论文中，红色荧光表示系统模型的假设；蓝色荧光表示定义；黄色荧光表示文章的创新点；绿色荧光表示文章中某些重要定理。红色下划线表示文章中写的比较好的英文；绿色下划线表示文章中仿真结果和结论。

Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin

Ghassan O. Karame
NEC Laboratories Europe
69115 Heidelberg, Germany
ghassan.karame@neclab.eu

Elli Androulaki
ETH Zurich
8092 Zürich, Switzerland
elli.androulaki@inf.ethz.ch

Srdjan Capkun
ETH Zurich
8092 Zürich, Switzerland
srdjan.capkun@inf.ethz.ch

Abstract

Bitcoin is a decentralized payment system that is based on Proof-of-Work. Bitcoin is currently gaining popularity as a digital currency; several businesses are starting to accept Bitcoin transactions. An example case of the growing use of Bitcoin was recently reported in the media; here, Bitcoins were used as a form of *fast* payment in a local fast-food restaurant.

In this paper, we analyze the security of using Bitcoin for fast payments, where the time between the exchange of currency and goods is short (i.e., in the order of few seconds). We focus on double-spending attacks on fast payments and demonstrate that these attacks can be mounted at low cost on currently deployed versions of Bitcoin. We further show that the measures recommended by Bitcoin developers for the use of Bitcoin in fast transactions are not always effective in resisting double-spending; we show that if those recommendations are integrated in future Bitcoin implementations, double-spending attacks on Bitcoin will still be possible. Finally, we leverage on our findings and propose a lightweight countermeasure that enables the detection of double-spending attacks in fast transactions.

1 Introduction

First introduced in 2009, Bitcoin [21] is an emerging digital currency that has, as of September 2011, approximately 60,000 users [1]. Bitcoin is currently integrated across several businesses [2] and has several exchange markets (e.g., [3]). It is also foreseen that Bitcoin ATMs will be deployed in locations around the globe [4] in order to bridge the gap between digital currency and cash.

Bitcoin is a Proof-of-Work (PoW) based currency that allows users to "mine" for digital coins by performing computations. Users execute payments by digitally signing their transactions and are prevented from double-spending their coins (i.e., signing-over the same coin to two different users) through a distributed time-stamping service [21]. This service operates on top of the Bitcoin Peer-to-Peer (P2P) network that ensures that all transactions and their order of execution are available to all Bitcoin users.

Nowadays, Bitcoin is increasingly used in a number of "fast payment" scenarios, where the exchange time between the currency and goods is short. Examples include online services, ATM withdrawals, vending machine payments and fast-food payments (recently featured in media reports on Bitcoin [5]), where the payment is followed by fast (< 30 seconds) delivery of goods. While Bitcoin PoW-based time-stamping mechanism is appropriate for slow payments (e.g., on-line orders with delivery of physical goods), it requires tens of minutes to confirm a transaction and is therefore inappropriate for fast payments. This mechanism is, however, essential for the detection of double-spending attacks—in which an adversary attempts to use some of her coins for two or more payments. Since Bitcoin users are anonymous and users (are encouraged to) hold many accounts, there is only limited value in verifying the payment after the user obtained the goods (and e.g., left the store) or services (e.g., access to on-line content). The developers of Bitcoin implicitly acknowledge this problem and inform users that they do not need to wait for the payment to be verified as long as the payment is not of high value [6]; this, however, does not solve this problem but merely limits the damage as the system still remains vulnerable to double-spending attacks.

Until now, double-spending attacks on fast payments in Bitcoin or mechanisms for their prevention have not been studied. In this work, we analyze double spending attacks in detail and we demonstrate that double-spending attacks can be mounted

on currently deployed version of Bitcoin, when used in fast payments. We further show that the measures recommended by Bitcoin developers for fast transactions are not always effective in resisting double-spending; we argue that if those recommendations are followed[1], double-spending attacks on Bitcoin are still possible. Finally, we propose a lightweight countermeasure to detect double-spending attacks in fast transactions.

More specifically, our contributions in this paper can be summarized as follows:

- We measure and analyze the time required to confirm transactions in Bitcoin. Our analysis shows that transaction confirmation in Bitcoin can be modeled with a shifted geometric distribution and that, although the average time to confirm transactions is almost 10 minutes, its standard deviation is approximately 15 minutes. We argue that this hinders the reliance of transaction confirmation when dealing with fast payment scenarios.
- We thoroughly analyze the conditions for performing successful double-spending attacks against fast payments in Bitcoin. We then present the first comprehensive double-spending measurements in Bitcoin. Our experiments were conducted using modified Bitcoin clients running on a handful of hosts located around the globe. Our results demonstrate the feasibility and easy realization of double-spending attacks in current Bitcoin client implementations[2].
- We explore and evaluate empirically a number of solutions for preventing double-spending attacks against fast payments in Bitcoin. We show that the recommendations of Bitcoin developers on how to counter double-spending are not always effective. Leveraging on our results, we propose a lightweight countermeasure that enables the secure verification of fast payments.

The remainder of the paper is organized as follows. In Section 2, we briefly present Bitcoin. In Section 3, we review how Bitcoin payments can be processed. In Section 4, we analyze and evaluate the security of fast payments with existing Bitcoin clients. We then evaluate the security of possible measures to alleviate double-spending against fast payments in Bitcoin. In Section 6, we overview related work and we conclude the paper in Section 7.

[1]These recommendations are still not integrated in the Bitcoin implementation.

[2]In our experiments, we solely used Bitcoin wallets and accounts that we own; other Bitcoin users were not affected by our experiments.

2 Background on Bitcoin

Bitcoin is a decentralized P2P payment system [21] that relies on PoW. Electronic payments are performed by generating *transactions* that transfer Bitcoin coins (BTCs) between Bitcoin peers. These peers are referenced in each transaction by means of virtual pseudonyms—referred to as *Bitcoin addresses*. Generally, each peer has hundreds of different Bitcoin addresses that are all stored and managed by its (digital) wallet. Each address is mapped through a transformation function to a unique public/private key pair. These keys are used to transfer the ownership of BTCs among addresses.

Peers transfer coins to each other by issuing a transaction. A transaction is formed by digitally signing a hash of the previous transaction where this coin was last spent along with the public key of the future owner and incorporating this signature in the coin [21]. Any peer can verify the authenticity of a BTC by checking the chain of signatures.

Transactions are included in Bitcoin *blocks* that are broadcasted in the entire network. To prevent double-spending of the same BTC, Bitcoin relies on a hash-based PoW scheme. More specifically, to generate a block, Bitcoin peers must find a nonce value that, when hashed with additional fields (i.e., the Merkle hash of all valid and received transactions, the hash of the previous block, and a timestamp), the result is below a given target value. If such a nonce is found, peers then include it (as well as the additional fields) in a block thus allowing any entity to publicly verify the PoW. Upon successfully generating a block, a peer is typically granted 50 new BTCs. This provides an incentive for peers to continuously support Bitcoin. Table 1 depicts the information included in Bitcoin block number 80,000 as reported in the Bitcoin block explorer [7]. The resulting block is forwarded to all peers in the network, who can then check its correctness by verifying the hash computation. If the block is "valid", then the peers append it to their previously accepted blocks, thus growing the Bitcoin block chain.

The main intuition behind Bitcoin is that for peers to double-spend a given BTC, they would have to replace the transaction where the BTC was spent and the corresponding block where it appeared in, otherwise their misbehavior would be detected immediately. This means that for malicious peers to double-spend a BTC without being detected, they would not only have to redo all the work required to compute the block where that BTC was spent, but also recompute all the subsequent blocks in the

Hash: 000000000043a8c0fd1d6f726790caa2a406010d19efd2780db27bdbbd93baf6
Previous block: 00000000001937917bd2caba204bb1aa530ec1de9d0f6736e5d85d96da9c8bba
Next block: 00000000000036312a44ab7711afa46f475913fbd9727cf508ed4af3bc933d16
Time: 2010-09-16 05:03:47
Difficulty: 712.884864
Transactions: 2
Total BTC: 100
Size: 373 bytes
Merkle root: 8fb300e3fdb6f30a4c67233b997f99fdd518b968b9a3fd65857bfe78b2600719
Nonce: 1462756097

Input/Previous Output	Source & Amount	Recipient & Amount
N/A	Generation: 50 + 0 total fees	Generation: 50 + 0 total fees
f5d8ee39a430...:0	1JBSCVF6VM6QjFZyTnbpLjoCJ...: 50	16ro3Jptwo4asSevZnsRX6vf..: 50

Table 1: Example Block of Bitcoin. The block contains 2 transactions, one of which awards the generator peer with 50 BTCs.

chain[3]. This ensures that the Bitcoin network can counter such misbehavior as long as the fraction of honest peers in the network exceeds that of malicious colluding peers [21].

In what follows, we provide a summary (adapted from [21]) of the steps that peers undergo in Bitcoin when a payment occurs.

- New transactions are broadcasted by peers in the network.
- When a new transaction is received by a peer, it checks whether the transaction is correctly formed, and whether the BTCs have been previously spent in a block in the block chain. If the transaction is correct, it is stored locally in the *memory pool* of peers.
- Peers work on constructing a block. If they find a nonce that solves the PoW, they include all the transactions that appear in their memory pool within the newly-formed block. Peers then broadcast the block in the network.
- When peers receive a new block, they verify that the block hash is valid and that every transaction included within the block has not been previously spent. If the block verification is successful, peers continue working towards constructing a new block using the hash of the last accepted block in the "previous block" field (cf. Table 1).

[3]Bitcoin claims that it is computationally infeasible for an attacker to redo the PoW required to compute 6 consecutive blocks.

Further details on Bitcoin can be found in [8,9,21]. Throughout the rest of the paper, we will adopt the following notations.

Confirmed transactions: These refer to Bitcoin transactions that appear in a valid block. As mentioned earlier, these transactions are checked before being included in a block to prevent double-spending attacks; since they already appear in a block in the Bitcoin block chain, they cannot be modified easily. In the paper, we refer to a transaction that has acquired X confirmations (i.e., $X-1$ blocks appear in the chain after the block that confirms the transaction) by an X-confirmation transaction.

Memory Pool: This is a local structure at each peer that contains all transactions that have been received and not yet confirmed. If a transaction that appears in the memory pool of a given peer is confirmed elsewhere, the transaction is removed from the memory pool. Note that a peer does not need to be involved in any of the transactions appearing in its memory pool.

3 Payments in Bitcoin

In transactions where the exchange between currencies and services happens simultaneously, the payment verification is typically required to be immediate, e.g., *fast* credit-card authorization. Otherwise, the delivery of the services will only occur after the payment is processed, e.g., *slow* delivery by post.

Bitcoin is currently being used in both slow and fast payment scenarios. In this section, we review

and analyze how Bitcoin transactions are processed in these two scenarios. For that purpose, we consider a system that consists of a Bitcoin P2P network, a vendor $\mathcal{V}$ and a set of its customers.

3.1 "Slow Payments"—Transaction Confirmation

As described in Section 2, the most conventional and secure way for $\mathcal{V}$ to accept a payment made by a customer $\mathcal{C}$ is to wait until the transaction issued from $\mathcal{C}$ to $\mathcal{V}$ is confirmed in at least one block before offering service to $\mathcal{C}$. Note that the Bitcoin client can inform $\mathcal{V}$ whether its transactions have been confirmed or not. Since confirmed transactions are likely to be accepted by honest peers in the Bitcoin network, a malicious client $\mathcal{A}$ has negligible advantage in tricking $\mathcal{V}$ to accept incorrect or double-spent transactions.

Transaction Confirmation Time: In what follows, we briefly analyze the time it takes for a given transaction to be confirmed.

Bitcoin is designed so that blocks are generated every 10 minutes, on average. For that purpose, the difficulty of the work required to construct a block is adjusted dynamically depending on the time it took to solve the previous blocks. More specifically, Bitcoin requires that the hash of the block to be constructed is below a given target value. This target is interpolated from the overall time it took to solve the previous 2016 blocks [10]. If it took more than two weeks[4] to generate the last 2016 blocks, the target is increased, otherwise the target is decreased. Since the fields required to construct the hash of a block also change with time (e.g., timestamp, previous block hash), the probability to successfully construct a valid block in Bitcoin is almost constant with respect to the number of trials [11].

To measure the generation time of existing Bitcoin blocks, we created a Python script that parses the block chain of Bitcoin starting from the genesis block (Block # 0) until Block # 153260[5] and extracts the time intervals between the generation of consecutive blocks.

Our findings show that while the average block generation time is approximately 10 minutes (9 minutes and 54 seconds), the standard deviation of the measurements was about 881.24 seconds which corresponds to almost 15 minutes. This shows that there is a considerable variability among the block generation times. In Figure 1, we depict the distribution of the generation times of the extracted Bitcoin blocks. As shown in Appendix A, this distribution can be fitted to a shifted geometric distribution with success probability 0.19. Our results also show that only 64% of the blocks are generated in less than 10 minutes. The remaining 36% of the blocks require between 10 and 40 minutes to be generated.

[4]This corresponds exactly to a generation time of 10 minutes per block.

[5]This block was generated on the 14th of November 2011, as reported on the Bitcoin Block Explorer [12]

3.2 "Fast Payments"—Transaction Reception

Our analysis shows that the time required to confirm transactions impedes the operation of many businesses that are characterized by a fast-service time (i.e., when the exchange of currency and goods is shorter than a minute). As such, it is clear that vendors, such as supermarkets, vending machines, take-away stores [13], etc., cannot rely on transaction confirmation when accepting Bitcoin payments.

To enable fast payments, Bitcoin encourages vendors to give away service without waiting for the transaction verification, as long as the transaction is not of high value [6, 13].

快速支付服务不需要等到交易被验证，且交易并没有很高的价值

As such, for low-cost transactions, Bitcoin payments with zero-confirmations can be accepted as soon as the vendor receives a transaction from the network transferring the correct amount of BTCs from the client to one of its addresses. This verification can be done using current Bitcoin clients since the vendor can search in his wallet for the client's transaction. The main intuition here is that this constitutes sufficient proof that the transaction was indeed broadcasted in the network. We emphasize that it typically takes few seconds (< 3 seconds) for a transaction to propagate between two Bitcoin peers—which explains the use of the terms "zero-confirmation transactions" and *fast payments* interchangeably in the rest of this paper.

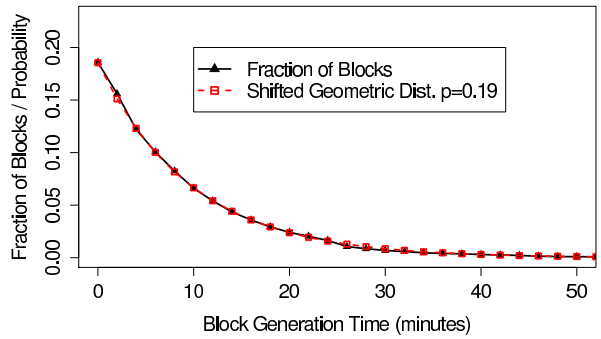
对于低花费交易，采用零确认支付可以被攻击者接受

4 Security of Fast Payments with Current Bitcoin Clients

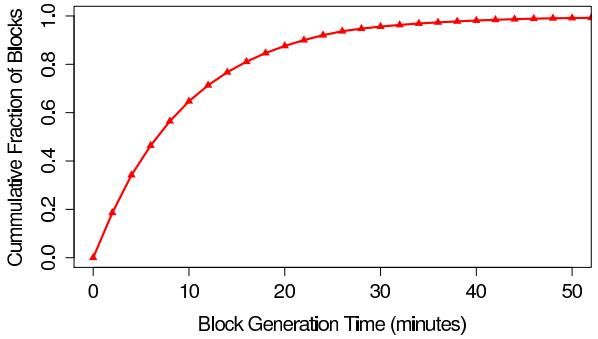
Fast payments cannot rely on the timestamp server of Bitcoin to prevent double-spending. Furthermore, current and past Bitcoin clients (up to version 0.5.2) do not employ any countermeasure against double-spending fast payments. In what follows, we analyze the necessary requirements for mounting a successful double-spending in existing Bitcoin implementations and we describe how to meet these requirements in practical settings.

快速支付如何防止双花交易？

分析发起一个成功的双花攻击的必要条件



(a) Block generation times in Bitcoin. Assuming a (time) bin size of 2 minutes, the block generation function can be fitted to a shifted geometric distribution with $p = 0.19$. Refer to Appendix A for further details.



(b) Cumulative Distribution Function (CDF) of block generation times. 36% of Bitcoin blocks take between 10 and 40 minutes to be generated.

Figure 1: Block generation times in Bitcoin. Transactions are confirmed when they appear in valid blocks. The histogram of block generation times was created assuming a bin size $\delta t = 2$ minutes. Our measurements show that the average time to generate a Bitcoin block is almost 9 minutes and 54 seconds with a standard deviation of 14 minutes and 41 seconds.

4.1 Attacker Model

A malicious client $\mathcal{A}$ constitutes the core of our attacker model. We assume that $\mathcal{A}$ is equipped with a device that runs Bitcoin. We further assume that $\mathcal{A}$ is motivated to acquire a service from $\mathcal{V}$ without having to spend its BTCs. For instance, $\mathcal{A}$ could try to double-spend the coin she already transferred to $\mathcal{V}$. By double-spending, we refer to the case where $\mathcal{A}$ can redeem and use the same coins with which she payed $\mathcal{V}$ so as to acquire a different service elsewhere. We assume that $\mathcal{A}$ can only control few peers in the network (that she can deploy since Bitcoin does not restrict membership) and does not have access to $\mathcal{V}$'s Bitcoin keys or machine. We assume, however, that $\mathcal{A}$ knows the Bitcoin and IP addresses of $\mathcal{V}$[6]. The remaining peers in the network are assumed to be honest and to correctly follow the Bitcoin protocol. We point out here that the computing power harnessed by $\mathcal{A}$ and its helpers does not exceed the aggregated computer power of all the honest peers in the network. This prevents $\mathcal{A}$ from inserting/confirming incorrect blocks in the Bitcoin block chain. In this paper, we consider the scenario where $\mathcal{A}$ does not mine for blocks (i.e., it does not participate in the block generation process). This also suggests that when a transaction is confirmed in a block, this transaction cannot be modified by $\mathcal{A}$.

Conforming with the operation of Bitcoin, we assume that the set of addresses used by $\mathcal{A}$ are insufficient to identify $\mathcal{A}$. This also suggests that although the misbehavior of $\mathcal{A}$ may be detected at some later point in time (after it has acquired a service), its true identity is unlikely to be revealed.

[6]When issuing a transaction, a Bitcoin client could either specify a recipient Bitcoin address or an IP address.

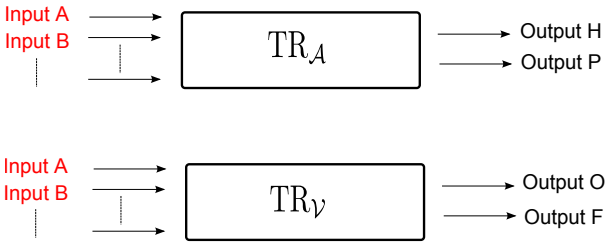


Figure 2: Example construction of $TR_{\mathcal{A}}$ and $TR_{\mathcal{V}}$. Here, we show two different transactions with the same inputs and different outputs. $TR_{\mathcal{A}}$ and $TR_{\mathcal{V}}$ can share a subset of their inputs—in which case $\mathcal{A}$ only tries to double-spend a subset of the BTCs that she pays with.

4.2 Necessary Conditions for Successful Double-Spending

To perform a successful double-spending attack, the attacker $\mathcal{A}$ needs to trick the vendor $\mathcal{V}$ into accepting a transaction $TR_{\mathcal{V}}$ that $\mathcal{V}$ will not be able to redeem subsequently. While this might be computationally challenging for $\mathcal{A}$ to achieve if $TR_{\mathcal{V}}$ was confirmed in a Bitcoin block[7], this task might be easier if the vendor accepts fast payments.

In this case, $\mathcal{A}$ creates another transaction $TR_{\mathcal{A}}$ that has the same inputs as $TR_{\mathcal{V}}$ (i.e., $TR_{\mathcal{A}}$ and $TR_{\mathcal{V}}$ use the same BTCs) but replaces the recipient address of $TR_{\mathcal{V}}$—the address of $\mathcal{V}$— with a recipient address that is under the control of $\mathcal{A}$. Note that our

[7]As mentioned earlier, $\mathcal{A}$ will then have to do all the work required to re-generate the block where the transaction appears in (and all subsequent blocks).

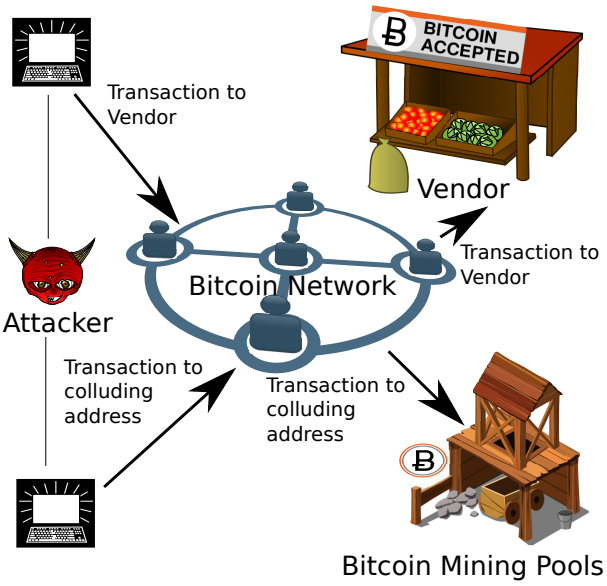


Figure 3: Sketch of a double-spending attack against fast payments in Bitcoin. In typical cases, the attacker $\mathcal{A}$ dispatches two transactions that use the same BTCs in the Bitcoin network. The double-spending attack is deemed successful if the BTCs that $\mathcal{A}$ used to pay for $\mathcal{V}$ cannot be redeemed (i.e., when the second transaction is included in the upcoming Bitcoin block).

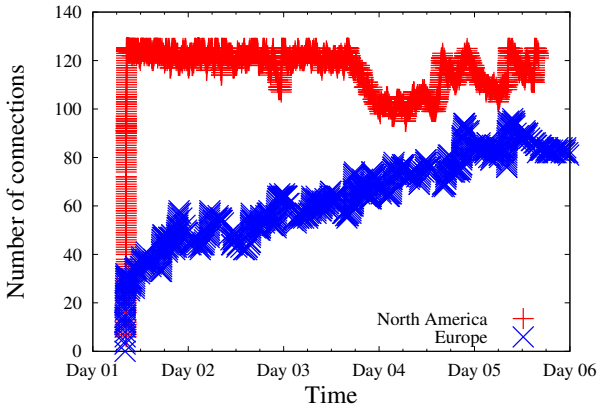


Figure 4: Number of connections that two Bitcoin nodes (in North America and in Europe, respectively) witnessed over the period of 6 consecutive days. Since the connectivity of peers in Bitcoin varies with time, $\mathcal{A}$ has considerable opportunities to establish at least one direct connection with $\mathcal{V}$.

analysis is not restricted to the case where the recipient address is controlled by $\mathcal{A}$ and applies to other scenarios, where the recipient is another merchant.

An example construction of $\text{TR}_\mathcal{A}$ and $\text{TR}_\mathcal{V}$ is depicted in Figure 2. If both transactions are sent adjacently in time, they are likely to have similar chances of getting confirmed in an upcoming block. *This is the case since Bitcoin peers will not accept multiple transactions that share common inputs; they will only accept the version of the transaction that reaches them first which they will consider for inclusion in their generated blocks and they will ignore all remaining versions. Given this, a double-spending attack can succeed if $\mathcal{V}$ receives $\text{TR}_\mathcal{V}$, and the majority of the peers in the network receive $\text{TR}_\mathcal{A}$ so that $\text{TR}_\mathcal{A}$ is more likely to be included in a subsequent block.* This is sketched in Figure 3.

In this respect, let $t_i^\mathcal{V}$ and $t_i^\mathcal{A}$ denote the times at which node i receives $\text{TR}_\mathcal{V}$ and $\text{TR}_\mathcal{A}$, respectively. As such, $t_\mathcal{V}^\mathcal{V}$ and $t_\mathcal{V}^\mathcal{A}$ denote the respective times at which $\mathcal{V}$ receives $\text{TR}_\mathcal{V}$ and $\text{TR}_\mathcal{A}$.

The necessary conditions for $\mathcal{A}$'s success in mounting a double-spending attack are the following:

Requirement 1 — $t_\mathcal{V}^\mathcal{V} < t_\mathcal{V}^\mathcal{A}$: This requirement is essential for the attack to succeed. In fact, if $t_\mathcal{V}^\mathcal{V} > t_\mathcal{V}^\mathcal{A}$, then $\mathcal{V}$ will first add $\text{TR}_\mathcal{A}$ to its memory pool and will reject $\text{TR}_\mathcal{V}$ as it arrives later. After waiting for few seconds without having received $\text{TR}_\mathcal{V}$, $\mathcal{V}$ can ask $\mathcal{A}$ to re-issue a new payment.

Requirement 2 — $\text{TR}_\mathcal{A}$ is confirmed in the block chain: As mentioned previously, if $\text{TR}_\mathcal{V}$ is confirmed first in the block chain, $\text{TR}_\mathcal{A}$ can never appear in subsequent blocks. In this case, $\mathcal{V}$ has received its BTCs, and can redeem them at its convenience. The attack of $\mathcal{A}$ therefore fails unless $\text{TR}_\mathcal{A}$ is confirmed first in the block chain.

We point out that Requirements **(1)** and **(2)** are sufficient for the case where the vendor only checks for the reception of the transaction as a proof of payment and does not employ other double-spending prevention/detection techniques. This accurately mimics the functionality provided by existing Bitcoin client implementations. In Section 5, we extend our analysis and we evaluate the effectiveness of possible measures to alleviate double-spending.

4.3 Mounting Double-Spending Attacks in Bitcoin

In this section, we discuss how $\mathcal{A}$ can satisfy Requirements **(1)** and **(2)**.

Satisfying Requirement 1: $\mathcal{A}$ connects as a direct neighbor to $\mathcal{V}$ in the P2P network[8]. Given the Bitcoin protocol specification, $\mathcal{V}$ will always accept

[8] Recall that the IP address of $\mathcal{V}$ is public.

the connection requests by other peers as long as the maximum number of its current inbound connections has not been reached. By default, this number is set to 125.[9] As shown in Figure 4, the connectivity of Bitcoin peers is heavily dependent on the churn of the Bitcoin network (i.e., peers departing/joining); this gives a considerable number of opportunities for $\mathcal{A}$ to establish a direct connection with $\mathcal{V}$ (e.g., $\mathcal{A}$ could try to connect with $\mathcal{V}$ over the weekend or in the evening, when the number of connections of $\mathcal{V}$ drops below the maximum).

In the sequel, we assume that $\mathcal{A}$ has access to one or more helpers, denoted by $\mathcal{H}$. $\mathcal{A}$ and $\mathcal{H}$ do not necessarily have to be on physically disjoint machines (e.g., $\mathcal{H}$ could run as a thread/process on the same machine as $\mathcal{A}$). We further assume that $\mathcal{A}$ and $\mathcal{H}$ communicate using a low-latency confidential channel (e.g., by exchanging encrypted messages using a direct TCP connection) and that $\mathcal{H}$ never connects to $\mathcal{V}$.

$\mathcal{A}$ sends $\text{TR}_\mathcal{V}$ to $\mathcal{V}$ at time $\tau_\mathcal{V}$ and $\text{TR}_\mathcal{A}$ to $\mathcal{H}$ at time $\tau_\mathcal{A}$, such that $\tau_\mathcal{A} = \tau_\mathcal{V} + \Delta t$. $\mathcal{V}$ and $\mathcal{H}$ relay the transactions that they received from $\mathcal{A}$ in the network. Let $\delta t^\mathcal{A}_{\mathcal{H}\mathcal{V}}$ refer to the time it takes $\text{TR}_\mathcal{A}$ to propagate in the Bitcoin P2P network from $\mathcal{H}$ to $\mathcal{V}$ and $\delta t^\mathcal{V}_{\mathcal{A}\mathcal{V}}$ denote the time it takes $\text{TR}_\mathcal{V}$ to reach $\mathcal{V}$. In this case, $t^\mathcal{A}_\mathcal{V} - t^\mathcal{V}_\mathcal{V}$ can be estimated as follows:

$$t^\mathcal{A}_\mathcal{V} - t^\mathcal{V}_\mathcal{V} \approx \tau_\mathcal{A} + \delta t^\mathcal{A}_{\mathcal{V}\mathcal{H}} - (\tau_\mathcal{V} + \delta t^\mathcal{V}_{\mathcal{A}\mathcal{V}}) \quad (1)$$

$$\approx \Delta t + \delta t^\mathcal{A}_{\mathcal{V}\mathcal{H}} - \delta t^\mathcal{V}_{\mathcal{A}\mathcal{V}}. \quad (2)$$

Note that since $\mathcal{H}$ is never an immediate neighbor of $\mathcal{V}$, there is at least one hop on the path between $\mathcal{H}$ and $\mathcal{V}$. Since $\mathcal{A}$ is an immediate neighbor of $\mathcal{V}$ and assuming no congestion at network paths, then $\delta t^\mathcal{A}_{\mathcal{V}\mathcal{H}} \geq \delta t^\mathcal{V}_{\mathcal{A}\mathcal{V}}$, . This suggests that, in this case, $t^\mathcal{V}_\mathcal{V} < t^\mathcal{A}_\mathcal{V}$ for reasonably chosen Δt (e.g., $\Delta t \geq 0$).

Satisfying Requirement 2: Since $\mathcal{H}$ and $\mathcal{V}$ are highly likely to have different neighbors, the broadcasted transactions are likely to spread in the network till the point where either *(i)* all Bitcoin peers accept in their memory pools $\text{TR}_\mathcal{V}$ or $\text{TR}_\mathcal{A}$ or *(ii)* either $\text{TR}_\mathcal{V}$ or $\text{TR}_\mathcal{A}$ get confirmed in a block.

In what follows, we estimate the probability that $\text{TR}_\mathcal{A}$ is confirmed in a block first. In our analysis, we assume that at time t_0 both transactions $\text{TR}_\mathcal{A}$ and $\text{TR}_\mathcal{V}$ coexist in the network[10], and that no block containing at least one of them has been generated till that time. We argue that this is a realistic assumption given that $\text{TR}_\mathcal{A}$ and $\text{TR}_\mathcal{V}$ need to be typically broadcasted back to back given a small delay (in the order of few seconds); it is therefore unlikely that one of them is confirmed within the first few seconds in a new block. In Section 4.4, we relax this assumption and we evaluate the general case where either $\text{TR}_\mathcal{A}$ and $\text{TR}_\mathcal{V}$ can be confirmed immediately when they are broadcasted in the network.

We divide time into equal intervals of size δt, such that, the probability of successful block generation in each δt can be modeled as a Bernoulli trial with success probability $\eta \cdot p$, where η is the number of peers and p the success probability of a peer in generating a block within δt (for the reasoning why, refer to Appendix A).

Let $t_k = k \cdot \delta t + t_0$ and $\eta^k_\mathcal{V}$ and $\eta^k_\mathcal{A}$ denote the number of Bitcoin peers that have received $\text{TR}_\mathcal{V}$ and $\text{TR}_\mathcal{A}$ respectively until time t_k. Recall that each Bitcoin node will only add the first transaction it receives (among $\text{TR}_\mathcal{V}$ and $\text{TR}_\mathcal{A}$) to its memory pool and that only the transactions that appear in the memory pool of peers are eligible to be confirmed in subsequent blocks. Given this, the probability that $\text{TR}_\mathcal{V}$ is included in a block that is generated within time interval $(t_k,\ t_{k+1}]$ is given by $\Pr^k_\mathcal{V} = \eta^k_\mathcal{V} \cdot p$. Similarly, for $\text{TR}_\mathcal{A}$, the corresponding probability is $\Pr^k_\mathcal{A} = \eta^k_\mathcal{A} \cdot p$. The probability $\mathsf{p}_\mathcal{V}(k)$ that a block containing $\text{TR}_\mathcal{V}$ is generated within the time interval $(t_k,\ t_{k+1}]$, is:

$$\mathsf{p}_\mathcal{V}(k) = \Pr^k_\mathcal{V} \cdot \prod_{i=0}^{k-1}(1 - \Pr^i_\mathcal{V}) = \eta^k_\mathcal{V} p \cdot \prod_{i=0}^{k-1}(1 - \eta^i_\mathcal{V} p).$$

Similarly, the probability that a block containing $\text{TR}_\mathcal{A}$ is generated at the same time interval is given by:

$$\mathsf{p}_\mathcal{A}(k) = \Pr^k_\mathcal{A} \cdot \prod_{i=0}^{k-1}(1 - \Pr^i_\mathcal{A}) = \eta^k_\mathcal{A} p \cdot \prod_{i=0}^{k-1}(1 - \eta^i_\mathcal{A} p).$$

If at time $t_s = s \cdot \delta t + t_0$ every node in the network has received at least one of the transactions $\text{TR}_\mathcal{V}$ or $\text{TR}_\mathcal{A}$, the following holds: 在ts时刻，每个网络中节点至少接收过TRa或TRv，

$$\begin{cases} \eta^k_\mathcal{A} \leq \eta^{k+1}_\mathcal{A} \text{ and } \eta^k_\mathcal{V} \leq \eta^{k+1}_\mathcal{V}, \\ \text{if } k < s \\ \\ \eta^k_\mathcal{A} = \eta^{k+1}_\mathcal{A} = \eta^s_\mathcal{A} \text{ and } \eta^k_\mathcal{V} = \eta^{k+1}_\mathcal{V} = \eta^s_\mathcal{V}, \\ \text{otherwise.} \end{cases}$$

This suggests that $\forall i \geq s$, $\eta^i_\mathcal{V} + \eta^i_\mathcal{A} = \eta^s_\mathcal{V} + \eta^s_\mathcal{A}$.

To compute the probability of success of the double-spending attack, we make the assumption

[9] The default maximum number of connections is 125. Note that the maximum number of connections can be modified using the "-maxconnections" command [14].

[10] This does not necessarily mean that $\text{TR}_\mathcal{A}$ and $\text{TR}_\mathcal{V}$ are broadcasted at the same time.

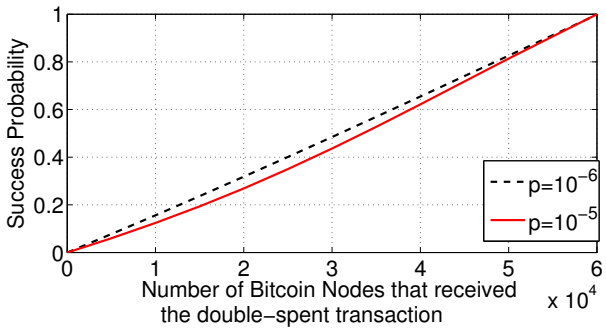


Figure 5: P_S for various values of $\eta_{\mathcal{A}}^k$ and $\eta_{\mathcal{V}}^k$, with $p = 10^{-6}$, $\delta t = 10$ seconds, $t_0 = 0$, $t_s = \delta t$ and the number of peers in the network is 60000. Further details can be found in Appendix B.

that, $\forall k$, $\eta_{\mathcal{V}}^k$ and $\eta_{\mathcal{A}}^k$ do not exchange their newly constructed blocks; in this way, the time $t_{g_{\mathcal{V}}}$ required by peers that are mining in favor of $TR_{\mathcal{V}}$ to generate a new block is independent of that required by the peers that are mining in favor of $TR_{\mathcal{A}}$, $t_{g_{\mathcal{A}}}$. Given this, the probability that Requirement **(2)** is satisfied, $P_S^{(2)}$, is computed as follows:

$$P_S^{(2)} = \text{Prob}(t_{g_{\mathcal{A}}} < t_{g_{\mathcal{V}}}) + \frac{1}{2}\text{Prob}(t_{g_{\mathcal{A}}} = t_{g_{\mathcal{V}}}). \quad (3)$$

That is, P_S is composed of two components: one corresponds to the event that the block containing $TR_{\mathcal{A}}$ is first generated and the second to the event where the blocks containing $TR_{\mathcal{A}}$ and $TR_{\mathcal{V}}$ are generated at the same time, i.e., $t_{g_{\mathcal{A}}} = t_{g_{\mathcal{V}}}$. In the latter case, the probability that the block containing $TR_{\mathcal{A}}$ is eventually adopted by the Bitcoin peers is 0.5.

$$\text{Prob}(t_{g_{\mathcal{A}}} < t_{g_{\mathcal{V}}}) = \sum_{g_{\mathcal{A}}=0}^{\infty} p_{\mathcal{A}}(g_{\mathcal{A}}) \cdot p_{\mathcal{V}}(g_{\mathcal{V}} > g_{\mathcal{A}} | g_{\mathcal{A}})$$

$$= \eta_{\mathcal{A}}^0 p(1 - \eta_{\mathcal{V}}^0 p) +$$

$$\sum_{g_{\mathcal{A}}=0}^{\infty} \eta_{\mathcal{A}}^{g_{\mathcal{A}}} p \cdot (1 - \eta_{\mathcal{V}}^{g_{\mathcal{A}}} p) \cdot \prod_{j=0}^{g_{\mathcal{A}}-1} (1 - \eta_{\mathcal{V}}^j p)(1 - \eta_{\mathcal{A}}^j p).$$

Similarly,

$$\text{Prob}(t_{g_{\mathcal{A}}} = t_{g_{\mathcal{V}}}) =$$

$$\sum_{g_{\mathcal{A}}=1}^{\infty} p^2 \eta_{\mathcal{V}}^{g_{\mathcal{A}}} \eta_{\mathcal{A}}^{g_{\mathcal{A}}} \cdot \prod_{j=0}^{g_{\mathcal{A}}-1} (1 - \eta_{\mathcal{V}}^j p)(1 - \eta_{\mathcal{A}}^j p).$$

In Figure 5, we depict $P_S^{(2)}$ for various values of $\eta_{\mathcal{V}}^k$, $\eta_{\mathcal{A}}^k$ and p when $\delta t = 10$ seconds, $t_s = \delta t$ and the number of peers in the network is 60000. Due to lack of space, we include in Appendix B the approximations/formulas that were used to generate the plots in Figure 5.

Location	Number of Hosts
Europe	4
North America	3
Asia Pacific	2
South America	1

Table 2: Geographic location of the Bitcoin nodes that we used in our measurements. We made use of 10 Bitcoin nodes located around the globe. All the hosts were running Ubuntu 11.10 with at least 613 MB of RAM.

Our analysis therefore shows that $\mathcal{A}$ can maximize $P_S^{(2)}$ by increasing the number of peers that receive $TR_{\mathcal{A}}$, $\eta_{\mathcal{A}}^k, \forall t_k$. $\mathcal{A}$ can achieve this: *(i)* by sending $TR_{\mathcal{A}}$ before $TR_{\mathcal{V}}$ and therefore giving $TR_{\mathcal{A}}$ a better advantage in spreading in the network and/or *(ii)* by relying on multiple helpers to spread $TR_{\mathcal{A}}$ faster in the network. In the former case, $\mathcal{A}$ can delay the transmission of $TR_{\mathcal{V}}$ by a maximum of $\Delta t = \delta t_{\mathcal{VH}}^{\mathcal{A}} - \delta t_{\mathcal{AV}}^{\mathcal{V}}$ (cf. Equation 1) after sending $TR_{\mathcal{A}}$ while ensuring that $\mathcal{V}$ first receives $TR_{\mathcal{V}}$. In this way, both Requirements **(1)** and **(2)** can be satisfied.

We denote by $P_S = P_S^{(1)} \cdot P_S^{(2)}$, the probability that the attack succeeds (in satisfying Requirements **(1)** and **(2)**). Here, $P_S^{(1)}$ refers to the probability that Requirement **(1)** is satisfied.

4.4 Experimental Evaluation

We now present the experimental results of double-spending experiments in the Bitcoin network. Our experiments aim at investigating the satisfiability of aforementioned Requirements **(1)** and **(2)**.

Experimental Setup: We adopt the setup described in Section 4.3 in which the attacker $\mathcal{A}$ is equipped with one or more helper nodes $\mathcal{H}$ that help her relay the double-spent transaction. In our experiments, we made use of 10 Bitcoin nodes located around the globe (Table 2); this serves to better assess the different views seen from multiple points in the Bitcoin overlay network and to abstract away the peculiarities that might originate from specific network topologies. Appendix C includes the Bitcoin addresses that we used while performing our experiments.

Conforming with our analysis in Section 4.3, we modified the C++ implementation of Bitcoin client version 0.5.2 as follows:

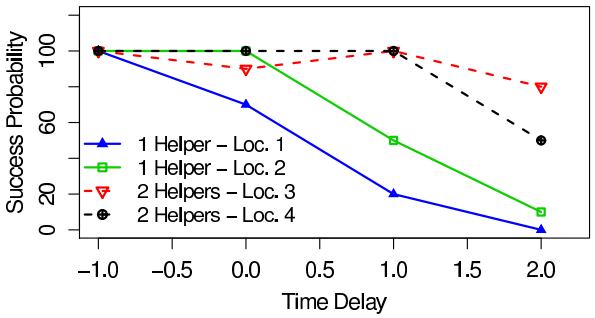


Figure 6: P_S versus Δt when the vendor has 8 connections.

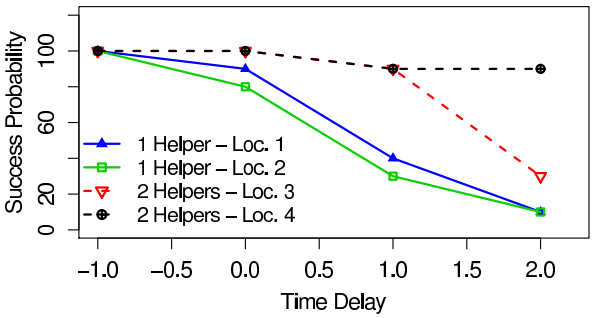


Figure 7: P_S versus Δt when the vendor has 40 connections.

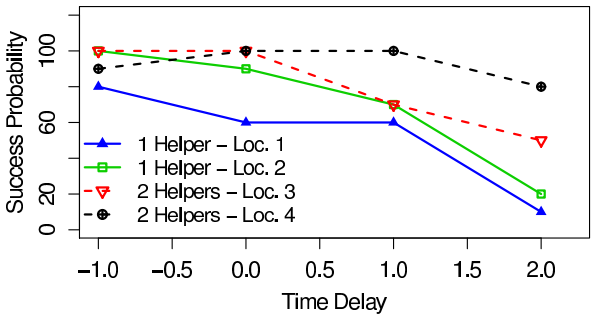


Figure 8: P_S versus Δt when the vendor has 125 connections.

Location	# Helpers	Δt (sec)	P_S
Asia Pacific 1, 125 conn.	2	0	100%
Asia Pacific 2, 125 conn.	2	0	100%
North America 1, 8 conn.	1	0	100%
North America 2, 40 conn.	1	0	90%
Asia Pacific 1, 8 conn.	2	1	100%
Asia Pacific 2, 125 conn.	2	1	100%
North America 1, 40 conn.	1	-1	100%

Figure 9: Summary of Results. Here, "Location" denotes the location of $\mathcal{V}$, "conn" is the number of $\mathcal{V}$'s connections.

- The attacker only connects to the vendor's machine.
- The attacker creates transactions $TR_{\mathcal{V}}$ and $TR_{\mathcal{A}}$ constructed using the same coins. She sends $TR_{\mathcal{V}}$ using the Bitcoin network to the neighboring vendor and $TR_{\mathcal{A}}$ via a direct TCP connection to one or more helper nodes with an initial delay Δt of -1, 0, 1, and 2 seconds. Here, Δt refers to the time delay between the transmission of $TR_{\mathcal{V}}$ and $TR_{\mathcal{A}}$ by $\mathcal{A}$.
- Upon reception of $TR_{\mathcal{A}}$, each helper node broadcasts it in the Bitcoin network.
- $\mathcal{V}$ accepts the payment if it receives $TR_{\mathcal{A}}$.

With this setup, we performed double-spending attempts when the vendors are located in 4 different network locations (2 vendors were in North America and the remaining 2 were in Asia Pacific). In our experiments, $\mathcal{A}$ was located in Europe. However, since $\mathcal{A}$ does not contribute in spreading in the Bitcoin network any transaction herself, her location does not affect the outcome of the attack. That is, the sole role of $\mathcal{A}$ is to send $TR_{\mathcal{V}}$ to $\mathcal{V}$ using a direct connection in the Bitcoin network and $TR_{\mathcal{A}}$ to the helper nodes using a direct TCP connection.

We conduct our experiments with a varying number of connections of the vendor (8, 40 and 125 connections) and by varying the number of helper nodes (1 and 2) chosen randomly from the nodes in Table 2.The helper nodes were connected to at least 125 other Bitcoin peers. Each data point in our measurements corresponds to 10 different measurements, totaling approximately 500 double-spending attempts with a total of 20 BTCs[11]. We point out that since all the hosts in our measurements were using wallets that were under our control, other Bitcoin users were not affected by our measurements.

We also created a Python script that, for each conducted measurement, parses the generated logs along with the Bitcoin block explorer [12] to check whether the Requirements **(1)** and **(2)** described in Section 4.2 are satisfied.

Results: To assess the feasibility of double-spending in fast Bitcoin payments, we evaluate empirically the success probability, P_S, with respect to the number of helper nodes, the number of connections of the vendor and Δt.

[11] We point out that since all the hosts in our measurements were using wallets that were under our control, other Bitcoin users were not affected by our measurements.

双花攻击成功的概率与特殊网络拓扑并无直接关系

Our results, depicted in Figures 6, 7, 8, and 9 show that, irrespective of a specific network topology, the probability that $\mathcal{A}$ succeeds in mounting double-spending attacks is significant.

Confirming our analysis, P_S decreases as Δt increases. As explained in Section 4.3, this is due to the fact that the higher is Δt, the larger is the number of peers that receive $TR_{\mathcal{V}}$; in turn, the probability that $TR_{\mathcal{A}}$ is confirmed before $TR_{\mathcal{V}}$ decreases. As shown in Figures 6, 7, and 8, this can be remedied if the number of helper nodes that spread $TR_{\mathcal{A}}$ increases. Our results show that even for a large Δt of 2 seconds, relying on 2 helper nodes still guarantees that double-spending succeeds with a considerable probability; when $\Delta t = 1$ seconds, the attack is guaranteed to succeed (P_S approaches 1) using 2 helpers. This is summarized in Figure 9.

The number of $\mathcal{V}$'s connections considerably affects P_S especially when $\mathcal{A}$ controls only one helper; in the case where $\mathcal{V}$ has a similar number of connections when compared to the number of connections of the helper, P_S approaches 0.5. This corresponds to the case where both $TR_{\mathcal{A}}$ and $TR_{\mathcal{V}}$ are spread equally in the network (Figure 8). On the other hand, as the connectivity of $\mathcal{V}$ decreases, $TR_{\mathcal{A}}$ spreads faster in the network. This is depicted in Figures 6 and 7. We thoroughly investigate the impact of the connectivity of $\mathcal{V}$ in Section 5.1.

5 Countering Double-Spending in Fast Bitcoin Payments

In this section, we evaluate two strategies that the vendor can adopt, as recommended by Bitcoin developers, to resist double-spending: *using a listening period* and *inserting observers*. Moreover, we propose an effective countermeasure based on the propagation of "alert" messages and we show that it does not require significant modifications on existing clients.

5.1 Using a "Listening Period"

The Bitcoin daemon locally generates an error if it receives a transaction whose inputs have already been spent. However, this error is not displayed to the Bitcoin user. Therefore, Requirements **(1)** and **(2)** can be satisfied (as shown in Section 4), but $\mathcal{A}$ can only trick a vendor who does not thoroughly check the transactions that are processed by its Bitcoin daemon. Such a functionality would require a modification of existing Bitcoin clients.

As advocated in [13], one possible way for $\mathcal{V}$ to detect double-spending attempts is to adopt a listening period, of few seconds, before delivering its service to $\mathcal{A}$; during this period, $\mathcal{V}$ monitors all the transactions it receives, and checks if any of them attempts to double-spend the coins that $\mathcal{V}$ previously received from $\mathcal{A}$. This countermeasure is based on the intuition that since it takes every transaction few seconds[12] to propagate to every node in the Bitcoin network, then it is highly likely that $\mathcal{V}$ would receive both $TR_{\mathcal{V}}$ and $TR_{\mathcal{A}}$ within the listening period (and before granting service to $\mathcal{A}$).

We show that this detection technique can be, however, circumvented by $\mathcal{A}$ as follows. $\mathcal{A}$ can attempt to delay the transmission of $TR_{\mathcal{A}}$ such that $t = (t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}})$ exceeds the listening period while $TR_{\mathcal{A}}$ still has a significant chance of being spread in the network. On one hand, as t increases, the probability that all the immediate neighbors of $\mathcal{V}$ in the Bitcoin P2P network receive $TR_{\mathcal{V}}$ first also increases; when they receive $TR_{\mathcal{A}}$ later on, $TR_{\mathcal{A}}$ will not be added to the memory pool of $\mathcal{V}$'s neighbors and as such $TR_{\mathcal{A}}$ will not be forwarded to $\mathcal{V}$. On the other hand, $\mathcal{A}$ should make sure that $TR_{\mathcal{A}}$ was received by enough peers so that Requirement **(2)** can be satisfied. To that end, $\mathcal{A}$ can increase the number of helpers it controls.

To validate this claim, we conducted experiments using the setup described in Section 4.4. Our experiments aimed at finding triplets $(\Delta t, N_{\mathcal{H}}, C)$, where $N_{\mathcal{H}}$ is the number of helper nodes, and C is the number of $\mathcal{V}$' connections, such that the probability P_D that $\mathcal{V}$ receives $TR_{\mathcal{A}}$ (after having received $TR_{\mathcal{V}}$) is minimized. We illustrate our results in Table 3.

Conforming with our analysis, our findings show that, for $C < 80$, $\exists \Delta t$, such that $P_S \geq P_D$. That is, in the case where $\mathcal{V}$ utilizes a listening period of few seconds to detect double-spending, there are a number of cases in which $\mathcal{A}$ can circumvent this countermeasure and perform a successful double-spending attack without being detected. Note that if P_D is small, $\mathcal{A}$ has incentives to attempt to double-spend even in the case when $P_S \approx 0.1$), since her attempts will be detected with low probability. For a subset of our measurements, we also show that $\exists (\Delta t, N, C)$, such that $t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}} = \infty$. This case corresponds to the event where all the neighbors of $\mathcal{V}$ receive $TR_{\mathcal{V}}$ first (and do not forward $TR_{\mathcal{V}}$ to $\mathcal{V}$). In this case, $P_D = 0$ and $\mathcal{V}$ cannot detect the misbehavior of $\mathcal{A}$, irrespective of the number of double-spending attempts that $\mathcal{A}$ performs.

[12] Our experiments in Section 4.4 show that the average time for a peer to receive both $TR_{\mathcal{A}}$ and $TR_{\mathcal{V}}$ is approximately 3.354 seconds.

	P_S	P_D	$t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}}$ (sec)	% Observed
Europe, 8 Connections, 3 Helpers, $\Delta t = 2.00$	10%	10%	8.664	53%
Europe, 8 Connections, 3 Helpers, $\Delta t = 2.25$	10%	10%*	5.65	47%
South America, 8 Connections, 2 Helpers, $\Delta t = 2.5$	20%	6.66%*	3.749	62%
South America, 8 Connections, 3 Helpers, $\Delta t = 2.5$	7.7%	0%	∞	53%
South America, 8 Connections, 4 Helpers, $\Delta t = 3.0$	13.33%	0%	∞	57%
Asia Pacific, 8 Connections, 3 Helpers, $\Delta t = 2.75$	10%	0%	∞	57%
Asia Pacific, 8 Connections, 2 Helpers, $\Delta t = 1.75$	55%	20%*	5.5	91%
Asia Pacific, 8 Connections, 3 Helpers, $\Delta t = 2.75$	5%	0%*	∞	66%
North America, 20 Connections, 3 Helpers, $\Delta t = 2.75$	5%	0%	∞	47%
North America, 20 Connections, 5 Helpers, $\Delta t = 3.00$	11%	11%	3.208	46%
North America, 20 Connections, 5 Helpers, $\Delta t = 1.75$	10%	10%	5.76	74%
North America, 20 Connections, 1 Helper, $\Delta t = 1.25$	30%	30%*	3.34	78%
North America, 20 Connections, 4 Helpers, $\Delta t = 2.00$	82%	63%	2.85	78%
North America, 20 Connections, 2 Helpers, $\Delta t = 2.00$	20%	20%*	4.79	60%
North America, 20 Connections, 1 Helper, $\Delta t = 1.50$	40%	30%*	3.51	60%
Europe, 20 Connections, 3 Helpers, $\Delta t = 1.0$	45%	45%*	3.844	87%
Europe, 30 Connections, 1 Helper, $\Delta t = 1.5$	15%	10%*	3.412	42%
Asia Pacific, 40 Connections, 1 Helper, $\Delta t = 2.9$	10%	10%*	4.946	42%
Europe, 40 Connections, 1 Helper, $\Delta t = 1.25$	10%	10%	1.841	36%
Europe, 40 Connections, 2 Helpers, $\Delta t = 1.5$	20%	20%*	3.075	36%
South America, 40 Connections, 1 Helper, $\Delta t = 2.0$	30%	40%	3.217	57%
South America, 60 Connections, 1 Helper, $\Delta t = 2.5$	6.67%	20%*	3.999	41%
South America, 60 Connections, 2 Helpers, $\Delta t = 2.75$	6.67%	13.33%	4.157	28%
Asia Pacific, 60 Connections, 1 Helper, $\Delta t = 3.00$	10%	0%*	∞	20%
Asia Pacific, 60 Connections, 2 Helpers, $\Delta t = 2.75$	30%	50%	3.992	50%
Asia Pacific, 80 Connections, 1 Helper, $\Delta t = 3.7$	10%	20%	5.04	18%
Europe, 80 Connections, 1 Helper, $\Delta t = 2.25$	33.33%	66.67%	3.648	61%
Europe, 80 Connections, 1 Helper, $\Delta t = 2.75$	13.33%	26.67%	5.093	28%
North America, 100 Connections, 3 Helpers, $\Delta t = 1.0$	60%	80%*	2.25	88%
Asia Pacific, 100 Connections, 1 Helper, $\Delta t = 1.75$	44.4%	66.66%	2.871	82%
Asia Pacific, 100 Connections, 1 Helper, $\Delta t = 1.5$	80%	80%	2.807	88%

Table 3: Monitoring received transactions at $\mathcal{V}$ (Section 5.1) and its observers (Section 5.2) to detect double-spending. Each data point corresponds to 20 measurements. The helpers used in these experiments had between 125 and 400 connections. "P_D" denotes the probability that $\mathcal{V}$ receives $TR_{\mathcal{A}}$. "% Observed" refers to the fraction of observers (among 5) that received $TR_{\mathcal{A}}$. $t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}}$ refers to the average of the time it takes $\mathcal{V}$ to receive $TR_{\mathcal{A}}$ after having received $TR_{\mathcal{V}}$, for those cases where $\mathcal{V}$ receives $TR_{\mathcal{A}}$ (P_D). * refers to the case where $TR_{\mathcal{A}}$ was received after 15 seconds (beyond the listening period); otherwise, the absence of * shows that $TR_{\mathcal{A}}$ was not received.

The Connectivity of $\mathcal{V}$ as a Security Parameter: As shown from our results in Table 3, the fewer connections of $\mathcal{V}$, the more likely is that *all* the neighbors of $\mathcal{V}$ receive $TR_{\mathcal{V}}$ before $TR_{\mathcal{A}}$ and thus that $\mathcal{V}$ does not receive $TR_{\mathcal{A}}$. Similarly, as the number of connections of $\mathcal{V}$ increases, the effort (i.e., the number of helpers, the amount of delay) that $\mathcal{A}$ needs to invest to ensure that none of $\mathcal{V}$'s neighbors receive $TR_{\mathcal{A}}$ becomes considerable. This is depicted in Figure 10; when $\mathcal{V}$ adopts a listening period of at least 15 seconds, P_D increases as the number of $\mathcal{V}$'s connections increases. When $\mathcal{V}$ has more than 100 connections, the probability that it does not receive $TR_{\mathcal{A}}$ is negligible; in this case, using a listening period can be indeed effective to detect double-spending.

However, since the connectivity of Bitcoin peers largely varies with the network churn[13] (Figure 4), $\mathcal{V}$ needs to constantly monitor its connection count to ensure that it does not drop below a threshold so that it can detect double-spending.

In Section 5.3, we propose a countermeasure that effectively detects double-spending even in the case when the number of the connections of $\mathcal{V}$ is small.

[13]For example, some of the nodes used in our experiments could not acquire more than 40 connections over a period of few days, due to the fact that these nodes were often restarted. In such cases, other Bitcoin peers will assign a low priority to connecting to these nodes.

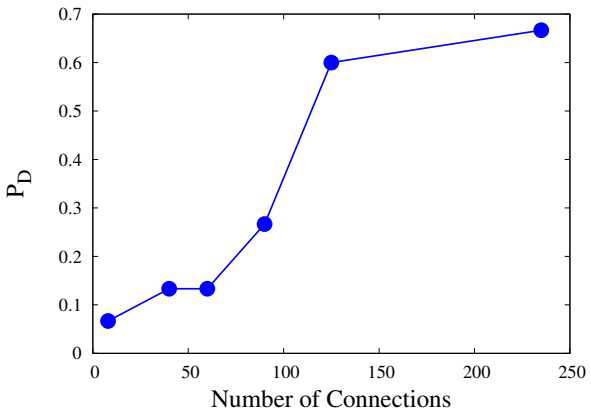


Figure 10: The connectivity of $\mathcal{V}$ as a security parameter. We measure P_D with respect to the number of connections of $\mathcal{V}$. Here, we consider a setting where $\mathcal{V}$ is located in Asia Pacific, $\Delta t = 3.0$ seconds and $\mathcal{A}$ controls 4 helpers. As the number of connections of $\mathcal{V}$ increases, the probability that $\mathcal{V}$ receives both $TR_{\mathcal{A}}$ and $TR_{\mathcal{V}}$ also increases.

5.2 Inserting Observers in the Network

Another possible technique that naturally extends the aforementioned proposal based on the adoption of listening period would be for $\mathcal{V}$ to insert a node that it controls within the Bitcoin network—an "observer"—that would directly relay to $\mathcal{V}$ all the transactions that it receives. In this case, $\mathcal{V}$ can be aware within few seconds of a double-spending attempt if either it or its observer receive $TR_{\mathcal{A}}$.

We evaluated this technique using up to 5 observers. These observers were chosen from the hosts listed in Table 2. Our findings in Table 3 show that this method can indeed help detecting double-spending as all double-spent transactions were received by at least one observer within few seconds. However, given that $\mathcal{A}$ delays the transmission of $TR_{\mathcal{A}}$, our results show that only a subset of the observers receive $TR_{\mathcal{A}}$. As mentioned previously, this corresponds to the case where all the neighbors of these observers have received $TR_{\mathcal{V}}$ first and as such they will not forward $TR_{\mathcal{A}}$ back to the observers.

Therefore, $\mathcal{V}$ needs to employ a considerable number of observers (≈ 3) (that connect to a large number of Bitcoin peers) to ensure that at least one observer detects any double-spending attempt; this, however, comes at the expense of additional costs for $\mathcal{V}$ to maintain the observers in the network.

5.3 Communicating Double-Spending Alerts Among Peers

Given our findings, an efficient countermeasure to combat double-spending on fast Bitcoin payments would be for Bitcoin peers to propagate alerts whenever they receive two or more transactions that share common inputs and different outputs (cf. Figure 2). This countermeasure extends the solutions outline in the previous sections. However, unlike the aforementioned solutions, double-spending alerts *(i)* cannot be evaded by $\mathcal{A}$ and *(ii)* do not incur additional costs on $\mathcal{V}$.

The main intuition behind this solution is that while $\mathcal{A}$ might be able to prevent $\mathcal{V}$ and a subset of $\mathcal{V}$'s observers from receiving $TR_{\mathcal{A}}$, a considerable number of Bitcoin peers receive both $TR_{\mathcal{A}}$ and $TR_{\mathcal{V}}$. If the majority of these peers are honest[14], they can immediately alert the entire network by broadcasting an `alert` that includes both $TR_{\mathcal{A}}$ and $TR_{\mathcal{V}}$ as a proof of double-spending to all network peers. Given the fast broadcast medium in Bitcoin, `alert` broadcasts would eventually reach $\mathcal{V}$ within few seconds; the double-spending of $\mathcal{A}$ can be therefore detected before $\mathcal{A}$ actually receives the service from $\mathcal{V}$.

We emphasize that similar `alert` mechanisms already exist in the current implementation of Bitcoin, but were devised for other purposes and are currently unused[15]. As such, this technique does not require any significant modification in the Bitcoin client. Furthermore, we argue that this solution can easily combat false claims; honest Bitcoin peers will only forward `alert` messages if $TR_{\mathcal{A}}$ and $TR_{\mathcal{V}}$ are indeed correctly formed, share common inputs and different outputs and are equipped with legitimate signatures.

6 Related Work

First introduced in 2009, Bitcoin has recently attracted the attention of the research community. In [20], Elias investigates the legal aspects of privacy in Bitcoin. Reid and Harrigan [35] explore user anonymity limits in Bitcoin. In [24], Babaioff *et al.* address the lack of incentives for Bitcoin peers to include recently announced transactions in a block. In [19], Syed *et al.* propose a user-friendly technique for managing Bitcoin wallets.

Finney [15] describes a double-spending attack in Bitcoin where the attacker includes in her generated

[14]This is the underlying assumption that ensures the correct operation of Bitcoin.

[15]Bitcoin plans to use "alert" messages to broadcast information signed by the private key of Bitcoin's founder.

blocks transactions that transfer some coins between her own addresses; these blocks are only released to the network after the attacker double-spends the same coins using fast payments and acquires a given service. In its official documentation [6, 13], Bitcoin acknowledged that double-spending might occur when dealing with fast payments but still encouraged its users to "sell things without waiting for a confirmation as long as the transaction is not of high value" [6]. Bitcoin developers claim in [6] that the cost of mounting double-spending attacks for low-cost transactions is comparatively high and that vendors could adopt a listening period for few seconds before delivering a product to counter double-spending attacks. In this work, we thoroughly investigate double-spending attacks in Bitcoin. Namely, we show that *(i)* double-spending fast payments in Bitcoin can be performed without incurring costs on the attacker and *(ii)* the countermeasures recommended by Bitcoin to alleviate double-spending can be circumvented.

Double-spending in online payments has received considerable attention in the literature [23, 31]. In Credit-Card based payments, fairness is achieved through the existence of a bank (e.g., [26,33]) or another trusted intermediary (e.g., PayPal [30], MoneyBookers [16]). Here, the intermediaries are trusted *(i)* to verify that the client has not already spent the funds he/she is paying the vendor with, and *(ii)* to reverse a charge, if the vendor has misbehaved. Micropayments [22, 32, 34, 36] is an efficient payment scheme aiming primarily at enabling low-cost transactions. Here, the payer provides signed endorsements of monetary transfers on the vendor's name. Digital signatures in these systems constitute the main double-spending resistance mechanism. ECash [27–29] is another form of digital cash which supports payer anonymity. ECash offers strong accountability guarantees by relying on a set of cryptographic primitives that ensure that when a user double-spends a coin, his/her identity is revealed.

Similar to Bitcoin, a number of decentralized payment systems [25,37] were designed to resist double-spending attacks. In [37], Yang and Garcia-Molina introduce a P2P payment system, in which the first owner of an electronic coin authorizes that coin's transfer among other peers in the network. Thus, the generator of the coin is responsible for preventing double-spending. In [25], Belenkiy *et al.*, introduce an ECash-based P2P payment scheme that provides accountability at the cost of privacy.

7 Concluding Remarks

Given its design, Bitcoin can only ensure the security of payments when a payment verification time of few tens of minutes can be tolerated. Moreover, our results show that the verification time of payments exhibits a large variance as it depends on the confirmation of transactions in blocks—which follows a shifted geometric distribution. This slow payment verification is clearly inappropriate for fast payments; it is, however, essential for the detection of double-spending attacks.

In this paper, we addressed the double-spending resilience of Bitcoin in fast payments, in which the time to acquire a service is in the order of few seconds. More specifically, we showed that not only these attacks succeed with overwhelming probability, but also that, contrary to common beliefs, they do not incur any significant overhead on the attacker. For that purpose, we analyzed the conditions for performing successful double-spending attacks against fast payments in Bitcoin and we experimentally confirmed our analysis. As far as we are aware, our experiments constitute the first comprehensive double-spending measurements in Bitcoin. It is noteworthy that we have performed thousands of double-spending attempts using fixed Bitcoin addresses without having to bear any type of penalty.

Finally, we explored the solution space for securing Bitcoin against double-spending attacks. Our findings show that the measures recommended by Bitcoin developers for fast transactions are not always effective in resisting double-spending. By leveraging on our results, we propose a lightweight measure that would enable the secure and albeit verification of Bitcoin transactions.

Given that the vulnerability of existing clients to double-spending might severely harm the growth of Bitcoin, and impact its financial and economic standing, we argue that the integration of double-spending countermeasures in the current implementation of Bitcoin emerges as a necessity. As we show in this work, the propagation of double-spending alerts in the network would constitute a first important step towards efficiently detecting double-spending.

References

[1] Bitcoin – Wikipedia, Available from `https://en.bitcoin.it/wiki/Introduction`.

[2] Trade - Bitcoin, Available from `https://en.bitcoin.it/wiki/Trade`.

[3] Bitcoin Charts, Available from http://bitcoincharts.com/.

[4] Bitcoin ATM, Available from http://bitcoinatm.com/.

[5] CNN: Bitcoin's uncertain future as currency, Available from http://www.youtube.com/watch?v=75VaRGdzMM0.

[6] FAQ - Bitcoin, Available from https://en.bitcoin.it/wiki/FAQ.

[7] Bitcoin Block 80000, Available from http://blockexplorer.com/b/80000.

[8] Protocol Rules – Bitcoin, Available from https://en.bitcoin.it/wiki/Protocol_rules.

[9] Protocol Specifications – Bitcoin, Available from https://en.bitcoin.it/wiki/Protocol_specification.

[10] Difficulty – Bitcoin, Available from https://en.bitcoin.it/wiki/Difficulty.

[11] Block hashing algorithm – Bitcoin, Availabe from https://en.bitcoin.it/wiki/Block_hashing_algorithm.

[12] Bitcoin Block Explorer, Available from http://blockexplorer.com/.

[13] Myths - Bitcoin, Available from https://en.bitcoin.it/wiki/Myths#Point_of_sale_with_bitcoins_isn.27t_possible_because_of_the_10_minute_wait_for_confirmation.

[14] Satoshi Client Node Connectivity, Available from https://en.bitcoin.it/wiki/Satoshi_Client_Node_Connectivity.

[15] The Finney Attack, Available from https://en.bitcoin.it/wiki/Weaknesses#The_.22Finney.22_attack.

[16] MoneyBookers, Available from https://www.moneybookers.com/app/.

[17] Comparison of Mining Pools, Available from https://en.bitcoin.it/wiki/Comparison_of_mining_pools.

[18] Comparison of Mining Hardware, Available from https://en.bitcoin.it/wiki/Mining_hardware_comparison.

[19] Bitcoin Gateway, A Peer-to-peer Bitcoin Vault and Payment Network, 2011. Available from http://arimaa.com/bitcoin/.

[20] Bitcoin: Tempering the Digital Ring of Gyges or Implausible Pecuniary Privacy, 2011. Available from http://ssrn.com/abstract=1937769ordoi:10.2139/ssrn.1937769.

[21] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System.

[22] Androulaki, E., Raykova, M., Stavrou, A., and Bellovin, S. M. PAR: Payment for Anonymous Routing. In *Proceedings of the 8th Privacy Enhancing Technologies Symposium* (2008).

[23] Asokan, N., Janson, P., Steiner, M., and Waidner, M. State of the Art in Electronic Payment Systems. *IEEE Computer* (1999).

[24] Babaioff, M., Dobzinski, S., Oren, S., and Zohar, A. On Bitcoin and Red Balloons. *CoRR* (2011).

[25] Belenkiy, M., Chase, M., Erway, C., Jannotti, J., Küpçü, A., Lysyanskaya, A., and Rachlin, E. Making P2P Accountable without Losing Privacy. In *Proceedings of WPES* (2007).

[26] Bellare, M., Garay, J., Hauser, R., Krawczyk, H., Steiner, M., Herzberg, A., Tsudik, G., van Herreweghen, E., and Waidner, M. Design, Implementation and Deployment of the iKP Secure Electronic Payment System. *IEEE Journal on Selected Areas in Communications* (2000).

[27] Brands, S. Electronic Cash on the Internet. In *Proceedings of the Symposium on the Network and Distributed System Security* (1995).

[28] Camenisch, J., Hohenberger, S., and Lysyanskaya, A. Compact E-Cash. In *Proceedings of Advances in Cryptology - EUROCRYPT* (2005).

[29] Chaum, D., Fiat, A., and Naor, M. Untraceable electronic cash. In *Proceedings on Advances in Cryptology - CRYPTO* (1990).

[30] Company, P. Paypal, Available from http://www.paypal.com.

[31] Everaere, P., Simplot-Ryl, I., and Traore, I. Double Spending Protection for E-Cash Based on Risk Management. In *Proceedings of Information Security Conference* (2010).

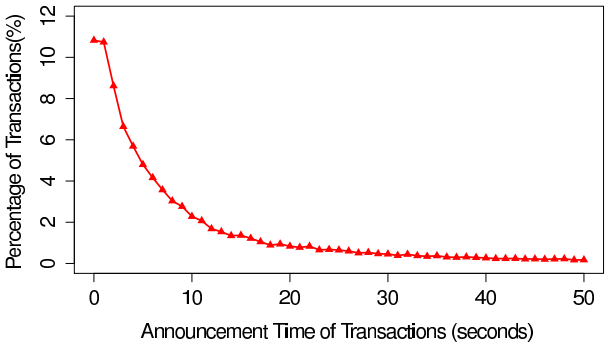


Figure 11: Distribution of time intervals between announcements of successive transactions. To measure the transaction announcement times, we parsed the block chain of Bitcoin and we counted the number of transactions in each block. To generate the plot, we assumed that transactions are announced uniformly at random within two successive block generations.

[32] Karame, G., Francillon, A., and Čapkun, S. Pay as you Browse: Microcomputations as Micropayments in Web-based Services. In *Proceedings of the International World Wide Web Conference (WWW)* (2011).

[33] Krawczyk, H. Blinding of Credit Card Numbers in the SET Protocol. In *Proceedings of the International Conference on Financial Cryptography* (1999).

[34] Micali, S., and Rivest, R. L. Micropayments Revisited. In *Proceedings of CT-RSA* (2002).

[35] Reid, F., and Harrigan, M. An Analysis of Anonymity in the Bitcoin System. *CoRR* (2011).

[36] Rivest, R. Peppercoin Micropayments. In *Proceedings of Financial Cryptography* (2004).

[37] Yang, B., and Garcia-Molina, H. PPay: micropayments for peer-to-peer systems. In *Proceedings of the ACM Conference on Computer and Communication Security* (2003).

A Block Generation in Bitcoin

In what follows, we analyze the distribution of block generation times in Bitcoin.

To generate a block, peers *work* on solving a PoW problem. In particular, given the set of transactions that have been announced since the last block's generation, and the hash of the last block, Bitcoin peers need to find a nonce that would make the SHA-256 hash of the formed block smaller than a 256-bit number, referred to as *target*. That is:

$$\texttt{SHA-256}\{\mathsf{Bl}_l \mid\mid \mathrm{MR}(\mathrm{TR}_1, \ldots, \mathrm{TR}_n) \mid\mid No\} \leq \textit{target}, \quad (4)$$

where,

- Bl_l denotes the last generated block,
- $\mathrm{MR}(\overline{x})$ denotes the root of the Merkle tree with elements $\overline{x}$,
- $\mathrm{TR}_1 \mid\mid \ldots \mid\mid \mathrm{TR}_n$ is a set of transactions which have been announced (and not yet confirmed) since Bl_l's generation,
- No is the 32-bit nonce,
- *target* is a 256-bit number which that sets the difficulty of the PoW. It is updated according to the generation times of the last 2016 blocks.

To solve the PoW, each peer chooses a particular subset of the candidate solutions' space and performs brute-force search. It is apparent that the bigger *target* is, the easier is to find a nonce that satisfies the PoW. For the purpose of our analysis, we note the following:

1. The probability of success in a single nonce-trial is negligible. Taking in consideration that SHA-256 maps its arguments randomly to the target space, each of the 2^{32} nonces has $\frac{\textit{target}}{max(\textit{target})}$ probability of satisfying the PoW, where, *target* is the number of targets which are less or equal than *target* and $max(\textit{target}) = 2^{256} - 1$ is the number of all possible targets[16].
2. Peers compute their PoW independently of each other; as such, the probability that one of them succeeds does not depend on the progress of PoW of the others.
3. Peers are frequently required to restart their PoW. In particular, whenever a new transaction is added to the memory pool of a peer, the Merkle root (included in the block) changes; therefore, the effort in constructing the PoW is "reset". This also applies to the case when new blocks are generated; the "previous block" field needs to be updated (cf. Figure 1). For the sake

[16] Note that *target* is equal to the number of 256-bit numbers which are less or equal to it; as such, $max(\textit{target})$ is equal to $2^{256} - 1$, which is the number of all possible targets (target is never equal to zero). The inverse of this probability is known as *difficulty* in Bitcoin block generation.

of our analysis, we approximate the time interval between the announcement of successive transactions as follows. We extract the various block generation times from the Bitcoin block explorer (cf. Figure 1) and we assume that transactions are announced uniformly at random within two successive block generations. Our findings (Figure 11) show that the time interval between the announcement of most pairs of successive transactions is below 15 seconds. Therefore, we assume in the sequel that the PoW for block generation is "reset" approximately every dt $\approx$ 15 seconds[17].

Given the first two observations, the probability of an average peer in succeeding in an *individual* block generation attempt can be modeled as an independent Bernoulli process with success probability $\varepsilon = \frac{target}{max(target)}$.

Given this, and based on the last observation, we claim that *consecutive* block generation attempts can be modeled as sequential Bernoulli trials with *replacement*. Our claim for replacement is justified by the fact that maximum possible PoW progress performed by a peer (expressed as a number of hash calculations) before its PoW resets, is negligible in comparison to $max(target)$. This is the case since the PoW progress approximates $2^{35} \ll max(target)$ given the computing power of most Bitcoin peers [17, 18]. This also means that ε is always equal to $\frac{target}{max(target)}$.

Let n_i refer to the number of attempts that a peer m_i performs within a time period δt. Typically, δt is in the order of few minutes. the probability p_i of m_i finding at least one correct PoW within these trials is given by $p_i = 1 - (1-\varepsilon)^{n_i}$. Since ε and n_i are small, p_i can be approximated to:

$$p_i = 1 - (1-\varepsilon)^{n_i} \approx n_i \varepsilon.$$

Therefore, the set of trials of m_i within δt can be unified to constitute a single Bernoulli process with success probability $n_i\varepsilon$.

Assuming that there are ℓ peers, $\mathsf{m}_i,\ i = 1 \dots \ell$ with success probability $p_i,\ i = 1 \dots \ell$ respectively, the overall probability of success in block generation can be approximated to:

$$\mathsf{p} \approx 1 - \prod_{i=1}^{\ell} (1 - p_i).$$

[17]While there are other parameters that can affect the "reset" of PoW, we believe that transaction announcements causes the most frequent PoW "resets" at honest Bitcoin peers.

In the simplified case when the peers have equal computing power, $p_i = p, i = 1 \dots \ell$, the previous probability reduces to:

$$\mathsf{p} = 1 - (1-p)^{\ell} \approx \ell \cdot p,$$

since $p\ell < 1$.

Let time be divided into small intervals $(t_0, t_1], \dots, (t_{n-1}, t_n]$ of equal size δt, where $t_0 = 0$ denotes the time when the last block was generated. Here, each peer can make up to n_i trials for block generation within each interval. Let the random variable X_k denote the event of success in $\delta t = \{t_k, t_{k+1}\}$. That is,

$$X_k = \begin{cases} 1 & \text{if a block is created within} \{t_k, t_{k+1}\}, \\ 0 & \text{otherwise.} \end{cases}$$

It is evident that:

$$\mathsf{Prob}(X_k = 1) = \mathsf{p}.$$

Conceivably, after a success in block generation, peers stop mining for that particular block. We denote the number of attempts until a success is achieved by another random variable $\mathcal{Y}$.

$$\begin{aligned} \mathsf{Prob}(\mathcal{Y} = k) &= \mathsf{Prob}(X_k = 1) \prod_{i=1}^{k-1} \mathsf{Prob}(X_i = 0) \\ &= \mathsf{p}(1-\mathsf{p})^{k-1}. \end{aligned}$$

Assuming a constant rate of trials per time window δt, the number of failures until a success is observed in block generation is proportional to the time it takes for a block to be generated. More specifically, let $\mathcal{T}$ denote the time period till a block is generated. Then, the following holds:

$$\mathsf{Prob}(\mathcal{T} = k \cdot \delta t) = \mathsf{Prob}(\mathcal{Y} = k) \ = \ \mathsf{p}(1-\mathsf{p})^{k-1}.$$

Given this, we conclude that the distribution of block generation times can be modeled with a shifted geometric distribution with parameter p. In Figure 1, we confirm this analysis and we show that (experimental) block generation times in Bitcoin, as reported in [12], can be fitted to a shifted geometric distribution with $p = 0.19$. For the purpose of our experiments, we considered δt to be 2 minutes.

B Formulas for Deriving the Plot in Figure 5

In Equation 3, we derived the probability $\mathrm{P_S}^{(2)}$. In what follows, we show how to estimate Equation 3 to produce the plots in Figure 5.

For the purpose of this analysis, we assume that $t_s = \delta t$ and that $\delta t = 10$ seconds. We can therefore rewrite Equation 3 as follows:

$$\mathrm{P_S} = \mathrm{Prob}(t_{g_\mathcal{A}} < t_{g_\mathcal{V}}) + \tfrac{1}{2} Prob(t_{g_\mathcal{A}} = t_{g_\mathcal{V}}).$$

$$\mathrm{Prob}(t_{g_\mathcal{A}} < t_{g_\mathcal{V}}) =$$
$$\eta^0_\mathcal{A} p(1-\eta^0_\mathcal{V} p) + \eta^1_\mathcal{A} p(1-\eta^0_\mathcal{A} p)(1-\eta^0_\mathcal{V} p)(1-\eta^1_\mathcal{V} p)$$
$$\sum_{g_\mathcal{A}=2}^{\infty} \eta^1_\mathcal{A} p(1-\eta^0_\mathcal{A} p)(1-\eta^1_\mathcal{A} p)^{(g_\mathcal{A}-2)} \cdot (1-\eta^0_\mathcal{V} p)(1-\eta^1_\mathcal{V} p)^{(g_\mathcal{A}-1)}$$

$$\mathrm{Prob}(t_{g_\mathcal{A}} = t_{g_\mathcal{V}}) = \eta^0_\mathcal{V} \eta^0_\mathcal{A} p^2 +$$
$$\sum_{g_\mathcal{A}=1}^{\infty} \eta^1_\mathcal{V} \eta^1_\mathcal{A} p^2 (1-\eta^0_\mathcal{V} p) \cdot (1-\eta^0_\mathcal{A} p)[(1-\eta^1_\mathcal{V} p) \cdot (1-\eta^1_\mathcal{A} p]^{(g_\mathcal{A}-1)}$$

C Bitcoin Addresses Used in Our Experiments

Throughout our experiments, we used the following Bitcoin addresses.

- **Merchant's addresses:**
  - 1Lr3SYuKnjdQA9fxMghd7sYwgWR4Q4ANxi.
  - 15SPb6DZZxVBagNHEQMz6DQVPTggiYknU9.
  - 1AhvENkVCus9XT835VjoCfHAghbigyF5XZ.
- **Attacker's addresses:**
  - 1C4x2tkYezM2keLVdSMo77ZLuvPiJ4MfdV.
  - 1Bkcc27HRur9SMdMPHCvUD8ecj9r1v5qkC.

The complete log of the exchanged transactions can be obtained from the Bitcoin Block Explorer [12] by searching for these addresses.