and B open their CS commitments.
10. If A did not broadcast Open$^A$ until time $t$, then depending on whether she opened her commitment CS$^A$ or not, B broadcasts Fuse$^A$ or CS$^A$.Fuse to get extra $d_B$ (in addition to $2d_B$ already claimed from Open$^B$ and CS$^B$.Open). Similarly, if B misbehaved then A broadcasts Fuse$^B$ or CS$^B$.Fuse to get her extra $d_B$.

Fig. 6. The description of the NewSCS protocol.