On the Privacy Provisions of Bloom Filters in Lightweight Bitcoin clients

Arthur Gervais[†], Ghassan O. Karame[‡], Damian Gruber[⌟], Srdjan Capkun[†]
[†⌟]ETH Zurich, Switzerland, [‡]NEC Laboratories Europe, Germany

[†]firstname.lastname@inf.ethz.ch, [‡]firstname.lastname@neclab.eu, [⌟]gruberda@student.ethz.ch

ABSTRACT

Lightweight Bitcoin clients are gaining increasing adoption among Bitcoin users, owing to their reduced resource and bandwidth consumption. These clients support a simplified payment verification (SPV) mode as they are only required to download and verify a part of the block chain—thus supporting the usage of Bitcoin on constrained devices, such as smartphones. SPV clients rely on Bloom filters to receive transactions that are relevant to their local wallet. These filters embed all the Bitcoin addresses used by the SPV clients, and are outsourced to more powerful Bitcoin nodes which then only forward to those clients transactions relevant to their outsourced Bloom filters.

In this paper, we explore the privacy of existing SPV clients. We show analytically and empirically that the reliance on Bloom filters within existing SPV clients leaks considerable information about the addresses of Bitcoin users. Our results show that an SPV client who uses a modest number of Bitcoin addresses (e.g., < 20) risks revealing almost all of his addresses. We also show that this information leakage is further exacerbated when users restart their SPV clients and/or when the adversary has access to more than one Bloom filter pertaining to the same SPV client. Motivated by these findings, we propose an efficient countermeasure to enhance the privacy of users which rely on SPV clients; our proposal can be directly integrated within existing SPV client implementations.

1. INTRODUCTION

Bitcoin has already witnessed a wider adoption and attention than any other digital currency proposed to date. Bitcoin implements a distributed time-stamping service [28], which operates on top of the Bitcoin Peer-to-Peer (P2P) network and ensures that all transactions and their order of execution are visible to all Bitcoin users. Transactions are included in Bitcoin *blocks* that are broadcasted in the entire network; blocks link to each other, thus forming the Bitcoin block chain.

Currently, a typical Bitcoin installation requires more than 18 GB of disk space, and requires considerable time to download and locally index blocks and transactions that are contained in the block chain. Given its increasing use, the transactional volume in Bitcoin is only expected to increase—thus yielding a considerable growth in the size of the block chain [1]. In addition to disk space usage, the continuous growth of Bitcoin's transactional volume incurs considerable overhead on the Bitcoin clients that need to verify the correctness of broadcasted blocks and transactions in the network. This problem becomes even more evident when users wish to perform/verify Bitcoin payments using resource-constraint devices, such as mobile devices.

To remedy this, the Bitcoin developers released a lightweight client, BitcoinJ [2], which supports a simplified payment verification (SPV) mode where only a small part of the block chain is downloaded— thus supporting the typical usage of Bitcoin on constrained devices (e.g., smartphones, cheap virtual private servers). SPV clients were originally proposed by Nakamoto in [28] and were later extended to rely on Bloom filters [20] in order to receive transactions that are relevant to their local wallet. These Bloom filters embed all the addresses used by the SPV clients, and are outsourced to more powerful Bitcoin nodes; these nodes will then forward to the SPV clients those transactions relevant to their Bloom filters.

Bloom filters can be defined with a target false positive rate; by appropriately setting the target false positive rate of Bloom filters, Bitcoin developers aim to provide a suitable anonymity set to hide the addresses of SPV clients. As far as we are aware, the information leakage associated with the reliance on Bloom filters has not been yet thoroughly analyzed in the context of Bitcoin [3].

In this paper, we address this problem, and explore the privacy provisions due to the use of Bloom filters in existing SPV client implementations. We show analytically and empirically that the current integration of Bloom filters within Bitcoin leaks considerable information about the addresses of Bitcoin users. More specifically, we show that the information leakage due to Bloom filters significantly depends on the number of addresses that each user possesses; notably, users who have a modest number of addresses (< 20) risk leaking all of their addresses by embedding them in a Bloom filter. This information leakage is further exacerbated when nodes restart their client, or generate additional Bitcoin addresses to their SPV clients; in these cases, the SPV clients re-compute new Bloom filters. We show that the computation of new Bloom filters considerably reduces the privacy of SPV clients.

Our work therefore motivates a careful assessment of the current implementation of SPV clients, prior to any large-scale deployment. In this work, we make the following contributions:

- We show that considerable information about users who possess a modest number of Bitcoin addresses (e.g., < 20) is leaked by a single Bloom filter in existing SPV clients.
- We show that an adversary can easily link different Bloom filters which embed the same elements—irrespective of the target false positive rate. This also enables the adversary to link, with high confidence, different Bloom filters which pertain to the same originator.
- We show that a considerable number of the addresses of users are leaked if the adversary can collect at least two Bloom filters issued by the same SPV client—irrespective of the target false positive rate and of the number of user addresses.
- Finally, we propose a lightweight and efficient countermeasure to enhance the privacy offered by SPV clients. Our

countermeasure can be integrated with minimum modifications within existing SPV client implementations.

The remainder of this paper is organized as follows. In Section 2, we briefly overview the operation of SPV clients. In Section 3, we introduce our system and attacker model. In Section 4, we analyze the privacy provisions of existing SPV client implementations when the adversary captures a single Bloom filter of an SPV client. In Section 5, we discuss the information leakage when the adversary can acquire multiple Bloom filters of an SPV node. In Section 6, we propose an efficient solution to enhance the privacy of users in SPV clients. In Section 7, we review related work in the area and we conclude the paper in Section 8.

2. BACKGROUND

In what follows, we briefly overview Bitcoin and introduce SPV clients.

Bitcoin enables its users to perform payments by issuing transactions. Standard transactions transfer Bitcoins (BTC) from one or several input addresses to at least one output address. A Bitcoin address corresponds to a public key, whose corresponding secret key enables the address owner to spend the BTCs stored in the respective address. In order to spend BTCs, a user first creates a transaction, which typically takes as inputs the outputs of earlier transactions addressed to his addresses, and specifies as outputs the Bitcoin addresses (or their corresponding public keys) which will collect the resulting BTCs. Finally, the user signs the transaction and broadcasts it into the Bitcoin peer-to-peer (P2P) network.

Transactions are included and stored into Bitcoin blocks. Blocks are generated (or mined) by solving a hash-based proof-of-work (PoW) scheme. More specifically, miners need to find the appropriate block parameters (e.g., a nonce value) such that the resulting block hash is below a given target value. Once such a block is found, the miner broadcasts the block in the network enabling all Bitcoin peers to verify the correctness of the included transactions and the PoW. If the block is deemed correct, the miner is awarded a fixed amount of BTCs. Note that every generated block points to the previous block, thus growing the block chain.

Bitcoin requires all peers in the system to verify all broadcasted transactions and blocks. Clearly, this comes at the expense of storage and computational overhead. Currently, a typical Bitcoin installation requires more than 18 GB of disk space, and considerable time to download and locally index blocks and transactions that are contained in the block chain. Additionally, the continuous growth of Bitcoin transactional volume incurs significant computational overhead on the Bitcoin clients, when verifying the correctness of broadcasted blocks and transactions in the network. This problem becomes even more evident when users wish to perform/verify Bitcoin payments from resource-constrained devices such as mobile devices, tablets, etc.

To remedy that, lightweight client implementations (or simplified payment verification, SPV) have been proposed in [28]; SPV clients do not store the entire block chain, nor do they validate all transactions in the system. Notably, SPV clients only perform a limited amount of verifications, such as the verification of block difficulty and the presence of a transaction in the Merkle tree, etc., and offload the verification of all transactions and blocks to the full Bitcoin nodes. Note that in order to calculate their own balance, SPV clients request full blocks from a given block height on; here, the full Bitcoin nodes can also provide "filtered blocks" to the SPV client that only contain relevant transactions from each block.

Unlike full Bitcoin nodes, SPV clients do not receive all the transactions that are broadcasted within the Bitcoin P2P network,

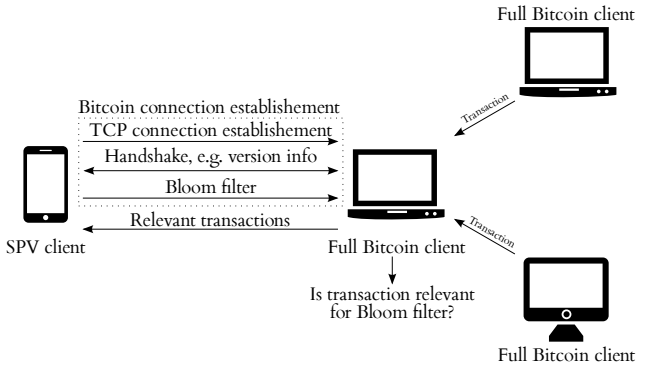

Figure 1: Sketch of the operation undergone by an SPV client. SPV clients connect to a full Bitcoin node, which only forwards to the SPV clients the transactions relevant to their Bloom filters.

but instead receive a subset of transactions, filtered for them by the full nodes to which they are connected[1]. This is mainly done to reduce the communication load on SPV clients, typically run on mobile devices. To reduce bandwidth consumption, SPV clients make use of Bloom filters [20]. A Bloom filter [14] is a space-efficient probabilistic data structure which is used to test membership of an element. An SPV client constructs a Bloom filter by embedding all the Bitcoin addresses which appear in its wallets. Upon connection to a full Bitcoin node, the constructed Bloom filter is outsourced to the full node following an initial handshake protocol (cf. Figure 1). Whenever the full node receives a transaction, it first checks to see if its input and/or output addresses match the SPV client's Bloom filter. If so, the full node forwards the received transaction to the SPV client.

Bloom Filters: Bloom filters have been first proposed by Bloom in 1970; we refer the readers to [15] for detailed information on Bloom filters. In SPV clients [2], a Bloom filter B of an SPV client is specified by the maximum number of elements that it can fit, denoted by M, without exceeding its target false-positive rate P_t. Let $m \leq M$ denote the number of elements that are inserted in $B(M, P_t)$. In SPV clients, the size of the filter n is computed as follows:

$$n = -\frac{M \ln(P_t)}{(\ln(2))^2} \tag{1}$$

A Bloom filter B basically consists of an array $B[.]$ of n bits accessed by k independent hash functions $H_1(.), \ldots, H_k(.)$, each of which maps an input string $x \in \{0,1\}^*$ to one of the n bits of the array. In SPV clients, k is computed as follows:

$$k = \ln(2)\frac{n}{M} \tag{2}$$

To insert an element $x \in \{0,1\}^*$ to a Bloom filter B, then $\forall j \in \{1, \ldots, k\}, B[H_j(x)] \leftarrow 1$. Similarly, to query the presence of an element $x \in \{0,1\}^*$ in B, then this entails computing $\wedge_{j=1}^{k} B[H_j(x)]$ (thus returning 1, if all corresponding bits are 1).

Bloom filters can generate a number of false positives, but cannot result in false negatives. The literature features a number of techniques to estimate the false positive rate of Bloom filters [15, 24]. In this paper, we focus on the proposal due to Mullin *et al.* [24] to estimate the false positive rates in Bloom filters (which is computed

[1]Currently, SPV clients connect to a default of four different randomly chosen nodes

over all possible inputs to the Bloom filter). More specifically, we compute the false positive rate of a filter $B(M, P_t)$ which has m elements, $P_f(m)$, as follows [15]:

$$P_f(m) = \left(1 - \left(1 - \frac{1}{n}\right)^{km}\right)^k \quad (3)$$

In Section 4.2, we show that this estimation experimentally matches the false positive rate featured by existing implementations of SPV clients. We acknowledge that there might be more accurate techniques for computing the false positive rates (e.g., [15]); our findings, however, show that the difference in false positives resulting from [15] and [24] was negligible and did not affect our results. We therefore elected to rely on the estimation of P_f which appears in [24]. Here, note that $P_f(M) \approx P_t$. That is, the target false positive rate of a Bloom filter is only reached when the number of elements contained in the filter matches M.

3. MODEL

In this section, we present our system and attacker model. We also introduce our privacy metric which will be used to quantify the privacy of SPV clients.

System Model.

We assume that lightweight SPV clients connect to the Bitcoin P2P network through full Bitcoin nodes. As described earlier, full nodes only inform the SPV clients about transactions specific to their corresponding Bloom filters. We further assume that the full Bitcoin nodes do not know the IP address of the SPV clients; for example, SPV clients might rely on TOR [4] when connecting to other Bitcoin nodes.

Attacker Model.

We assume that the adversary can compromise one or more full Bitcoin nodes and eavesdrop on communication links in order to acquire one or more Bloom filters pertaining to an SPV client. Here, the goal of the adversary is to identify the Bitcoin addresses that are inserted within a Bloom filter created by a particular SPV client. The addresses inserted in the Bloom filter typically correspond to addresses that the SPV client is interested in receiving information about (e.g., these addresses typically belong to the wallet of the SPV client). For example, the adversary might be connected to the node which generated the Bloom filter or might try to assign an identity to nodes according to their addresses, etc. Note that since the Bitcoin network provides currently less than 10,000 reachable full Bitcoin nodes, it is likely that regular nodes receive one or more filter pertaining to each SPV client over a sufficiently long period of time.

In our analysis, we assume that the adversary knows the parameters used to create a Bloom filter. This includes, for instance, the target false positive rate, P_t, that the Bloom filter is designed to achieve, and the number of hash functions k used in the Bloom filter. Since Bitcoin is an open payment system, we also assume that the adversary has access to all addresses/transactions which appear in the block chain, and to their respective order of execution.

Clearly, we assume that the adversary is computationally bounded. However, since the Bitcoin network only contains a bounded number of addresses (around 33 million Bitcoin addresses), the adversary can simply check whether all existing addresses match to a given Bloom filter.

Note that an adversary who is connected to an SPV client can see the transactions issued by the client and could potentially use this in order to learn the clients' addresses. This can be countered e.g., by SPV clients using TOR whenever they issue Bitcoin transactions. Instead, in this work, we focus on analyzing the privacy issues of the use of Bloom filters within existing SPV client implementations, which as we show, cannot be solved by simply relying on anonymizing networks. In Section 6, we still discuss the case where the adversary can additionally link Bitcoin addresses by observing the behavior of users, timing of transactions, etc.

Let $\mathbb{B}$ refer to the set of all existing Bitcoin addresses. Furthermore, let $\mathcal{B}_i$ refer to the set of all elements y in $\mathbb{B}$ that are members of Bloom filter B_i (i.e., for which a query in B_i returns true), and $\mathcal{F}_i \subset \mathcal{B}_i$ denote the set of false positives of B_i.

In our analysis, we assume that all elements map uniformly at random to the bits of the Bloom filter; that is, we assume that all addresses $y \in \mathcal{B}_i$ are equally likely to be a true member of B_i.

We further assume that the adversary can collect additional information from the system which can help her classify (a fraction of) the false positives exhibited by Bloom filters; for example, the adversary can try to identify some false positives generated by filters by analyzing two different Bloom filters which embed the same true positives (cf. Section 5). We capture this additional knowledge using the set $\mathbb{K} \subseteq \mathcal{F}_i$ which contains addresses in $\mathcal{F}_i$ that the adversary can correctly classify. We acknowledge that the adversary could also try to gather prior knowledge about the addresses inserted in the Bloom filter; in this work, we focus however on the case where the adversary does not have any prior knowledge about the true positives of the filter.

Our analysis throughout the rest of the paper does not exploit the fact that the adversary knows the public keys of Bitcoin addresses. Indeed, in current implementations of SPV clients, both the addresses and their public keys are inserted in the outsourced Bloom filter. As such, if the adversary knows both the address and its public key, then she can trivially test whether an address is a true positive of the filter by checking whether both the address and its public key are inserted within the filter. If not, then it is highly likely that the address is a false positive of the filter. We believe that the inclusion of both the address and its public key in the Bloom filter is a severe flaw in the current SPV client implementations—and can be easily countered; we thus do not exploit this flaw in our analysis. In fact, more than 99% of all Bitcoin transactions consist of payments to Bitcoin addresses (or the public key hash); moreover, only 4587 out of 33 million total addresses in the system received transactions destined for *both* their public keys and their public key hashes[2]. This means that for the vast majority of Bitcoin clients, there is no need to include both the public keys and their hashes (i.e., the Bitcoin addresses) in the Bloom filters; inserting one or the other would suffice (in more than 99% of the cases)[3].

Privacy Metric.

We quantify the privacy offered by a Bloom filter using the probability, $P_{h_{(j)}}$, that the adversary correctly guesses any j true positives of a Bloom filter among all positives that match the filter and which are not included in the knowledge of the adversary. More specifically, we measure $P_{h_{(j)}}$ achieved by a Bloom filter B_i, as follows: $P_{h_{(j)}} = \prod_{k=0}^{j-1} \frac{N-k}{N+S-k} = \frac{N}{N+S} \cdot \frac{N-1}{N+S-1} \cdot \ldots$. Here, N refers to the number of Bitcoin addresses inserted into B_i, and S denotes the cardinality of the set $\{\mathcal{F}_i - \mathbb{K}\}$; S therefore corresponds

[2]We obtained these numbers by parsing the Bitcoin block chain till block # 296000.

[3]Note that only inserting the addresses in the Bloom filter would suffice since regular nodes can easily hash the public keys and check whether they match the Bloom filter. However, this clearly incurs computational overhead on regular Bitcoin nodes.

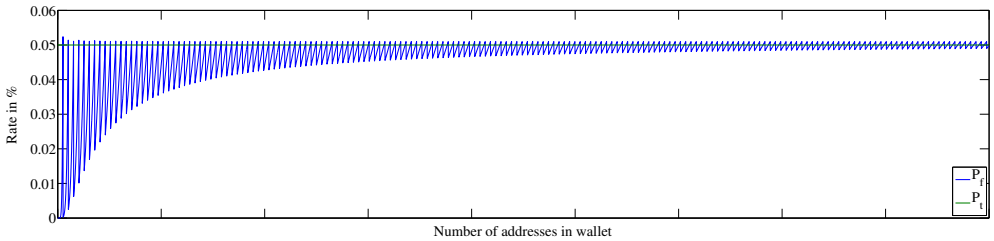

Figure 2: P_f, and P_t computed analytically with respect to the number of addresses N. Here, we assume that the SPV client did not restart since initialization.

$\mathbb{B}$	all existing Bitcoin addresses
B_i	Bloom filter i
$\mathcal{B}_i$	Positives of the Bloom filter
$\mathcal{F}_i$	False positives of the Bloom filter
N	Number of addresses inserted in the Bloom filter
S	False positives, the adversary has no knowledge about
M	Maximum number of elements the Bloom filter fits
m	Number of elements inserted in the Bloom filter
n	Size of the Bloom filter (in bits)
k	Number of hash functions of the Bloom filter
P_f	Actual false positive rate
P_t	Target false positive rate
$P_{h_{(j)}}$	Probability of correctly guessing any j true positives
X	Number of bits in the Bloom filter set to one

Table 1: Notations used throughout the paper.

to all false positives that match B_i, but for which the adversary does not have any knowledge about. Therefore, the probability that the adversary correctly guesses *all* the addresses of B_i is given by: $P_{h_{(N)}} = \prod_{k=0}^{N-1} \frac{N-k}{N+S-k} = \frac{N!S!}{(N+S)!}$. Clearly, the higher is $P_{h_{(.)}}$, the smaller is the privacy of the SPV node. Note that if the adversary is able to identify an SPV client (e.g., by some side channel information), then simply identifying any address pertaining to that client would be a considerable violation of its privacy. Otherwise, if the adversary can link a number of addresses to the same anonymous client, then the information offered by the clustering of these addresses offers the adversary considerable information about the profile of the client, such as its purchasing habits, etc.

Note that our privacy metric only assesses the probability of guessing addresses inserted within the Bloom filter. As mentioned earlier, these are the addresses that the SPV client is interested in receiving information about. $P_{h_{(.)}}$, however, does not necessarily capture the probability of guessing addresses which belong to the user of the SPV client. Indeed, addresses inserted within the filter do not necessarily have to belong to the user; for instance, the privacy of users—who only embed in their Bloom filters $L \leq N$ addresses—can be quantified by computing $\frac{L}{N} P_{h_{(.)}}$. Table 1 summarizes the notation used throughout the paper.

4. INFORMATION LEAKAGE DUE TO A SINGLE BLOOM FILTER

In this section, we start by analyzing the privacy provisions of existing SPV clients when the adversary acquires a single Bloom filter pertaining to an SPV client. In Section 5, we address the case when multiple Bloom filters are acquired by the adversary.

4.1 Leakage under a Single Bloom Filter

In existing SPV clients (which use the Bitcoinj library), a node initializes its Bloom filter B_i with a random nonce r, and specifies its target false positive rate P_t which can be achieved when a number of elements M have been inserted in the filter. M is equal to $M = m + 100 = 2N + 100$, where N is the number of Bitcoin addresses inserted in B_i.[4] Note that a Bitcoin address is inserted into the Bloom filter by adding both the corresponding public key and the public key hash to the filter; therefore $m = 2N$.

By default, the target false positive rate of the Bloom filter P_t is set to 0.05%[5]. The size of the filter n and the number of hashes k are computed given Equations 1 and 2, respectively.

Note that at initialization time, the SPV client is only equipped with $j = 1$ Bitcoin address. The corresponding Bloom filter will then be initially created to fit $M = 102$ elements, and will only contain $m = 2$ elements. However, if the SPV client restarts (e.g., mobile phone reboots, mobile application is restarted), then the Bloom filter will be re-computed with $M = 2j + 100$ (the SPV client stores the Bloom filter in volatile memory). In either cases, when the user acquires 50 or more additional addresses such that $m > M$, then the SPV client will resize the Bloom filter by recomputing $M = 2N + 100$, and will send the updated Bloom filter to the full Bitcoin nodes that it is connected to. This process reiterates whenever the number of addresses inserted in B_i increase such that $m > M$.

In Figure 2, we analytically compute P_t with respect to the number of addresses N that an SPV client is equipped with. Here, we assume that the adversary has access to only one Bloom filter pertaining to the SPV client, and that the adversary has no prior knowledge about addresses in Bitcoin (i.e., $\mathbb{K} = \phi$). Note that given n and k, the number of elements contained in a Bloom filter can be estimated by the adversary as follows [29]:

$$m \approx -n \frac{ln(1 - \frac{X}{n})}{k} \tag{4}$$

Here, X corresponds to the number of bits of the Bloom filter set to one. Given n and P_t, M can also be computed by the adversary from Equation 1.

Since $\mathbb{K} = \phi$, $S + N = |\mathcal{B}_i|$ (i.e., the number of all existing Bitcoin addresses which match B_i)[6]. Note that, in April 2014, the Bitcoin block chain comprised nearly $|\mathbb{B}|$ =33 million addresses. This means that an adversary can simply try all possible addresses

[4]The additional number '100' is added to m by the Bitcoin developers in order to avoid the recomputation of a new filter in the case where a user inserts up to 50 additional Bitcoin addresses.

[5]The Bitcoin developers claim that a target false positive rate of 0.1% should provide "very good privacy" [9].

[6]$|X|$ denotes the cardinality of set X.

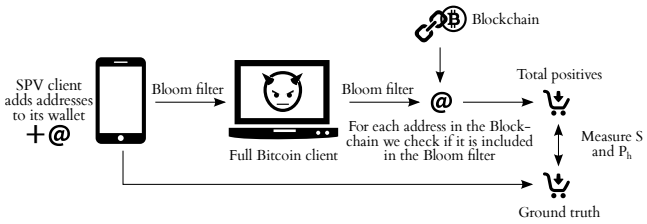

Figure 3: Experimental setup. We constructed around 18,060 different Bloom filters of various sizes and pertaining to 10 different wallets. For each Bloom filter, we compute the matching set of Bitcoin addresses, and we compare this set with the actual Bitcoin addresses inserted in each Bloom filter in order to compute $P_{h_{(.)}}$, and S.

in the Bitcoin system in order to compute $\mathcal{B}_i$. In doing so, it is straightforward to see that:

$$P_{h_{(j)}} = \prod_{k=0}^{j-1} \frac{N-k}{N+S-k} \approx \prod_{k=0}^{j-1} \frac{N-k}{N+|\mathbb{B}-N|P_f(2N)-k}, \tag{5}$$

where $N \ll |\mathbb{B}|$, and $m = 2N$ is the number of elements contained in the Bloom filter seen by the adversary. In analyzing our findings, we distinguish two cases:

1) $2N/M \leq 0.4$ *and* $N < 100$.

In Figure 2, we compute P_t and P_f with respect to the number of addresses N. Our results show that $P_{h_{(1)}}$ (the probability of correctly guessing one address as a true positive) is large when $2N/M \leq 0.4$, as long as $N < 100$. Given a modest number of addresses in the Bloom filter (i.e, $N < 100$), $P_f(m = 2N)$ is small when $\frac{m}{M}$ is small. As $\frac{m}{M}$ increases, $P_f(m = 2N)$ increases (and $P_{h_{(1)}}$ decreases). As shown in Figure 2, $P_f(m = 2N)$ is less affected by $\frac{m}{M}$ for larger M. For example, when $N = 10$, $P_{h_{(10)}} = 0.99$ which corresponds to the probability of correctly guessing all the true positives when the SPV client has 10 addresses.

This means that the information leakage in SPV clients is considerable for new Bitcoin users, and for Bitcoin users which restart their SPV clients. For instance, at initialization time, the Bloom filter of SPV clients is typically instantiated using $M = 102$. Our results show that if the user is new in the Bitcoin system and only has 1 Bitcoin address, $P_{h_{(1)}} \approx 1$, which signals complete lack of privacy. Recall that this observation also holds when the SPV client restarts and $N < 100$; for instance, when $N = 20$, $m = 40$, $P_{h_{(1)}} \approx \frac{N}{N+|\mathbb{B}|P_f(m)} \approx 0.98$ and $P_{h_{(20)}} \approx 0.20$ (cf. figure 4). We validate our analysis experimentally in Section 4.2.

2) $2N/M > 0.4$.

Conforming with our previous analysis, when $2N/M > 0.4$, $P_f(m = 2N)$ is close to P_t. Recall that in this case, the probability of correctly guessing one address reaches a local minimum when $N = \frac{M}{2}$ (cf. Figure 2). Our results also show that the global minimum achieved by $P_{h_{(1)}}$ is 0.002961 and is reached when the user has $N = 51$ addresses.

In addition, we analytically compute $P_{h_{(j)}}$ when the SPV client has 5, 10, 15 and 20 addresses. Our results in Figure 4 show that guessing all addresses given one filter which embeds less than 15 addresses can be achieved with almost 0.80 probability. This probability decreases as the number of addresses embedded within the filter increases beyond 15.

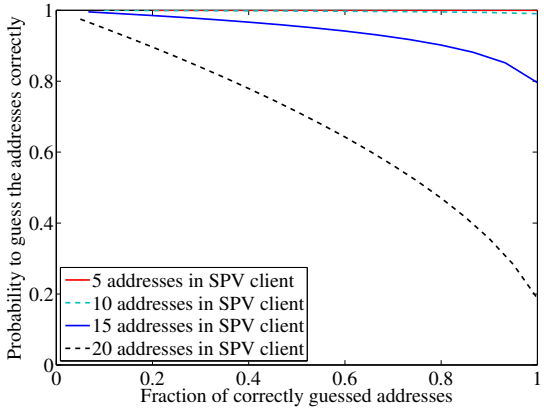

Figure 4: $P_{h_{(j)}}$ with respect to the number of addresses inserted in the Bloom filter. Given an SPV client with 5 addresses, all addresses can be guessed; when the SPV client has 20 addresses, 20% of the addresses can be guessed with almost 0.90 probability. Here, we assume that the user restarts its SPV client.

4.2 Experimental Evaluation

We now proceed to validate our analytical results empirically, by means of implementation. For that purpose, we parsed the block chain from the genesis block mined in 2009 until the beginning of April 2014 using the parser in [7] and collected nearly 33 million distinct addresses. In our evaluation, we rely on *Bitcoin Wallet* [8], which builds upon the standard Bitcoinj library (0.12-SNAPSHOT of end of April 2014).

As mentioned earlier, Bitcoinj initializes by constructing one address by default. The user can subsequently add addresses to his wallet. Our implementation setup is depicted in Figure 3. Here, we construct 10 different Bitcoin wallets, and we gradually increase the number of Bitcoin addresses which populate each wallet from 1 address to 9,000 Bitcoin addresses. Whenever new addresses are added to the wallets, we compute the modified Bloom filters; we increase the number of addresses by a step of 2 addresses to reach 19 addresses, and then by a step of 5 addresses to reach 8,999 addresses. As such, our experiments resulted in the evaluation of a total of 18,060 different Bloom filters which contain various number of elements, and pertain to 10 different SPV clients. For each Bloom filter, we compute the matching set of Bitcoin addresses, we compare this set with the actual addresses inserted in each Bloom filter in order to compute S, and $P_{h_{(.)}}$. Since we assume here that the adversary has no a priori knowledge about Bitcoin addresses, S corresponds to the number of existing Bitcoin addresses (among the 33,000,000 total Bitcoin addresses) that match the Bloom filter of the SPV client, and that are not the addresses of the SPV client (i.e., S corresponds to the false positives generated by the Bloom filter).

Each data point in our experiments corresponds to the average of 10 independent measurements obtained from each of the 10 wallets. Our results (cf. Table 2) confirm our analysis in Section 4.1. More specifically, our results show that given few addresses (small N), $P_{h_{(.)}}$ is large; for example, when $N = 19$, B_i results in an average of 6.1 false positives among all 33,000,000 addresses which corresponds to $P_{h_{(1)}} = 0.76$ and $P_{h_{(N/2)}} = 0.0433$. As N increases to 49 addresses, $P_{h_{(1)}}$ and $P_{h_{(N/2)}}$ decrease to 0.004 and 0, respectively. Indeed, when $N = 49$, $N/M \approx 0.5$, since initially B_i is computed with $M = 102$. Here, B_i incurs $P_f(98) \approx P_t$ false positive rates. When N increases beyond 51, then $m > M$, and the Bloom filter is resized to fit $M = 2N + 100$

elements. In this case, when $2N/M$ is small, $P_{h_{(.)}}$ is large. For instance, when $N = 54$, $P_{h_{(1)}}$ is 0.36, which corresponds to a total of 96 false positives seen by the adversary among all addresses in $\mathbb{B}$. This process re-iterates whenever B_i is resized, however as N increases, we observe that the fluctuations in $P_{h_{(.)}}$ decrease as $P_f(2N)$ converges towards P_t.

Our results therefore show that the information leakage due to the reliance on Bloom filters in SPV clients is considerable when the user has a modest number of addresses. In this case, our findings show that an adversary who captures a single Bloom filter can learn with high probability the Bitcoin addresses inserted within the filter.

In Figure 5, we also measure $P_{h_{(1)}}$ and the number of false positives when the SPV client restarts and has to re-compute its Bloom filter B_i. As mentioned earlier, our results show that the restarting of an SPV client causes $P_{h_{(1)}}$ to significantly increase when compared to the case where the client is not restarted. As mentioned earlier, at any restart and given N addresses, the SPV client recomputes B_i with a size of $M = 2N + 100$. Therefore, for modest values of N (e.g., $N < 100$), $2N/M$ becomes small, thus resulting in modest $P_f(2N)$ and a large $P_{h_{(1)}}$. Our findings also show that as N increases above 500 addresses, $P_{h_{(1)}}$ also increases. This is the case since the number of false positives provided by B_i is bounded by $P_t|\mathbb{B}|$, and does not depend on N.

Summary.

- We show that the probability $P_{h_{(.)}}$ that the attacker correctly guesses the addresses of the SPV client is large when the number N of addresses inserted in the filter is smaller than 20. For example, if the SPV client has 15 addresses in its wallet and restarts, correctly guessing all of its address can be achieved with a probability of at least $P_{h_{(15)}} = 0.8$.
- Due to addition of the constant '100' to M (the maximum number of elements the Bloom filter is designed for), the closer m (the actual number of elements in the filter) is to M, the higher is the privacy offered by the Bloom filter.

5. INFORMATION LEAKAGE DUE TO MULTIPLE BLOOM FILTERS

In Section 4.1, we analyzed the information leakage in the presence of an adversary who has access to a single Bloom filter instance of an SPV client. In this section, we extend our analysis to cover a more powerful adversary who can acquire multiple Bloom filters pertaining to SPV clients.

5.1 Leakage under Multiple Bloom Filters

In what follows, we assume that the adversary can acquire $b > 1$ Bloom filters pertaining to different users. For example, the adversary might be connected to SPV clients for a long period of time, and receive their updated Bloom filter. Alternatively, the adversary can acquire additional Bloom filters by compromising/colluding with other full Bitcoin nodes. Note that in our analysis, we do not assume that the adversary knows the correct association of the Bloom filters to the respective users.

Two Bloom Filters.

We start by analyzing the case where the adversary acquires two different Bloom filters B_1 and B_2. In the sequel, we focus on computing $P_{h_{(.)}}$ corresponding to filter B_1, which we assume to be the smallest of the two filters (in size). In analyzing the information leakage due to the acquisition of two Bloom filters, we distinguish two cases.

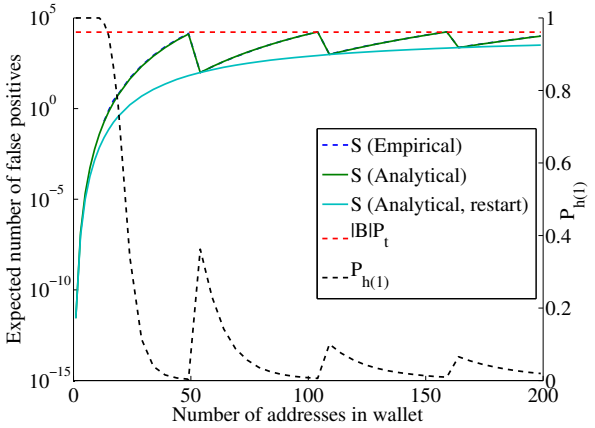

Figure 5: $P_{h_{(.)}}$ and S computed experimentally for the first 200 insertion of Bitcoin addresses into the Bloom filter (with and without restart of the SPV client).

1) B_1 and B_2 pertain to different users.

Recall that each Bloom filter is initialized with a random seed, chosen uniformly at random from $\{0,1\}^{64}$. Therefore, if B_1 and B_2 pertain to different users, then it is highly likely that they are initialized with different random seeds. This means that the false positives generated by each filter are highly likely to correspond to different addresses. Moreover, since different users will have different Bitcoin addresses, B_1 and B_2 will contain different elements. Therefore, $\mathcal{B}_1 \cap \mathcal{B}_2$ is likely to be comprised of only few addresses, if any. Notably, when B_1 and B_2 pertain to different users, then $|\mathcal{B}_1 \cap \mathcal{B}_2|$ can be computed as follows:

$$E[|\mathcal{B}_1 \cap \mathcal{B}_2|] \approx (|\mathcal{B}_1| - N_1)|\mathcal{B}_2| \frac{1}{|\mathbb{B}| - N_1} \quad (6)$$

$$\approx \frac{P_f(m_1)P_f(m_2)|\mathbb{B}|^2}{|\mathbb{B}| - N_1} \quad (7)$$

$$\approx P_f(m_1)P_f(m_2)|\mathbb{B}|, \quad (8)$$

where N_1 corresponds to the number of elements inserted in B_1. $E[|\mathcal{B}_1 \cap \mathcal{B}_2|]$ is the expected number of elements which match $\mathcal{B}_2$ and $\mathcal{B}_1$. The number of elements in $\mathbb{B}$ which match $\mathcal{B}_2$ is given by $P_f(m_2)|\mathbb{B}|$. Then, $E[|\mathcal{B}_1 \cap \mathcal{B}_2|]$ can be computed by assuming a binomial distribution with success probability $P_f(m_2)$, and with $P_f(m_2)|\mathbb{B}|$ number of trials.

Note that the adversary can compute m_1 (using Equation 4); if $m_1 > |\mathcal{B}_1 \cap \mathcal{B}_2|$, then this offers a clear distinguisher for the adversary that the two acquired Bloom filters B_1 and B_2 pertain to different user wallets.

2) B_1 and B_2 pertain to the same user.

On the other hand, in the case where B_1 and B_2 correspond to the same SPV client, three sub-cases emerge:

B_1 and B_2 use the same size/seed: This is the case when users, e.g., create additional Bitcoin addresses and need to update their outsourced Bloom filters to include those addresses. In this case, $\mathcal{B}_1$ and $\mathcal{B}_2$ are likely to comprise of similar Bitcoin addresses. This includes both the actual elements of the filters, i.e., the Bitcoin addresses of the user, and the false pos-

N / P_t	$P_{h_{(1)}}$ (0.05%)	$P_{h_{(1)}}$ (0.1%)	$P_{h_{(1)}}$ (0.5%)	$P_{h_{(\lceil N/2 \rceil)}}$ (0.05%)	$P_{h_{(\lceil N/2 \rceil)}}$ (0.1%)	$P_{h_{(\lceil N/2 \rceil)}}$ (0.5%)	$P_{h_{(N)}}$ (0.05%)	$P_{h_{(N)}}$ (0.1%)	$P_{h_{(N)}}$ (0.5%)
1	1(±0)	1(±0)	1(±0)	-	-	-	1	1	1
3	1(±0)	1(±0)	1(±0)	1	1	1	1	1	1
19	0.76(±0.03)	0.42(±0.03)	0.03(±0.002)	0.0433	0.000026	0	0	0	0
49	0.004(± 0.00032)	0.0021(±0.00019)	0.00035(±0.00002)	0	0	0	0	0	0
54	0.36(±0.02)	0.14(±0.0059)	0.01(±0.00089)	0	0	0	0	0	0
8,999	0.35(±0.002)	0.21(±0.00075)	0.05(±0.00032)	0	0	0	0	0	0

Table 2: $P_{h_{(.)}}$ with respect to P_t and N. Each data point is averaged over 10 independent runs; we also show the corresponding 95% confidence intervals.

itives generated by the Bloom filter. In this case, $|\mathcal{B}_1 \cap \mathcal{B}_2|$ can be computed as follows:

$$E[|\mathcal{B}_1 \cap \mathcal{B}_2|] \approx N_1 + P_f(2N_1)|\mathbb{B}| \quad (9)$$

$$P_{h_{(j)}} \approx \prod_{k=0}^{j-1} \frac{N_1 - k}{N_1 + P_f(2N_1)|\mathbb{B}| - k} \quad (10)$$

Notice that $|\mathcal{B}_1 \cap \mathcal{B}_2| = S_1 + N_1 = |\mathcal{B}_1|$ (since $\mathcal{B}_1 \subset \mathcal{B}_2$); therefore, $P_{h_{(j)}}$ is not affected by the acquisition of the second filter B_2.

B_1 and B_2 use different seeds: In existing SPV clients, the random nonce r used to instantiate the Bloom filter is stored in volatile memory. Therefore, each time the SPV client is restarted (e.g., smartphone reboots), a new filter will be created with a new seed chosen uniformly at random. If the adversary acquires two Bloom filters of the same user which are initialized with different seeds, then these filters are likely to exhibit different false positives. B_1 and B_2 will however comprise of a number of identical elements (which map to the Bitcoin addresses of the user). More specifically,

$$E[|\mathcal{B}_1 \cap \mathcal{B}_2|] \approx N_1 + (|\mathcal{B}_1| - N_1)\frac{|\mathcal{B}_2|}{|\mathbb{B}| - N_1} \quad (11)$$

$$\approx N_1 + P_f(m_1)P_f(m_2)|\mathbb{B}| \quad (12)$$

$$P_{h_{(j)}} \approx \prod_{k=0}^{j-1} \frac{N_1 - k}{N_1 + P_f(m_1)P_f(m_2)|\mathbb{B}| - k} \quad (13)$$

As we show in Section 5.2, the obtained $P_{h_{(j)}}$ is considerably large in this case, when compared to the case where the adversary has access to only one filter.

B_1 and B_2 use the same seed, but have different sizes: This is the case when users, e.g., create additional Bitcoin addresses beyond the capacity of their current Bloom filters. SPV clients therefore need to resize their Bloom filters. Note that filter resizing typically shuffles the bits of the Bloom filters; the resulting distribution of bits in the new resized filter is not necessarily pseudo-random (since the same seed is used) and depends on the sizes of the filters. As such, only the lower bound on $|\mathcal{B}_1 \cap \mathcal{B}_2|$ can be estimated using Equation 12 (which estimates the worst case where filter resizing causes a pseudo-random permutation of the bits of the filters); in Section 5.2, we confirm our analysis by means of experiments.

Recall that since there are only few tens of millions of addresses in Bitcoin, the adversary can brute-force search the entire list of Bitcoin addresses in order to acquire $\mathcal{B}_1$ and $\mathcal{B}_2$ and compute $\mathcal{B}_1 \cap \mathcal{B}_2$. Given any two Bloom filters B_1 and B_2, the adversary can easily guess whether these two Bloom filters contain Bitcoin addresses from the same wallet. Indeed, if $|\mathcal{B}_1 \cap \mathcal{B}_2|$ is small, then it is highly likely that B_1 and B_2 map to different elements (if m_1 and m_2 are not small), and therefore pertain to different users. On the other hand, when $|\mathcal{B}_1 \cap \mathcal{B}_2| \gg 0$, it is highly likely that all the Bitcoin addresses in the set $\mathcal{B}_1 \cap \mathcal{B}_2$ belong to the same SPV client.

Multiple Bloom Filters.

In the previous paragraphs, we discussed the case where the adversary is equipped with only two Bloom filters. We point out that our analysis equally applies to the case where the adversary possesses any number $b > 2$ of Bloom filters pertaining to the same entity.

As mentioned earlier, by computing the intersection between each pair of filters, the adversary can find common elements to different filters; this also enables the adversary to guess with high confidence whether different filters have been generated by the same client. Given b filters which belong to the same SPV client, the adversary can compute the number of elements inserted within each filter using Equation 4. In the sequel, we assume that filters $B_1, \ldots, B_b$ are sorted by increasing number of elements (i.e., B_b contains the largest number of elements), and that filters are constructed using different seeds. Let $\mathcal{K}_j = \mathcal{B}_j \cap \cdots \cap \mathcal{B}_{(b-1)}, \forall j \in [1, b-1]$.

Note that $|\mathcal{K}_1| \leq |\mathcal{K}_2| \cdots \leq |\mathcal{K}_{(b-1)}|$. Here, the larger the number of Bloom filters at the disposal of the adversary, the smaller is the error of the adversary in correctly classifying the genuine addresses of the SPV client, and the larger is $P_{h_{(.)}}$. That is, the larger is b, the smaller are the number of common false positives that are exhibited by the different filters, and the higher is the confidence of the adversary in identifying the false positives of B_j. Following Equation 12,

$$E[|\mathcal{K}_1|] = \min(|\mathcal{B}_1|, |\mathcal{B}_2|, \ldots) \approx N_1 + |\mathbb{B}| \prod_{\forall j} P_f(m_j) \quad (14)$$

$$P_{h_{(j)}} \approx \prod_{k=0}^{j-1} \frac{N_i - k}{N_i - k + |\mathbb{B}| \prod_{\forall j} P_f(m_j)} \quad (15)$$

Moreover, as j increases, $\mathcal{K}_j$ will contain more false positives, and $P_{h_{(j)}}$ will decrease.

5.2 Experimental Evaluation

In what follows, we validate our analysis in Section 5.1 by experiments using existing SPV clients. For that purpose, we rely on a similar setup to the one used in Section 4.2, and we perform four different experiments. We perform all four experiments by setting the target false positive rate P_t to 0.05%, 0.1% and 0.5%, respectively.

In our first experiment (Experiment 1), we create 10 different user wallets, each generating five different Bloom filter B_1,B_2,…, B_5 with the same size and using the same random seed r, but each having a different number of elements $N = \{25, 30, 35, 40, 45\}$. This corresponds to the case where 10 different users constantly

N_i, N_j	P_t (%)	$\lvert\mathcal{B}_i \cap \mathcal{B}_j\rvert$	$P_{h_{(1)}}$ $(b=2)$	$P_{h_{(1)}}$ $(b=1)$	$P_{h_{(N/2)}}$ $(b=2)$	$P_{h_{(N)}}$ $(b=2)$	N_i, N_j	P_t (%)	$\lvert\mathcal{B}_i \cap \mathcal{B}_j\rvert$	$P_{h_{(1)}}$ $(b=2)$	$P_{h_{(1)}}$ $(b=1)$	$P_{h_{(N/2)}}$ $(b=2)$	$P_{h_{(N)}}$ $(b=2)$
Experiment 1 (Same client, same seed, same size)							**Experiment 2 (Same client, same seed, different size)**						
25,45	0.05	83.6(±10.88)	0.2990	0.2910	0	0	70,270	0.05	70.3(±0.28)	0.9957	0.0678	0.8145	0.2502
25,45	0.1	245.0(±15.36)	0.1020	0.1070	0	0	70,270	0.1	71.70(±0.48)	0.9762	0.0266	0.3177	0.0011
25,45	0.5	3192.90(±230.79)	0.0078	0.0075	0	0	70,270	0.5	671.20(±46.54)	0.1043	0.0031	0	0
Experiment 3 (Same client, different seed)													
10, 10	0.05	10.0(±0.0)	1	0.9997	1	1	10, 10	0.5	10.0(±0.0)	1	0.7448	1 1	1
100, 100	0.05	100.0(±0.0)	1	0.1164	1	1	100, 100	0.5	110.90(±1.40)	0.9017	0.0052	0.0009	0
1,000, 1,000	0.05	1004.70(±1.54)	0.9953	0.0785	0.0390	0	1,000, 1,000	0.5	1499.70(±22.7)	0.6668	0.0077	0	0
5,000, 5,000	0.05	5007.30(±1.96)	0.9985	0.0003	0.0064	0	5,000, 5,000	0.5	5755(±15.05)	0.8688	0.0308	0	0
10, 10	0.1	10.0(±0.0)	1	0.9969	1	1	1,000, 1,000	0.1	1015.60(±2.37)	0.9846	0.0395	0	0
100, 100	0.1	100.30(±0.28)	0.9970	0.0464	0.8138	0.225	5,000, 5,000	0.1	5032.40(±3.15)	0.9936	0.1376	0	0

Table 3: Measuring $|\mathcal{B}_i \cap \mathcal{B}_j|$ and $P_{h_{(.)}}$ in Experiments 1,2, and 3, using filters of the same SPV client with respect to N and P_t. Here, $m_i \leq m_j$. Each data point is averaged over 10 independent runs; we also present the 95% confidence intervals.

insert new Bitcoin addresses and update their outsourced Bloom filters. Here, we assume that our adversary captures all 50 Bloom filters and applies our analysis described in Section 5.1 to learn additional information about the user addresses. In this experiment, we compute $I_1 = \mathcal{B}_1 \cap \mathcal{B}_5$ for each SPV client; we then report the average intersection size for all 10 wallets in Table 3.

Our results in Table 3 show that Bloom filters pertaining to the same SPV client, and which share the same initial seed, are likely to exhibit the same false positives (in addition to the elements inserted in the Bloom filter). Indeed, our results show that $|I_1|$—measured experimentally—matches Equation 9, irrespective of the target P_t; moreover, in this case, $P_{h_{(.)}}$ obtained using $b = 2$ Bloom filters (cf. Equation 13) is similar to $P_{h_{(.)}}$ when $b = 1$ one Bloom filter.

In our second experiment (Experiment 2), we extend our evaluation to account for the case where Bloom filters originating from the same user have different sizes. Here, we also create 10 different user wallets, each generating five different Bloom filters B_1,B_2,…, B_5 using the same random seed, but each having a different number of elements (respectively $N = \{70, 120, 170, 220, 270\}$) and different sizes ($M$ ranges between 3224 and 9680 bits). Analogously to Experiment 1, we compute $I_2 = \mathcal{B}_1 \cap \mathcal{B}_5$ for each SPV client; we then report the average intersection size for all 10 wallets in Table 3. Additionally, we compute the intersection set I_3 shared by B_1 from the first wallet, with a randomly chosen Bloom filter from the remaining 9 wallets. In Table 5, we report the average intersection set size over the 9 wallets. Our results in Tables 3 and 5 confirm our aforementioned analysis. Indeed, $|I_2|$ matches the values derived using Equation 12 when the Bloom filters pertain to the same user. Otherwise, $|I_3|$ matches Equation 6.

Our results also show that $P_{h_{(.)}}$ obtained using $b = 2$ Bloom filters (cf. Equation 12) is considerably larger when compared to the case where the adversary has access to only one Bloom filter (cf. Equation 5).

In our third experiment (Experiment 3), we investigate the case where Bloom filters pertaining to the same user are initialized with different random seeds. As mentioned earlier, this, e.g., corresponds to the case when the user restarts the SPV client. In this experiment, we create 10 different user wallets, each generating 16 different Bloom filters constructed using different initial seeds as follows: 4 filters B_1,B_2,B_3,B_4 with $N = 10$, $N = 100$, $N = 1,000$, and $N = 10,000$, respectively. For each wallet, and all filters of the same size, we compute $I_4 = \mathcal{B}_1 \cap \mathcal{B}_4$ and we report the average $|I_4|$ for each filter sizes in Table 3. Additionally, for each filter size, we also compute the intersection set I_5 using B_1 from the first wallet, with a randomly chosen Bloom filter from all other 9 wallets. In Table 5, we report the average intersection set size over the 9 wallets. Similar to Experiment 2, our results in Table 3 show that, irrespective of the filter size, and of the target false positive rate, $P_{h_{(.)}}$ significantly decreases when the adversary acquires $b = 2$ Bloom filters pertaining to the same SPV client.

Finally, in our final experiment (Experiment 4), we investigate the impact of having $b > 2$ Bloom filters pertaining to the same SPV client. For that purpose, we use 5 Bloom filters B_1,B_2,…,B_5 generated using different seeds with $N = \{3070, 3120, 3170, 3220, 3270\}$. We then compute $\mathcal{K}_j = \mathcal{B}_1 \cap \cdots \cap \mathcal{B}_{(j+1)}, \forall j \in [1, b-1]$, and the corresponding $P_{h_{(.)}}$ using Equation 15. Our findings are depicted in Table 4; our results validate our aforementioned analysis and show that the larger the number b of acquired Bloom filters of the same SPV client, the larger is $P_{h_{(.)}}$, and the smaller is the privacy of the user's addresses.

Summary.

- In Experiments 1,2, and 3, notice that $|I_3| \approx |I_5| \ll \frac{\min(m1,m2)}{2}$. This means that given any two Bloom filters B_1 and B_2, if $|B_1 \cap B_2| \ll \frac{\min(m1,m2)}{2}$ (estimated using Equation 1), then an adversary can easily tell whether any two Bloom filters pertain to the same SPV client.

- If the two Bloom filters acquired by the adversary belong to the same SPV client, the adversary can identify whether the SPV client has restarted while generating his Bloom filters; here, notice that $\frac{\min(m1,m2)}{2} \leq |I_4| \ll |I_2| \ll |I_1|$.

- $P_{h_{(.)}}$ corresponding to $b = 2$ filters is considerably larger when compared to the case where the adversary has access to one Bloom filter. This means that an adversary which can acquire more than one Bloom filter pertaining to an SPV client can learn considerable information about the addresses of the node—irrespective of the size of the Bloom filter and P_t. In this case, our results show that $P_{h_{(N)}}$ approaches 1, which signals full leakage of the addresses of the SPV client. $P_{h_{(.)}}$ increases to 1 as the number b of Bloom filters of the same SPV client captured by the adversary increases.

6. OUR PROPOSED SOLUTION

Given our findings, we propose in what follows a solution that enhances the privacy of SPV clients which rely on Bloom filters. Before presenting our solution, we briefly summarize our observations from Sections 4 and 5.

b	$P_{h_{(1)}}$ ($P_t = 0.05\%$)		$P_{h_{(1)}}$ ($P_t = 0.1\%$)		$P_{h_{(N/2)}}$ ($P_t = 0.05\%$)		$P_{h_{(N/2)}}$ ($P_t = 0.1\%$)		$P_{h_{(N)}}$ ($P_t = 0.05\%$)		$P_{h_{(N)}}$ ($P_t = 0.1\%$)	
	(Analy.)	(Emp.)	(Analy.)	(Emp.)	(Analy.)	(Emp.)	(Analy.)	(Emp.)	(Analy.)	(Emp.)	(Analy.)	(Emp.)
1	0.1713	0.1715	0.0926	0.0916	0	0	0	0	0	0	0	0
2	0.9978	0.9977	0.9911	0.9908	0.0091	0.0079	0	0	0	0	0	0
3	1	1	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1	1	1

Table 4: Experiment 4: $P_{h_{(.)}}$ w.r.t. the number b of Bloom filters which pertain to same SPV client. Here, we assume that each filter is generated using a different seed.

N_i	P_t (%)	$E[\lvert\mathcal{B}_i \cap \mathcal{B}_j\rvert]$ (Analytical)	$\lvert\mathcal{B}_i \cap \mathcal{B}_j\rvert$ (Empirical)
Experiment 2 (Different client, different seed, different size)			
-	0.05	-	0.0(±0)
-	0.1	-	1.22(±0.60)
-	0.5	-	31.60(±6.34)
Experiment 3 (Different client, different seed, same size)			
10	0.05	0	0.0(±0.0)
10	0.1	0	0.0(±0.0)
10	0.5	0	0.0(±0.0)
100	0.05	0	0.0(±0.0)
100	0.1	0.13	0.0(±0.0)
100	0.5	11.27	13.89(±1.64)
1,000	0.05	4.21	5.0(±0.97)
1,000	0.1	18.06	21.11(±2.35)
1,000	0.5	523.20	512.89(±23.21)
5,000	0.05	7.37	11.89(±1.73)
5,000	0.1	30.01	40.11(±3.40)
5,000	0.5	753.41	789.78(±18.92)

Table 5: Measuring $|\mathcal{B}_1 \cap \mathcal{B}_2|$ in Experiments 1,2, and 3, using filters pertaining to different SPV clients with respect to P_t. Each data point is averaged over 10 independent runs.

Observation 1: The number of elements inserted within a Bloom filter significantly affects the resulting false positive rate of the filter. This is especially true when the filter's size is modest (e.g., < 500). Indeed, the number of elements inserted in the filter should match at all times the filter's size in order to achieve the target false positive rate (i.e., $P_f(m) = P_t$).

Observation 2: The acquisition of multiple Bloom filters considerably reduces the privacy of SPV clients. The construction of multiple Bloom filters per SPV client should be avoided. Otherwise, different Bloom filters should be constructed with different initial seeds, and should contain different elements. In this way, an adversary does not gain considerable advantage when acquiring two or more Bloom filters.

Observation 3: SPV clients should keep state about their outsourced Bloom filters (i.e., on persistent storage) to avoid the need to re-compute a filter which contains the same elements using different parameters.

Observation 4: As mentioned earlier, inserting both the public key and the public key hash (the address) in the Bloom filter provides a sufficient distinguisher for the adversary in guessing whether an address is a true positive or not. Note that for the most common transaction type *Pubkey Hash* (P2PKH), inserting the hash of the public key is sufficient. However, there might be other transaction types where it is beneficial to also store the public key in the Bloom filter. In this case, the client can insert either a Bitcoin address or its corresponding public key (but not both) in the same Bloom filter. Indeed, sample experimental results that we conducted show that for almost 99% of all addresses in the network, it suffices to insert either the public key or the public key hash within the same Bloom filter in order to receive all the relevant transactions destined to the address.

Our Solution.

In what follows, we describe a solution which leverages Observations 1, 2, 3, and 4. Our proposed solution unfolds as follows. During the first setup of the client, each SPV client generates N Bitcoin addresses, and embeds them in a Bloom filter which can fit $M = m = N$. Here, the Bloom filter is constructed with a realistic target false positive rate P_t, which combined with N and M, results in a target privacy level (cf. Equation 3). Note that since $M = m$, then we ensure that the Bloom filter's false positive rate matches P_t (cf. Section 4). Here, we assume that only the address is inserted within each filter.

Clearly, since the user might not directly use all N addresses, some of his Bitcoin addresses will not be revealed and will remain in his wallets. Whenever users run out of their N addresses and need to get additional addresses, they repeat the aforementioned setup process. That is, users create an additional set of N addresses and embed them in a new Bloom filter—constructed with a new initial seed— with $M = m = N$, and using the previously chosen P_t. Here, the advantage of an adversary which captures one or more Bloom filters pertaining to the same SPV client is negligible, since these filters do not have any element in common.

Additionally, our solution requires the SPV clients to keep state, e.g., about each Bloom filter, to avoid the need of re-computation of the same filter if the client restarts at any point in time. We point out that the required storage overhead to maintain information about the state of each Bloom filter is negligible. Indeed, to keep all the necessary state to reconstruct a Bloom filter, an SPV client needs to locally store: *(i)* the number of addresses embedded in the filer (4 bytes), *(ii)* the used target false positive rate P_t (8 bytes), *(iii)* the used seed value (8 bytes), *(iv)* the *BloomUpdate* flag (2 bytes), and *(v)* the addresses inserted in the filter. The SPV client can add a pointer in the *ECKey* class of Bitcoinj in order to link each Bitcoin address to the Bloom filter that embeds it; the size of the pointer is roughly 2 bytes per address. This amounts to a total storage of $2N + 20$ bytes per Bloom filter. For example, when $N = 100$, the SPV client needs to store an additional 220 bytes per Bloom filter in order to reconstruct the filter upon restart; clearly, this overhead can be easily tolerated in existing SPV client implementations.

Clearly, our proposed solution can be directly integrated within existing SPV clients, and only incurs in small modifications to existing client implementations. Moreover, our solution does not incur additional overhead on the SPV clients—apart from the pre-generation of N Bitcoin addresses (which is only done at setup time), and the storage space required for each generated Bloom filter. Note that our solution requires SPV clients to outsource all their

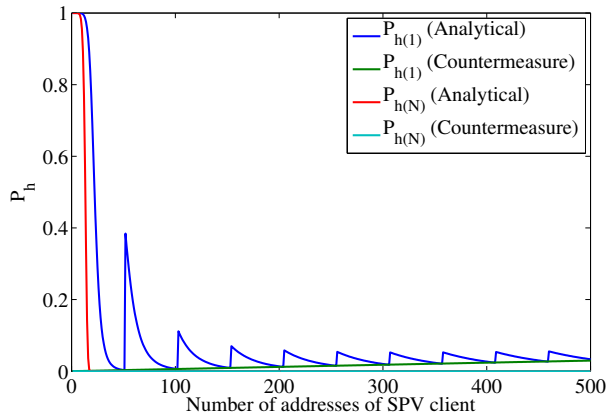

Figure 6: Evaluation of our countermeasure. Here, we measure $P_{h_{(1)}}$ and $P_{h_{(N)}}$ given $P_t = 0.05$ and assuming the current SPV client implementation. In computing $P_{h_{(1)}}$, we assume that the SPV client did not restart since initialization.

filters to full Bitcoin nodes since different filters embed different sets of their addresses. Therefore, N should be carefully chosen. If a user needs many addresses, choosing a larger N would avoid the generation of a larger number of Bloom filters. We leave the task of finding the best parameters when constructing a Bloom filter as an interesting direction for future work. Since users typically have few hundred Bitcoin addresses, we argue that this overhead can be largely tolerated given the current usage patterns in Bitcoin. Notice that the larger is N, the smaller are the number of Bloom filters created by the SPV client. Note that, in our solution, an SPV client should not outsource more than one Bloom filter to each regular node that it connects to. Indeed, if the client outsources several filters to the same node, then the regular node can correlate these filters (even if they do not embed the same elements).

In Figure 6, we analytically compute P_h given our countermeasure. Clearly, our countermeasure considerably decreases P_h, especially for clients that have a small number of addresses.

Additional Insights.

We point out that an adversary which has access to the Bitcoin block chain can observe that Bitcoin addresses which recently were used by an SPV client do match the Bloom filter of that client. This knowledge clearly reduces the privacy of newly used addresses. Here, we stress that every newly used address of an SPV client is likely to match the Bloom filters of a fraction P_t of all other SPV clients. In our solution, P_t therefore defines the minimum anonymity set size of each address.

One alternative would be to embed existing Bitcoin addresses that do not belong to the node, when constructing each Bloom filter. Subsequently, whenever an SPV client needs to generate a new Bitcoin address, it will choose a Bitcoin address which matches its existing Bloom filter[7]. However, this alternative also results in an anonymity set defined by P_t; it also comes at the expense of computational overhead incurred on the SPV clients. In an experiment that we conducted using a Bloom filter B_t of size 1608 bits with $P_t = 0.05\%$ and $m = 102$, we were able to generate three new Bitcoin addresses which match B_t; these addresses were found after 240,351, 415,877, and 5,767,346 tries, respectively.

[7]That can be achieved, e.g., by constantly generating a new Bitcoin address until it matches its existing Bloom filter.

Moreover, we point out that our analysis and solution do not address the case where the adversary can link addresses by using side-channel information from the Bitcoin block chain, namely:

Filtering false positives by date: SPV client implementations only appeared in the second half of 2011, which is 1.5 years after Bitcoin started. Therefore, if a Bitcoin address which was created before 2011 matches the Bloom filter of an SPV client, the adversary can label it, with high confidence, to be a false positive, and not a Bitcoin address of the node. Our proposed solution however relies on the assumption that the number of addresses created from 2011 exceeds by far that corresponding to the period from 2009-2011, during which Bitcoin was still not widely used at the time.

Clustering Bitcoin addresses: The adversary can make use of techniques such as [11, 18, 22] to link/cluster certain Bitcoin addresses based on user behavior, transaction amounts/time, etc., and assess whether an address matching a Bloom filter is a false positive or a true positive. For instance, if two addresses which match B_i are used as inputs to the same transaction, then the adversary can be certain that these addresses are linked to the same entity [11], and as such are unlikely to be false positives. Here, we point out that our proposed solution can be used in conjunction with existing solutions such as [10, 23] to prevent the linking/clustering of addresses using such techniques.

7. RELATED WORK

In this section, we overview related work in the area.

User Privacy in Bitcoin: Bitcoin has received considerable attention from the research community [5, 12, 16–19, 21].

In [11], Androulaki *et al.* evaluate user privacy in Bitcoin and show that Bitcoin leaks considerable information about the profiles of user. In [6], Elias investigates the legal aspects of privacy in Bitcoin. In [26], Reid and Harrigan explore user anonymity limits in Bitcoin. In [27], Ron and Shamir investigate how users move BTCs between their various accounts in order to better protect their privacy. In [22], the authors investigated the possibility of linking addresses of the same user together by utilizing the Bitcoin peers network address information (IPs). Miers *et al.* introduced in [23] ZeroCoin, a cryptographic extension to Bitcoin that augments the protocol to prevent the tracing of coin expenditure. In [10], Androulaki and Karame proposed an extension of ZeroCoin to hide the transaction values and address balances in the system.

Privacy in Bloom Filters: As far as we are aware, Mullin *et al.* [24] were the first to propose an estimate the false positive rate of Bloom filters. In [15], Christensen *et al.* propose a novel technique for computing the false positive rate, which results in tighter estimates when compared to [24].

In [13], Bianchi *et. al* quantify the privacy properties of Bloom filters; their analysis, however, does not address the privacy provisions when the adversary has access to multiple Bloom filters originating from the same entity. In [25], Nojima *et. al*, propose a cryptographically secure privacy-preserving Bloom-filtering protocol based on blind signatures; this proposal, however, incurs additional computational load on SPV nodes.

8. CONCLUSION

In this paper, we explored the privacy provisions due to the integration of Bloom filters in SPV clients. Our results show that

Bloom filters incur serious privacy leakage in existing SPV client implementations. More specifically, we show that a considerable number of the addresses of users of SPV clients which possess a modest number of Bitcoin addresses (e.g., < 20) are leaked by a single Bloom filter. Moreover, we show that a considerable number of the addresses of users is leaked if the adversary can collect two different Bloom filters issued by the same node, irrespective of the target false positive rate of the filter, and of the number of addresses owned by the user.

Given that such an information leakage might severely harm the privacy of users, we argue that the integration of appropriate countermeasures in the current SPV client implementation of Bitcoin emerges as a necessity. To this end, we propose a lightweight solution that enhances the privacy offered by Bloom filters; our proposal can be integrated within existing SPV client implementations with minimum modifications.

9. REFERENCES

[1] Core Development Status Report # 1 - Bitcoin, Available from https://bitcoinfoundation.org/2012/11/01/core-development-status-report-1/.

[2] BitcoinJ, Available from http://bitcoinj.github.io/.

[3] BitcoinJ limitations, Available from http://bitcoinj.github.io/limitations.

[4] TOR project. Available from: https://www.torproject.org/.

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

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

[7] Bitcoin Blockchain parser, 2013. Available from: https://github.com/znort987/blockparser.

[8] Bitcoin Wallet, Android, 2014. Available from: https://play.google.com/store/apps/details?id=de.schildbach.wallet.

[9] BitcoinJ, privacy assumptions, 2014. Available from: https://github.com/bitcoinj/bitcoinj/blob/ee2a91010e5cf66299684160d6a48a108ff2299b/core/src/main/java/com/google/bitcoin/core/PeerGroup.java#L250.

[10] Elli Androulaki and Ghassan Karame. Hiding transaction amounts and balances in bitcoin. In *Proceedings of International Conference on Trust & Trustworthy Computing (TRUST)*, 2014.

[11] Elli Androulaki, Ghassan Karame, and Srdjan Capkun. Evaluating user privacy in bitcoin. 2013. http://eprint.iacr.org/2012/596.pdf.

[12] S. Barber, X. Boyen, E. Shi, and E. Uzun. Bitter to Better - How to Make Bitcoin a Better Currency. In *Proceedings of Financial Cryptography and Data Security*, 2012.

[13] Giuseppe Bianchi, Lorenzo Bracciale, and Pierpaolo Loreti. Better than nothing privacy with bloom filters: To what extent? In *Privacy in Statistical Databases*, pages 348–363. Springer, 2012.

[14] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[15] Ken Christensen, Allen Roginsky, and Miguel Jimeno. A new analysis of the false positive rate of a bloom filter. *Information Processing Letters*, 110(21):944–949, 2010.

[16] J. Clark and A. Essex. (Short Paper) CommitCoin: Carbon Dating Commitments with Bitcoin. In *Proceedings of Financial Cryptography and Data Security*, 2012.

[17] C. Decker and R. Wattenhofer. Information Propagation in the Bitcoin Network. In *13-th IEEE International Conference on Peer-to-Peer Computing*, 2013.

[18] Meiklejohn et al. A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 127–140, New York, NY, USA, 2013. ACM.

[19] Arthur Gervais, Ghassan Karame, Srdjan Capkun, and Vedran Capkun. Is bitcoin a decentralized currency? *IEEE Security and Privacy Magazine, 2014*.

[20] Mike Hearn. Connection bloom filtering, 2012. Available from: https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki.

[21] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 906–917, New York, NY, USA, 2012. ACM.

[22] Philip Koshy, Diana Koshy, and Patrick McDaniel. An analysis of anonymity in bitcoin using p2p network traffic. 2014. http://fc14.ifca.ai/papers/fc14_submission_71.pdf.

[23] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. 2013.

[24] James K Mullin. A second look at bloom filters. *Communications of the ACM*, 26(8):570–571, 1983.

[25] Ryo Nojima and Youki Kadobayashi. Cryptographically secure bloom-filters. *Transactions on Data Privacy*, 2(2):131–139, 2009.

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

[27] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. 2013. http://eprint.iacr.org/2012/584.pdf.

[28] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2009.

[29] S Joshua Swamidass and Pierre Baldi. Mathematical correction for fingerprint similarity measures to improve chemical retrieval. *Journal of chemical information and modeling*, 47(3):952–964, 2007.