A Denial of Service Attack against Fair Computations using Bitcoin Deposits

Jethro Beekman

July 2014
Updated January 2015

Abstract

Bitcoin supports complex transactions where the recipient of a transaction can be programmatically determined. Using these transactions, multi-party computation protocols that aim to ensure fairness among participants have been designed. We present a Denial of Service attack against these protocols that results in a net loss for some or all of the honest parties involved, violating those fairness goals.

1 Introduction

Several recent works by Andrychowicz et al. [1, 2] (Protocol “ADMM”) and Bentov and Kumaresan [3] (Protocol “BK”) describe multi-party computation schemes in which Bitcoin deposits are used to ensure fairness. The general idea is that parties in the computation make a deposit at the beginning of the computation, which honest parties will get back in the end. This incentivizes parties to share their result of the computation with the other parties.

In this work, we introduce a Denial of Service (DoS) attack that results in a net loss for honest parties, destroying the incentive for honest parties to participate. In our attack, dishonest parties will turn a profit at the cost of the honest parties, which incentivizes participants to cheat. This undermines the incentive structure of the underlying protocols. In particular, we note that the security models of ADMM and BK did not consider the possibility of network-level DoS. We show how a dishonest party can use network-level DoS against honest parties.

2 Background

ADMM and BK are protocols for secure multi-party computation that are intended to be fair. Traditional multi-party computation has the problem that one or more dishonest parties might be able to learn the result of the distributed computation and then walk away, so the honest parties never learn the result.
of the computation. A perfectly fair protocol is one where this cannot happen: intuitively, either everyone learns the outcome of the computation, or no one does. In ADMM and BK, fairness is encouraged monetarily, but not guaranteed. Fairness is accomplished by having all parties initially pay a deposit. Dishonest parties who walk away forfeit their deposit and it is split among the honest parties as compensation, while honest parties receive their deposit back after the computation is finished. This is roughly how fairness and security are defined for ADMM\(^1\) and BK.\(^2\)

ADMM and BK use Bitcoin to define complex transactions like

\[
P_3 \leftarrow \tau: \frac{P_1}{q} \xrightarrow{\mathcal{C}} P_2
\]

which means \(P_1\) posts a Bitcoin transaction depositing \(q\) bitcoins (BTC); then, \(P_2\) can post a Bitcoin transaction satisfying condition \(\mathcal{C}\) \textit{and} collecting \(q\) BTC before time \(\tau\); otherwise, after time \(\tau\), \(P_3\) can post a Bitcoin transaction collecting \(q\) BTC. The protocols use a sequence of these transactions to provide fairness.

For example, the 2-party BK protocol [3, §3.1] is defined as

\[
P_1 \leftarrow \tau: \frac{P_1}{q} \xrightarrow{\mathcal{C}_1 \land \mathcal{C}_2} P_2
\]

\[
P_2 \leftarrow \tau: \frac{P_2}{q} \xrightarrow{\mathcal{C}_1} P_1
\]

with \(\tau_1 < \tau_2\). Here, \(P_1\) and \(P_2\) have the ability to satisfy \(\mathcal{C}_1\) and \(\mathcal{C}_2\), respectively. If both parties are honest, \(P_1\) will satisfy \(\mathcal{C}_1\) before \(\tau_1\) by publicly revealing a suitable witness. This lets \(P_1\) receive \(q\) BTC from \(P_2\). Then, \(P_2\) can satisfy \(\mathcal{C}_1 \land \mathcal{C}_2\) before \(\tau_2\) to receive \(q\) BTC from \(P_1\). This means that no one loses their deposits and everyone learns the result of the computation. If \(P_1\) is dishonest and does not satisfy \(\mathcal{C}_1\) in time, \(P_2\) gets \(q\) BTC back at \(\tau_1\) and later \(P_1\) gets \(q\) BTC back at \(\tau_2\). In this case, no one loses their deposits and no one learns the result of the computation. If \(P_2\) is dishonest and does not satisfy \(\mathcal{C}_1 \land \mathcal{C}_2\) in time, \(P_1\) has already gotten \(q\) BTC at \(\tau_1\) and later \(P_1\) gets \(q\) BTC back at \(\tau_2\). Here, \(P_2\) learns the result of the computation while \(P_1\) does not, but \(P_2\) has a net loss of \(q\) BTC and \(P_1\) has a net gain of \(q\) BTC.

ADMM is similar, but uses transactions of the form \(P_2 \leftarrow \tau: \frac{P_1}{q} \xrightarrow{\mathcal{C}} P_1\) instead of \(P_1 \leftarrow \tau: \frac{P_1}{q} \xrightarrow{\mathcal{C}} P_2\) (the difference is in the target parties).

\(^1\)“Formally, we say that the protocol is \textit{secure} if for any strategy of the adversary, that controls the network and corrupts the other parties, (1) the execution of the protocol terminates in [some time], and (2) the expected payoff of each honest party is at least negligible.” [2, §IV]

\(^2\)“Loosely speaking, our notion of fair secure computation guarantees: an honest party never has to pay any penalty; and if a party aborts after learning the output and does not deliver output to honest parties, then every honest party is compensated.” [3, §2.1]
3 Threat model and attack

Our threat model considers an adversary $A$ that participates in a multi-party computation protocol using Bitcoin and is able to perform a network-level Denial of Service attack against another party $B$ in the same computation for extended periods of time. This inclusion of control over the network is consistent with the security definition for ADMM: they assume the adversary has control over the network, which is sufficient to launch a network-level DoS attack against another party.\footnote{We show how an adversary $A$ that pretends to be honest can turn another honest party $B$ into a dishonest party in the eyes of the protocol by performing a DoS attack on $B$ at the appropriate time.}

Let’s reconsider the 2-party BK protocol with $P_1$ being malicious. First, $P_1$ can pretend to be honest and satisfy $C_1$ before $\tau_1$ to collect $q$ BTC. Then, $P_1$ can immediately perform a DoS attack on $P_2$, lasting at least until $\tau_2$. $P_2$ will be unable to post a transaction satisfying $C_1 \land C_2$ during this time, and at $\tau_2$ $P_1$ will be able to collect its original deposit $q$ BTC. $P_1$ now has $2q$ BTC while it deposited just $q$ BTC at the beginning, for a net gain of $q$ BTC. $P_2$ lost its deposit, for a net loss of $q$ BTC, even though it might have been intending and trying to satisfy $C_1 \land C_2$.

Similar attacks work against ADMM (see appendix A) and the same protocols extended for more than 2 parties.

4 Discussion

Denial of Service attacks are notoriously hard to defend against. A potential solution that future work could focus on is using very large time scales. This could give a party under attack an opportunity to reroute the message satisfying the condition. This is not a cure-all, as a powerful enough adversary might still be able to maintain the DoS attack for this prolonged period of time. However, it would give the victim a chance to find another network connection (e.g. at a coffee shop, Internet cafe, etc.) to collect his deposit. Also, using longer time scales would mean that an honest party would have to wait longer to reclaim the deposit from a dishonest party, even when no DoS is in progress, which might make these protocols less attractive.

5 Conclusion

In this work, we have demonstrated a Denial of Service attack against two recent fair multi-party computation protocols. This attack both defies the fairness aimed to be provided by these protocols and violates the security guarantees those protocols claimed to provide. We highlight an avenue future research could explore.
A DoS Attack on Protocol “ADMM”

The attack against ADMM is very similar to the one on BK shown in Section 3. It is detailed here for completeness. Let’s reconsider the 2-party ADMM protocol:

\[
P_2 \leftarrow P_1 \quad \begin{array}{c} \tau \\ q \end{array} \quad \xrightarrow{c_1} P_1 \\
\]

\[
P_1 \leftarrow P_2 \quad \begin{array}{c} \tau \\ q \end{array} \quad \xrightarrow{c_2} P_2 \\
\]

Now consider \( P_1 \) being malicious. \( P_1 \) can pretend to be honest and satisfy \( C_1 \) before \( \tau \) to collect its original deposit \( q \) BTC. Simultaneously, \( P_1 \) can perform a DoS attack on \( P_2 \), lasting at least until \( \tau \). \( P_2 \) will be unable to post a transaction satisfying \( C_2 \) during this time, and at \( \tau \) \( P_1 \) will be able to collect the \( q \) BTC that \( P_2 \) deposited. \( P_1 \) now has \( 2q \) BTC while it deposited just \( q \) BTC at the beginning, for a net gain of \( q \) BTC. \( P_2 \) lost its deposit, for a net loss of \( q \) BTC, even though it might have been intending and trying to satisfy \( C_2 \).