A Denial of Service Attack against Fair Computations using Bitcoin Deposits

Jethro Beekman

July 2014
Updated August 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

$$P_3 \xleftarrow{\quad\tau\quad} \overset{P_1}{\underset{q}{\because}} \xrightarrow{\quad\mathcal{C}\quad} P_2$$

which means P_1 posts a Bitcoin transaction depositing q bitcoins (BTC); then, P_2 can post a Bitcoin transaction satisfying condition $\mathcal{C}$ *and* collecting q BTC before time τ; otherwise, after time τ, 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

$$P_1 \xleftarrow{\quad\tau_2\quad} \overset{P_1}{\underset{q}{\because}} \xrightarrow{\quad\mathcal{C}_1 \wedge \mathcal{C}_2\quad} P_2$$

$$P_2 \xleftarrow{\quad\tau_1\quad} \overset{P_2}{\underset{q}{\because}} \xrightarrow{\quad\mathcal{C}_1\quad} P_1$$

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 τ_1 by publicly revealing a suitable witness. This lets P_1 receive q BTC from P_2. Then, P_2 can satisfy $\mathcal{C}_1 \wedge \mathcal{C}_2$ before τ_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 τ_1 and later P_1 gets q BTC back at τ_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 $\mathcal{C}_1 \wedge \mathcal{C}_2$ in time, P_1 has already gotten q BTC at τ_1 and later P_1 gets q BTC back at τ_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 $P_2 \xleftarrow{\tau} \overset{P_1}{\underset{q}{\because}} \xrightarrow{\mathcal{C}} P_1$ instead of $P_1 \xleftarrow{\tau} \overset{P_1}{\underset{q}{\because}} \xrightarrow{\mathcal{C}} P_2$ (the difference is in the target parties).

---

[1] "Formally, we say that the protocol is *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.[1]

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 $\mathcal{C}_1$ before τ_1 to collect q BTC. Then, P_1 can immediately perform a DoS attack on P_2, lasting at least until τ_2. P_2 will be unable to post a transaction satisfying $\mathcal{C}_1 \wedge \mathcal{C}_2$ during this time, and at τ_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 $\mathcal{C}_1 \wedge \mathcal{C}_2$. This attack prevents the adversary from learning the output of the computation.

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

4 Discussion

Since the adversary will not learn the output of the computation, further analysis of the outcomes is necessary to determine whether performing the attack is worthwhile. For example, in the ADMM fair 2-party lottery protocol [2], each player deposits at least 2 BTC and plays with another 1 BTC. The winner receives their deposit and the prize of 2 BTC for a total gain of 1 BTC. Losers just receive their deposit for a loss of 1 BTC. Using this attack results in the adversary receiving both deposits for a total gain of 1 BTC, the same as a guaranteed win. The victim gets nothing for a loss of 3 BTC and the prize of 2 BTC is stuck in limbo.

If these protocols were to be used in practice, this attack would enable adversaries to turn DoS capability into direct financial gain. Most other DoS monetization schemes rely on being paid for the DoS service [4], extortion [6], or selling goods obtained through influencing online auctions [7].

In previous work, Lindell and Pinkas [5] describe desired properties for ideal secure multiparty computation protocols. One of these is *guaranteed output delivery*: Corrupted parties should not be able to prevent honest parties from receiving their output. As mentioned in the introduction, the protocols discussed in this paper do not exhibit this property but encourage it through financial means. Both protocols are built on a notion of fairness, including that honest parties don't lose their deposit. We can codify this property—in a way similar to guaranteed output delivery—as *guaranteed deposit return*: Corrupted parties should not be able to prevent honest parties from receiving their deposit. ADMM and BK do not provide guaranteed deposit return, as our attacks show. We suggest that guaranteed deposit return should be considered part of what it means to provide fairness and recommend future protocols include this in their designs.

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.

Acknowledgments

Thanks to the anonymous reviewers, Frank Li, and David Wagner for helpful feedback.

References

[1] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. Fair two-party computations via bitcoin deposits. In *1st Workshop on Bitcoin Research*, March 2014.

[2] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. Secure multiparty computations on bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, May 2014.

[3] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *Advances in Cryptology – CRYPTO 2014*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014.

[4] Mohammad Karami and Damon McCoy. Understanding the emerging threat of ddos-as-a-service. In *Presented as part of the 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2013.

[5] Yehuda Lindell and Benny Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1 (1):59–98, 2009.

[6] Denise Pappalardo and Ellen Messmer. Extortion via DDoS on the rise. Network World, 2005. URL
`http://www.networkworld.com/article/2320986`.

[7] Michael Wellman and Peter R. Wurman. Real time issues for internet auctions. In *1st IEEE Workshop on Dependable and Real-Time E-Commerce Systems*, 1998.

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:

$$\begin{tikzcd}
P_2 & \underset{q}{\overset{P_1}{\cdot\cdot}} \arrow[l, "\tau"'] \arrow[r, "\mathcal{C}_1"] & P_1
\end{tikzcd}$$

$$\begin{tikzcd}
P_1 & \underset{q}{\overset{P_2}{\cdot\cdot}} \arrow[l, "\tau"'] \arrow[r, "\mathcal{C}_2"] & P_2
\end{tikzcd}$$

Now consider P_1 being malicious. P_1 can pretend to be honest and satisfy $\mathcal{C}_1$ before τ to collect its original deposit q BTC. Simultaneously, P_1 can perform a DoS attack on P_2, lasting at least until τ. P_2 will be unable to post a transaction satisfying $\mathcal{C}_2$ during this time, and at τ 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 $\mathcal{C}_2$.