On the difficulty of hiding the balance of lightning network channels

Jordi Herrera-Joancomarti, Guillermo Navarro-Arribas, Alejandro Ranchal Pedrosa, Perez-Sola Cristina, Joaquin Garcia-Alfaro

On the Difficulty of Hiding the Balance of Lightning Network Channels

Jordi Herrera-Joancomartí
jordi.herrera@uab.cat
Universitat Autònoma de Barcelona
Cybercat

Guillermo Navarro-Arribas
guillermo.navarro@uab.cat
Universitat Autònoma de Barcelona
Cybercat

Alejandro Ranchal-Pedrosa
alejandro.ranchal_pedrosa@
telecom-sudparis.eu
Telecom SudParis

Cristina Pérez-Solà
cperez@deic.uab.cat
Universitat Rovira i Virgili
Cybercat

Joaquin Garcia-Alfaro
jgalfaro@ieee.org
CNRS SAMOVAR, Institut
Polytechnique de Paris, T. SudParis

Abstract

The Lightning Network is a second layer technology running on top of Bitcoin and other Blockchains. It is composed of a peer-to-peer network, used to transfer raw information data. Some of the links in the peer-to-peer network are identified as payment channels, used to conduct payments between two Lightning Network clients (i.e., the two nodes of the channel). Payment channels are created with a fixed credit amount, the channel capacity. The channel capacity, together with the IP address of the nodes, is published to allow a routing algorithm to find an existing path between two nodes that do not have a direct payment channel. However, to preserve users' privacy, the precise balance of the pair of nodes of a given channel (i.e. the bandwidth of the channel in each direction), is kept secret. Since balances are not announced, second-layer nodes probe routes iteratively, until they find a successful route to the destination for the amount required, if any. This feature makes the routing discovery protocol less efficient but preserves the privacy of channel balances. In this paper, we present an attack to disclose the balance of a channel in the Lightning Network. Our attack is based on performing multiple payments ensuring that none of them is finalized, minimizing the economical cost of the attack. We present experimental results that validate our claims, and countermeasures to handle the attack.

1 Introduction

The Lightning Network is a second layer running on top of Bitcoin and other Blockchains. Its goal is to address scalability problems with Bitcoin payment systems, and to lower transaction fees [17]. It uses a peer-to-peer network, to transfer raw information data. Some of the links in the peer-to-peer network are identified as payment channels. Payment channels allow payments to be routed between Lightning Network clients.

The Lightning Network not only provides better scalability. It also enables users to perform payments privately, and with low or negligible fees. In its current specification [31], payments are conducted with an onion-routing protocol [9], to provide each node in the route with the minimum information required to relay and retrieve payments. A node other than the origin and the destination does not know who is the origin node or the destination node; it only knows from whom and to whom forward the payment.

Channels are created with a fixed credit amount, the channel capacity. Channels can be used to perform payments between the two nodes of the channel. The channel capacity, together with the IP address of the nodes, is published to allow a routing discovery algorithm to find an existing path between two nodes that do not have a direct payment channel. To preserve privacy properties, and although the channel capacity can be known, the particular balances of each of the nodes of the channel at a given time is set confidential only to the two members of the channel. Given that such balances (i.e., the bandwidth of the channel in each direction) are not announced, routing nodes need to probe and monitor routes iteratively, until they find a successful path to the destination for the amount required, if any. This feature makes the routing discovery protocol less efficient, but preserves the privacy of channel balances.

In this paper, we present an attack to disclose the balance of a channel in the Lightning Network. Our attack is based on performing multiple payments, ensuring that none of them is finalized, and minimizing the economical cost of the attack. Our attack exploits the detailed information provided by the Lightning Network clients on the occurred errors. The onion-routing nature of the Lightning Network routing makes difficult for victims to detect the source of the attack, i.e., the source of the payments.

IACR PrePrint, January, 2019

.

Section 2 introduces the basic background to understand the proposed attack and countermeasures. Section 3 describes the adversarial model and provides a detailed description of the attack. Section 4 provides experimental results. Section 5 provides the countermeasures. Section 6 surveys related work. Section 7 concludes the paper.

2 Lightning Network Background

From an architectural point of view, the Lightning Network is a separated peer-to-peer (P2P) network, connected to the main Bitcoin P2P network [12]. More precisely, the Lightning Network is formed from nodes that run a Lightning software client [14, 32]. The client maintains a P2P network with other nodes of the Lightning Network and also a connection with a node in the Bitcoin main P2P network.

Once a node establishes a connection with a peer in the Lightning P2P Network, they can open a payment channel in which they can exchange Bitcoin transactions without the need for such transactions to be set down in the blockchain. This payment channel is not a real network connection, but a state of the P2P network connection that both nodes have in the Lightning P2P Network. For that reason, a payment channel between two users A and B cannot be created or used without the existence of a connection between A and B in the Ligthning P2P Network. This online model of the Lightning Network differs from the offline mechanism of standard Bitcoin payments, in which A and B may perform payments between them without being connected, since payments pass through the blockchain.

2.1 Payment Channels

The core element of the Lightning Network is the payment channel. A payment channel can be seen as a contract between exactly two parties, A and B, with a capacity amount of Bitcoins $C_{AB} = C_{BA}$ that is divided between the parties and can be exchanged. Such division, represented as the balance of A and balance of B, can be updated without having to send a transaction to the Bitcoin blockchain.

To create a payment channel, A and B need to set a node in the Lightning P2P Network and connect both nodes in that network. Once the connection is established, they proceed to create a payment channel. Channel creation is performed by sending a special transaction to the Bitcoin blockchain. The open nature of the Bitcoin blockchain allows any user to check when a channel is created and some basic information for that channel, like the capacity of the channel or the Bitcoin addresses that created the channel[1].

The Bitcoin transaction needed to create a channel is known as the *funding transaction*, a transaction in which one of the users deposits some bitcoins in a multisignature address controlled by both A and B. The total amount included in the funding transaction is the channel capacity, $C_{AB} = C_{BA}$.

Once a channel is open, the two users holding that channel can perform payments in both directions, only restricted to the balance that each of them has in the channel. The balance of each user in a channel is a fraction of the capacity of that channel, and can be indicated with $balance_{AB}$ for the balance that user A has in the channel and $balance_{BA}$ for the balance of B. Obviously, the following expression always holds: $C_{AB} = C_{BA} = (balance_{AB} + balance_{BA})$. For instance, if the user A has deposited 0.1 Bitcoins in the funding transaction of the channel, then the initial balances of the channel will be $balance_{AB} = 0.1$ and $balance_{BA} = 0$. When A performs a payment of 0.02 to B in that channel, the balances are updated accordingly: $balance_{AB} = 0.08$ and $balance_{BA} = 0.02$.

Payments in a channel are performed through *commitment transactions*. When A wants to perform a payment to B both users exchange a Bitcoin transaction with special features. In short, the transaction takes as input the funding transaction output of the channel and splits the input creating outputs in which every user gets the new balance of the channel. Since such a transaction is not published in the Bitcoin blockchain, but stored by each user of the channel, when a new payment has to be performed, new commitment transactions are created and exchanged. However, since the new commitment transactions spend the same output from the funding transaction as used in a previous committed transaction, a mechanism should be added to invalidate a previous commitment transaction once a new one has been exchanged. This mechanism is performed by a set of transactions that are kept offline. The transactions are only published if the counterpart intends to commit to an invalid, old, commitment transaction. Users are discouraged to use previous transactions since at every new commitment transaction, (secret) information of the previous one is revealed, giving the opportunity to honest users to punish a dishonest one, by collecting all the balance of the channel [26].

Notice that commitment transactions are valid formatted Bitcoin transactions that, although they are not intended to, can be posted in the Bitcoin blockchain at any time for any of the participants of the channel.

At any moment, users of a channel can close the channel and refund the balance each one has in the channel. This reset can be performed unilaterally, by any of the users sending the last commitment transaction to the Bitcoin blockchain or jointly, by creating a closing transaction in which the funding transaction output of the channel is spent in two outputs that equal the actual balances of each user. More types of channels and similar constructions have been proposed and generalized, identifying the same open-close funds mechanism [28].

[1] Although Bitcoin addresses are included, the identity of users A and B holding the channel may be preserved.

2.2 Multihop Payments

Channels described so far are of little use in real scenarios, since they are based on a two party agreement, and often a stable payment relation between only two users is not common. However, payment channels can be concatenated allowing to route payments between two users that do not hold a direct payment channel. When A wants to perform a payment to C and there is no direct payment channel between A and C, A tries to find a multihop path of direct channels. In case the path exists (for instance, with two hops), a sequence $A \leftrightarrow B \leftrightarrow C$ is established where each arrow indicates a payment channel. In such a case, if A wants to pay 0.01 Bitcoins to C, she can pay 0.01 Bitcoins to B and B can perform the 0.01 payment to C. The only condition that has to be set to perform the aforementioned mechanism is that $balance_{AB} \geq balance_{BC} \geq 0.01$.

In the multihop approach, payments at each individual payment channel cannot be performed exactly in the same way that with a single hop because user B has to enforce that he would receive the payment from A once he has performed the payment to C, otherwise he would lose the amount of the payment. The enforcement of this type of atomic exchange between all the nodes of the path (i.e., all simple one-hop payments have to be completed or none can be processed) is performed using Hashed Timelock Contracts (HTLCs) [11]. In an HTLC between the sender A and the receiver B, A can deposit Bitcoins that can be redeemed by B if B can perform a digital signature and provide a preimage of a hash value. Furthermore, the deposit performed by A has an expiration date after which A can retrieve the deposit providing a digital signature. The idea is that C generates a random value x and sends $h(x)$ to A. A performs the single hop payment to B with an HTLC based on $h(x)$ and B also performs the single hop payment to C with an HTLC based on the same value $h(x)$. In that way, since C knows x, he can redeem the transaction from B, but redeeming the transaction implies revealing the value of x which, in turn, implies that B may also redeem the payment from A.

Information in a Lightning multihop payment is routed through an onion routing protocol where every node of the path is only aware of his previous and next neighbor. For that reason, the payer is the one that decides the route of the payment based on the path availability.

To determine the path, the network topology of the payment channels of the Lightning Network is published. For each payment channel, the capacity of the channel and the fees of each node are advertised. Based on this information, the payer determines the path for the payment. However, for privacy reasons, the only information available for a channel is its capacity, but not the exact balance for each of the two users of the payment in which the capacity is split. Hence, it is possible that, although the capacity of the channel could allow to route a payment through that channel, its exact balance for each part may not allow the payment to be processed. In that case, the payer cannot be aware of that situation until she tries to proceed with the payment and the protocol returns an error indicating that a particular hop in the path has not enough funds. Such an error indicates that the payer needs to find another path which avoids that hop with insufficient balance in the right direction.

2.3 Invoices in the Lightning Network

In contrast with regular Bitcoin payments, where a payment request is based only on the Bitcoin address of the payee and the amount of the payment, Lightning Network payments are requested through invoices. When user A wants to make a payment to user D, the payee creates an invoice. The invoice includes, among others, the amount of the payment, p, the key of the destination node, the value $h(x)$ described in the previous section (to redeem atomically all the payments of the payment path) and an invoice signature from the payee (see [31] for all the details). Once A receives the invoice, she looks for a path in the Lightning Network to route the payment. In case there is no direct payment channel with enough capacity between A and D, then A should find a path, for instance $A \leftrightarrow B \leftrightarrow C \leftrightarrow D$, in which $C_{AB} \geq p$, $C_{BC} \geq p$, $C_{CD} \geq p$. With this information, using the public keys of each node, A creates an onion-routing path in which every node can only decrypt information with regard to the next hop payment, and the value $h(x)$ needed to redeem all the conducted payments.

Every node of the network performs a commitment transaction to the next hop of the path, by using the existing payment channel. The commitment transaction includes the value $h(x)$ in an output to be redeemed once the value x will be revealed. Upon reception of the last payment, D looks for the value x that he previously generated and used to include $h(x)$ in the invoice. Then, D reveals the value x to obtain the payment performed by C. In doing this, C gets to know about x and can redeem the payment from B that, in turn, can do the same.

3 Channel Balance Discovery

Our scenario assumes that Alice, A, and Bob, B, have an open Bitcoin Lightning Channel, AB, with capacity C_{AB}. Then, the objective of the adversary, Mallory M, is to disclose the exact balances that each node has in channel AB, that is $balance_{AB}$ and $balance_{BA}$.

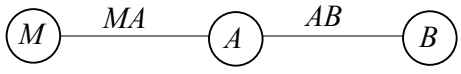

Figure 1. Simple single channel scenario.

To perform our attack, Mallory M needs to open a payment channel with[2] A, MA (see Figure 1). Once the MA channel is open, M performs a payment through the path $M \leftrightarrow A \leftrightarrow B$. In case that $balance_{MA} \geq balance_{AB}$, any payment with amount $p \leq balance_{AB}$ performed by M can be routed through that route and the payment will be correctly delivered to B. Obviously, a naive attack may be carried by performing multiple payments each of one increases p from $balance_{MA}$ step by step[3] until an error in the payment is obtained. The amount p of the last correctly processed payment can be considered the value $balance_{AB}$ and then $balance_{BA} = C_{AB} - balance_{AB}$.

The previously described attack can be enhanced, in order to reduce the economic cost for the adversary, by routing an invalid payment. To that end, M creates a fake invoice as if created by B, with a random value $h(x)$. However, the fake invoice cannot be detected by A but only by B, who will not be able to retrieve the corresponding x value (in fact, he could not locate the invoice, that was created by M) denying the last hop payment and, therefore, the whole payment.

3.1 Attack Extension

The previously described attack can be easily extended in order to discover the balance of all open channels that A has with n peers $B_1, B_2, \cdots B_n$ (see Figure 2). Notice that with the same set-up as before (M opens a single channel with A), M can also obtain the pairs $(balance_{AB_i}, balance_{B_iA})$ $\forall i = 1, \cdots, n$ as far as M is aware of the existence of B_i and $balance_{MA} \geq balance_{AB_i}$ $\forall i$. In that case, M should set payments with each end node B_i.

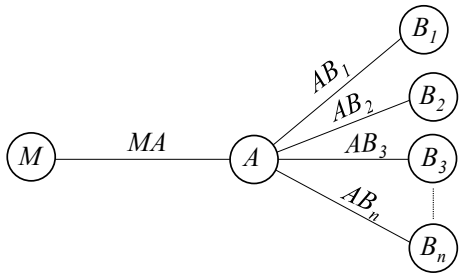

Figure 2. Multiple nodes scenario.

3.2 Attack Implementation

Algorithm 1 describes the first attack presented in the previous section, i.e., the adversary trying to discover the $balance_{AB}$ by performing invalid payments to B through the path $M \leftrightarrow A \leftrightarrow B$ route (cf. Figure 1). Algorithm 1 takes as inputs the target node B, the route to B (i.e., node A), the range of the

[2]There is a symmetry in the channel creation, so M can choose either A or B, or both, for the attack.

[3]The implemented approach, however, improves its efficiency by using a binary search to obtain the threshold value.

Algorithm 1: minmaxBandwidth

```
Data: route, target, maxFlow, minFlow,
      accuracy_threshold
Result: bwidth, an array of tuples that gives the range of bandwidth
        discovered for each channel
missingTests ← True;
bwidth.max ← maxFlow;
bwidth.min ← minFlow;
while missingTests do
    if bwidth.max − bwidth.min ≤ accuracy_threshold then
        missingTests ← False;
    end
    flow ← (bwidth.min + bwidth.max)/2;
    h(x) ← RandomValue;
    response ← sendFakePayment(route =
      [route, target], h(x), flow);
    if response = UnknownPaymentHash then
        if bwidth.min < flow then
            bwidth.min ← flow;
        end
    else if response = InsufficientFunds then
        if bwidth.max > flow then
            bwidth.max ← flow;
        end
    end
end
return bwidth
```

Algorithm 2: Complete Node Attack

```
Data: node, accuracy_threshold
Result: routes2Neighbors, an array of routes and bandwidths for
        each neighbor of the node
channelPoint ← getChannelPointListChannels(node);
created ← False;
if channelPoint is undefined then
    channelPoint = getChannelPointPendingChannels(node);
    if channelPoint is defined then
        waitChannelNotPending(channelPoint);
    else
        created ← True;
        createConnection(node.externalIP);
        maxFunding ← calculateMaxFunding;
        responseOpenChannel ← createChannelSync;
        channelPoint ← waitChannel(responseOpenChannel);
    end
end
neighbors ← getNeighbors(node);
for neighbor in neighbors do
    neighbor.minmaxBandwidth ←
      minmaxBandwidth(route =node, target =neighbor);
end
if created then
    closeChannel(channelPoint);
end
return routes2Neighbors
```

payment `minFlow` and `maxFlow`, that is $[0, C_{AB}]$, and the accuracy the adversary wants for the obtained balance. The function returns the value $balance_{AB}$.

The algorithm starts by trying a payment in the middle of the payment range (cf. Lines 8-10). The value $h(x)$ to perform the payment has been randomly created by M ensuring that the `sendFakePayment()` function always will return an error. If the failure message is "`UnknownPaymentHash`", then the adversary knows that the payment would have been possible (cf. Line 11), i.e., the channel can forward the payment. Thus, the minimum bandwidth of the channel can be updated to this new flow, unless its minimum was already greater (cf. Lines 12-14). If the failure message is instead "`InsufficientFunds`", then the payment could not pass through channel $A \leftrightarrow B$, due to insufficient funds[4] (Line 15). In this case, the maximum bandwidth (cf. Lines 16-18) is modified. Since the adversary cannot get any further information w.r.t. the maximum bandwidth, the new maximum and minimum values are those of the channel that produced the failure message, which held the lowest index in the route, w.r.t. the origin. Lines 5–7 ensure the termination of the algorithm when the range is less than the desired accuracy threshold (since, at least, the desired channel reaches such a precision when *maxFlow* equals the capacity of the channel created by the adversary created).

In order to estimate the balance of the channel, one simply has to add up 1% of the capacity of the channels to each of the results of the algorithm, since this is the default percentage that members of a channel require to always have as balance to be accountable for punishments in the event of fraud. In order to perform the attack described in Section 3.1, Algorithm 2 can be used. Given the victim node A, the adversary discovers all the channels she had open with other peers B_i for $i = 1, \cdots, n$ and obtains their current balances $balance_{AB_i}$ for $i = 1, \cdots, n$. We assume here that the adversary advertises its external IP address to the victim (i.e., the IP address which is accepting incoming connections). A straightforward variant of the attack can be conducted without releasing the external IP address of the adversary.

Lines 1-6 first check if the adversary has an open or pending open (i.e. waiting to get enough confirmations) channel with the victim node. If not, then the adversary creates the channel (Lines 7-13). Afterwards, the adversary gets the neighbors of the victim in Line 15, to perform Algorithm 1 on each of the channels with its neighbors (Line 17). Finally, if the attack opened a channel, it closes it in Line 20.

3.3 Adversary Model and Attack Cost

The Bitcoin Lightning Network is an open P2P network to which any user can connect, and for that reason, there is no special requirements for an adversary to perform the proposed attack, besides the ability to open a payment channel with the victim node A from which the adversary knows the IP address. Of course, node A could refuse to open a payment channel with the adversary, since such procedure should be authorized by both parties, and then the attack could not be initiated. However, nodes in the Lightning Network are expected to be willing to open new channels, to allow better connectivity. In case the adversary completely funds the funding transaction, node A should not need to provide any liquidity, being more likely to accept the request from the adversary to open the channel.

In order to disclose the balance of a channel, the adversary needs to open another channel (i.e., first hop in the path). To do this, the adversary needs to send two transactions to the Bitcoin blockchain: (1) a first transaction to open the channel and lock the funds of the channel; and (2) a second transaction to close the channel and refund the funds that were previously locked. No Lightning Network fees need to be used, since the attack does not fulfill any payment.

The total cost of the attack can be divided between the entrance barrier cost and the economic cost. On the one hand, the entrance barrier cost takes into account the economic resources that the adversary has to control to be able to perform the attack. Such resources will be completely recovered after the attack has been finished. On the other hand, the economic cost of the attack is the amount of money that the adversary will lose due to the execution of the attack.

Regarding the entrance barrier cost, the proposed attack needs to fund a Lightning Channel to perform the balance disclose of other channels. In order to achieve maximum accuracy, the adversary needs to open a channel with its maximum capacity, that at present time is bounded at 0.16777215 BTC (stock symbol for Bitcoins, in which 1 BTC represents $100,000,000$ satoshis[5]). Hence, the entrance barrier cost will be around[6] 640.05 USD (United States Dollars).

With regard to the economic cost of the attack, three values have to be taken into account: (i) the fee corresponding to the funding transaction of the channel; (ii) the fee corresponding to the transaction that closes the channel; and (iii) the financial cost from having funds of the channel locked during the attack execution. Although the amount of bitcoins deposited in the funding transaction has to be, at least, as big as the biggest capacity of the channel that the adversary wants to attack, the cost in fees of the transaction does not depend on the amount deposited in the channel but on the size in bytes of the transaction. Additionally, being such size mostly independent from its inputs that will vary for each particular transaction, funding transactions with a single input can cost as low as 0.00001534 BTC[7]. Secondly, and

[4]The payment should always pass through the channel $M \leftrightarrow A$ since M controls the channel and can ensure that $balance_{MA} \geq C_{AB}$.

[5]Smallest amount within Bitcoin, i.e., one hundred millionth of a BTC.

[6]Exchange rate from Jan 10, 2019 in which 1 BTC exchanges at 3, 815 USD.

[7]See, for instance, transaction: 930d1c204258afee13fac4d45f9fa98e6e807c918cdbfde49f9d56e2dc482f4a

regarding the closing transaction, it is difficult to estimate the exact fee for a generic closing transaction, since multiple parameters may affect such a value. A cost rounding 0.00000463 BTC can be achieved, as can be seen in different existing closing transactions[8].

The financial costs derived from the locking funds can be measured in terms of standard interest rate [8]. However, as we detail in the next section, even a standard fixed annual interest rate of 4% implies negligible values when estimating this type of costs, since the time to perform the aforementioned attack is in the range of seconds. Then, the economic cost of the attack will be around 0.00002 BTC, i.e., equivalent to 0.0763 USD. Such a reduced cost can be used not only to disclose the balance of a single channel, but also to disclosing the balances of all the open channels of node A.

4 Experimental Results

To analyze the feasibility of our attack, we have performed two different evaluations. The first evaluation focuses on the Lightning Network running over the Bitcoin Mainnet Blockchain, where real value is being transacted. In this first evaluation, we estimate the impact of our attack over Mainnet, based on topological measures, as well as the cost estimation that such an attack could have. The second evaluation focuses on the Lightning Network running over the Bitcoin Testnet Blockchain, to test a real attack over transactional functionality. This second evaluation reports the technical feasibility of the attack.

The choice of only running the second evaluation over Testnet, rather than executing the attack over Mainnet, does not follow any technical, nor economical, reasons. In fact, experiments over Testnet have a very erratic and unrealistic behavior. The implications on the exact information extracted from Testnet are not always easy to extrapolate over Mainnet. However, our decision of conducting the real evaluation only over Testnet follows ethical reasons. In addition, we performed a responsible disclosure to the developers of the Lightning Network, about our findings.

4.1 Bitcoin Mainnet Evaluation

To analyze the feasibility of our attack in the Bitcoin Lightning Mainnet, we have performed some measurements on that network that is composed by 1,732 nodes and 6,501 channels (snapshot taken the 8th of January, 2019).

At the moment of writing, there is a detail in the main implementations of the Lightning Network that may let our attack less effective, providing a bound on the balance of the channel rather than the exact balance. Lightning Network implementations have two main limits, one on the maximum amount to pay in one single payment (`MAX_PAYMENT_ALLOWED`), and another one on the maximum amount with which to create a channel (`MAX_FUNDING_ALLOWED`), with values 4,294,967 and 16,777,215 satoshis, respectively. Regarding the logic of Algorithm 1 and with these limits in mind, we can perform the attack to a channel with capacity C_{AB} and obtain its exact balance only if $C_{AB} \leq$ `MAX_PAYMENT_ALLOWED`. Should that not be the case, and considering that `MAX_FUNDING_ALLOWED` is almost four times the `MAX_PAYMENT_ALLOWED`, it is possible that, if a channel has a big amount of funds in both ends, the attack will not see the actual balance, but provide a lower bound for that balance that will be `MAX_PAYMENT_ALLOWED`. In these cases, the exact balance of the channel can be obtained only depending on the actual balance of the channel, and the direction in which the attack is performed.

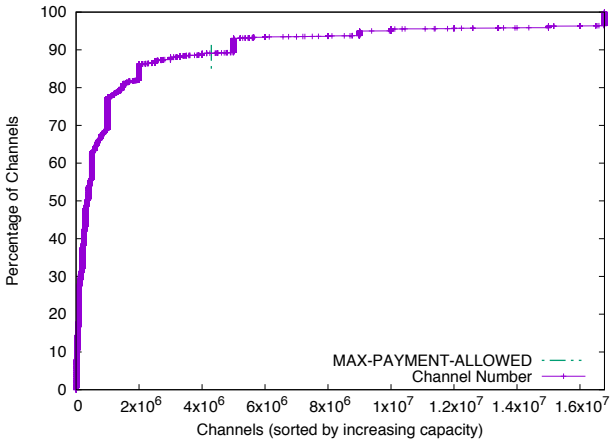

Figure 3. Percentage of deanonymizable channels, per number of channels attacked, sorted by increasing capacity.

Figure 3 shows that the number of channels in the Mainnet for which the exact balance may not be recovered is very low. The plot shows the cumulative distribution of the channels by its capacity, that is, for a given capacity, it shows the percentage of channels with lower or equal such capacity. We can see that there are 89.10% of channels in Mainnet with lower capacity than `MAX_PAYMENT_ALLOWED`, which means that their balances can be exactly disclosed.

Another interesting measure performed in Mainnet is the cost for an adversary to compute the balances of all the channels in the network. As mentioned in Section 3.3, if an adversary wants to perform the attack, the cost can be minimized by choosing as victims of the attack (i.e., node A in Figure 2) nodes that are highly connected. A single channel creation is required to get the balance of all the channels connected to such a node. A good strategy is to select the nodes by their number of channels and perform the attack until all the channels are processed.

It is the funding transaction corresponding to the Channel Id 614573123866132481 opened on January 17, 2019, by node 021387e1257d1da1c93996e10e7c4e2a2183683e978e60e40ae9f1927b86fabd27

[8]For instance, Channel Id 608922733705166848 with total capacity 0.1 BTC has been closed with the following close transaction 8da4d6b708eabbedaeb978e88fb8a5331c6e164c64cf9e561ba165dbdd200e71

Figure 4 shows the percentage of channels that can be deanonymized by attacking a given number of nodes, assuming that the nodes are sorted by their number of channels. We can see that we can deanonymize 50% of the channels by just attacking 18 nodes, 80% with 78 nodes, and 90% with 141 nodes. Moreover, we can easily estimate the minimum vertex cover of the Lightning Network using the local-ratio algorithm [5], which yields a set of 624 nodes. In other words, with less than 624 attacks to specific nodes, one could cover all the channels in the network. Note that, by the local-ratio algorithm, the vertex cover is guaranteed to be at most twice the minimum vertex cover. The actual size of the minimum cover set will range between 312 and 624 nodes. Then, to disclose the balance for all the network, there is a trade-off between the entrance barrier cost and the time needed to perform the attack. The attack can be parallelized by opening channels with each of the nodes at the same time. In the worst case, this implies an entrance barrier cost of $624 \times 640.05 = 339,391.2$ USD. If time is not a constrain, channels can be open sequentially, hence lowering the entrance barrier cost to 640.05 USD. From Figure 4, we can observe that by attacking only 78 nodes, an adversary can disclose the balance of 80% of the channels. Therefore, by performing a parallel attack with these settings, the entrance barrier cost gets reduced to $78 \times 640.05 = 49,923.9$ USD. With regard to the economic cost of the attack, and since the transactions fees are charged per channel, the parallel or sequential strategy does not affect the total cost. The total cost is low even in the worst case scenario, i.e., attacking the 624 nodes, in which it reaches $624 \times 0.0763 = 47.61$ USD.

Besides the economical cost of the attack, we can also consider a time cost estimation. Algorithm 1 looks for the balance in the same form as a binary search algorithm looks for a number in a sorted list. Therefore, each iteration of the algorithm reduces the range by half. The initial range would normally be always set by $maxFlow = channel\_capacity$ and $minFlow = 0$. Given the above-mentioned limitation, the current execution of the algorithm sets $maxFlow = min\{channel\_capacity, MAX\_PAYMENT\_ALLOWED\}$ instead. This means that, in the worst-case scenario, the algorithm iterates 23 times, considering an $accuracy\_threshold$ of 1. In the general case, the number of iterations is:

$$log_2\left(\left\lceil \frac{maxFlow - minFlow}{accuracy\_threshold} \right\rceil\right) \quad (1)$$

which means that there are three ways of reducing the number of iterations (and thus the running time to perform the attack on a given channel):

- reduce $maxFlow - minFlow$ by using historic information from previous iterations of the attack,
- reduce $maxFlow$ by choosing a channel with small capacity,
- increase $accuracy\_threshold$ and allow more coarse-grained results.

Regardless of the current `MAX_FUNDING_ALLOWED` and specially `MAX_PAYMENT_ALLOWED` limit, it is easy to predict the number of iterations for each of the channels capacity. Figure 5 shows the number of iterations for all existing channels, if there was no `MAX_PAYMENT_ALLOWED` limit, depending on the accuracy threshold. All currently existing channels have less capacity than `MAX_FUNDING_ALLOWED`, for which 24 iterations are enough to find out the balance within a range of 1 satoshi. In general, with n iterations, one can perform the attack of channels of capacity 2^n. Given that the running time between iterations can be considered constant, we can estimate the time required to perform an attack given the number of iterations. This time is also shown in Figure 5 in the right y axis. The actual time is taken as the average from the tests performed in the Bitcoin Testnet (see Section 4.2).

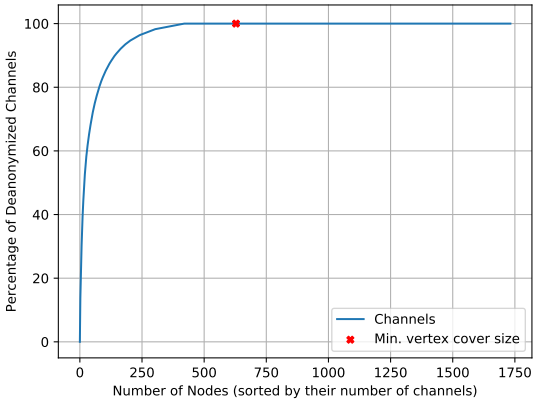

Figure 4. Percentage of deanonymizable channels per attacked node.

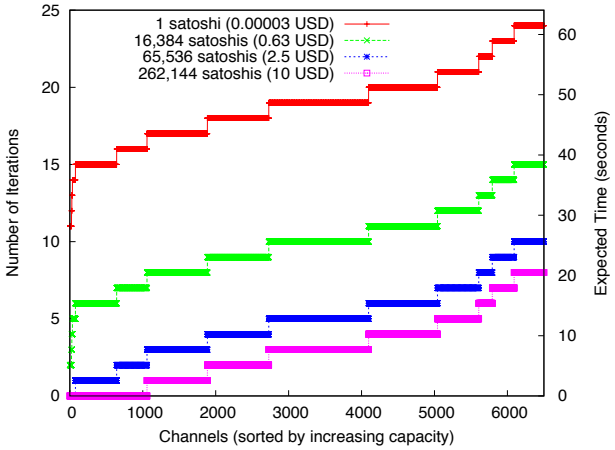

Figure 5. Expected time to perform attack on each channel, sorted by increasing capacity.

4.2 Bitcoin Testnet Evaluation

In order to provide a proof-of-concept of our attack, we have developed and performed a real attack on the Lightning Network running over Testnet. To perform the attack, we first identified 11 nodes with the largest amount of Lightning Channels in Testnet, as seen by a local deployment of *lnd* [32]. These 11 nodes have 2,518 open channels, more than 50% of the total number of channels in Testnet. Then, we performed the attack described in Algorithm 2 by sequentially connecting to each of the nodes and opening a channel, in order to retrieve the exact balance of the 2,518 channels they had previously open with other nodes of the network.

Contrarily to traditional Bitcoin payments, the Lightning Network requires both users, and also intermediary hops, to be online for the payment to take place. Whereas the adversary (our node) and intermediary nodes (the 11 nodes we open channels with) were online and responsive throughout the whole attack, the destination payment nodes (i.e., each of the targets in each iteration of Algorithm 1) were in most cases not online. This is likely due to nodes that were created in the Testnet for occasional testing and that may even not be used anymore. At the time of writing, Lightning Network client implementations do not have a mechanism to filter out unused channels, or unresponsive nodes. Out of the 2,518 target nodes that our attack contacted, only 710 replied. The remaining 1808 requests failed due to long delays, or simply because of their unresponsive behavior.

Out of the 710 channels that behaved normally, and each of the iterations of the attacks on each on them, we extracted the average time, median time, and minimum time per iteration of Algorithm 1. These values are 2,562, 2,603 and 1,106 milliseconds, respectively. Figure 6 shows the time (both in seconds and in number of iterations) that it would take to perform the attack, depending on the capacity of a channel, and the desired range (i.e., the accuracy_threshold parameter in Algorithm 1) in satoshis.

Notice that by detecting a balance with an accuracy of 0.0391 USD on a channel of capacity MAX_PAYMENT_ALLOWED takes about 33.3 seconds, according to the average; 33.84 seconds, according to median; and 14.4 seconds, following the minimum time. Detecting a balance with an accuracy of 1 USD takes about 20 seconds in both average and median; and less than 10 seconds in the minimum case. Detecting the balance with an accuracy of 10 USD can be conducted in 10.25, 10.4 and 4.4 seconds, respectively.

Figure 7 shows the time it would take to perform the attack on all of the channels of a single node, by assuming that the adversary only controls one channel, and ignoring the MAX_PAYMENT_ALLOWED limitation. The adversary could perform the attack in parallel by opening multiple channels and probing different channels at the same time. We see that 1,432 nodes can be attacked in a minute; or less in the sequential attack, with an accuracy of 10 USD; 1,369 with

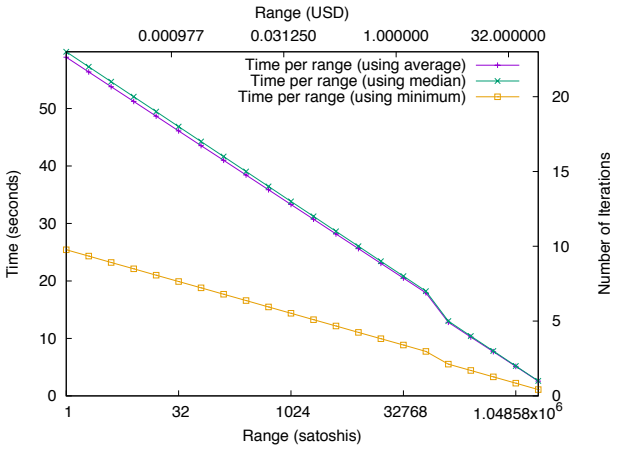

Figure 6. Time to perform the attack (in seconds left and in number of iterations right) per desired accuracy range (in satoshis bottom and in USD top).

an accuracy of 2.5 USD; 1,249 with 0.63 USD and 608 with 3.815×10^{-5} USD; or 1 satoshi, out of 1,682 nodes with visible channels by the adversary.

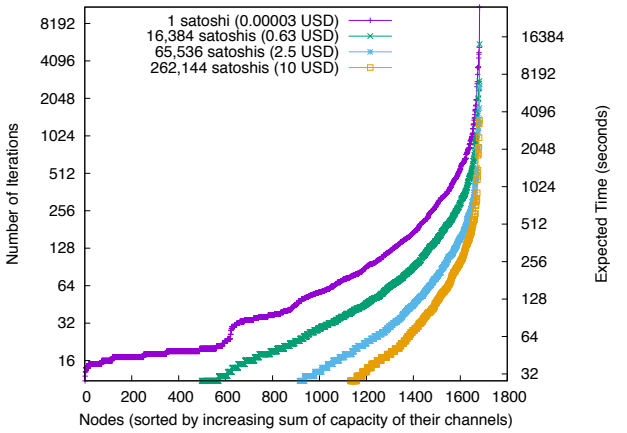

Figure 7. Expected time to perform attack on all channels of each node. Left y axis as per number of iterations, right y axis as per time in seconds.

Besides the strict attack of finding the balances of the channels, information about the balance can be used to measure the state of the Lightning Network. For the Lightning Network to maximize its utility, it is important for channels to be balanced or leveled. For this reason, we include the notion of level percentage. Given a channel between two nodes A and B, with capacity C_{AB}, and balance of each of the nodes $balance_{AB}$ for the balance of A and $balance_{BA}$ for that of B, then we refer to the level percentage lp_{AB} as:

$$lp_{AB} = \frac{C_{AB} - (0.01 \times C_{AB} + balance_{AB})}{C_{AB}} \times 100 \qquad (2)$$

And analogously to lp_{BA} as:

$$lp_{BA} = \frac{C_{AB} - (0.01 \times C_{AB} + balance_{BA})}{C_{AB}} \times 100 \quad (3)$$

Intuitively, if lp_{AB} is closer to zero, then A holds most of the capacity of the channel, and can thus perform payments to B, but not receive payments from B. Similarly, the closer lp_{AB} gets to 100, the less balance A holds in the channel, and the more payments it can receive from B. Ideally, without further knowledge on the characteristics of each channel, all the channels should always be equally leveled to ensure a good functioning of the network. Figure 8 shows the level percentage of the 272 channels of which we could find out the exact balance, even with the testing limit of `MAX_PAYMENT_ALLOWED`. Figure 8 shows that, for the channels whose balances we could find out, the level percentages are destituted: either nodes can only make payments, or nodes can only receive payments. A better payment network should provide more channels close to a level percentage of 50%.

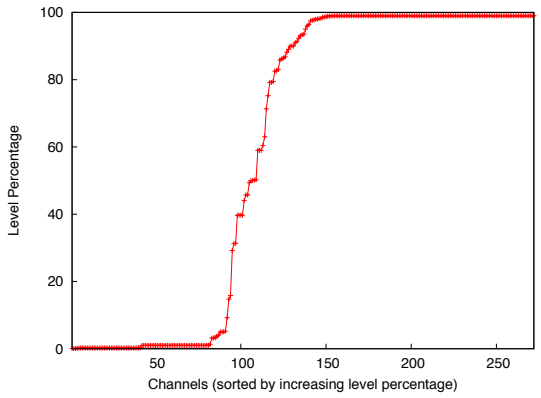

Figure 8. Level percentage of each channel.

5 Discussion

Several countermeasures can be developed to mitigate the attack described in this paper. A first solution relies on denying access to the debugging messages, e.g., by silently dropping the information provided by the *failure message* referred to in Section 3.2, Algorithm 1. Other possibilities include selectively or randomly denying given payment requests, or even allowing dynamically rechargeable payment channels to fully mask and randomize existing balances of two Lightning Network channel points. Additional details about the aforementioned solutions follow.

5.1 Payment Requests Denial

A first approach is for a node to randomly deny a given percentage of payment requests, e.g., by setting a *dropping rate* parameter in the node. The requesting node does not know the reason of the rejection (by using the *failure message* with no debugging information described in Section 3). This can make the attack unfeasible since the adversary might assume that the route has failed because there are not enough funds to carry the requested payment. The solution can be seen as a typical approach for masking information based on introducing noise. In this case, the adversary receives wrong information that makes the attack probabilistic or simply unfeasible. There is clearly the typical trade-off between privacy and usability tied to the dropping rate parameter. Several improvements can be developed in order to improve such trade-off.

Instead of setting the dropping rate at random, a node could define a more selective approach. We can identify some indications revealing that a balance disclosure attack (or other type of attack) is being carried on. For example, consider that node A receives a payment request from node B. Then node A can use the following information to decide its dropping rate:

– 1. Consider the number of channels and rate of payment requests of node B. If node A receives lots of payment requests from node B, and node B has just one channel (other than the AB channel). Node A can consider this as an abnormal situation, potentially an evidence of an adversary perpetrating the balance disclosure attack, hence increasing the dropping rate of the node in real time.

– 2. Consecutive payment requests with a suspicious amount pattern. If node A receives from node B (independently of its number of channels) payment requests that follow the pattern described in Algorithm 1, the node A considers again the situation as an anomaly, and increases the dropping rate of the node.

These are just two examples of simple heuristics that can be used to detect an abnormal behavior. In the general case, we can model the behavior of a node and provide anomaly detection measures to dynamically tune the dropping rate for specific nodes or situations. In the end, the node administrator can set the dropping rate and decide the privacy degree willing to accept. This privacy comes with the cost of not routing some payments that might be legitimate. We believe, however that a good trade-off can be achieved with relatively simple measures, like the ones outlined above.

5.2 Dynamic Absorption of Negative Balances

Another way to address the general attack uncovered in this paper could be the extension of the current implementations of the Lightning Network, by including additional masking functionalities capable of absorbing negative balances. This would be similar to energy-driven techniques discussed in [4], where adversaries trying to estimate metering consumption and billing functionality by adversarial collection of metering metadata get concealed by privacy-preserving mechanisms. The addition of synthetic computing and storage functionality, relying on charging channels, can be put in place as well between Lightning Network channel points, hence avoiding fine-grained collection of nodes' information

to mitigate the deanonymisation attack presented in Section 3.1. In addition, the masking solution can be randomized as in [3, 35], in such a way that the adversarial monitoring of balances between payment channels gets transformed from fine-grained processing into coarse-grained collection, hence guaranteeing that more powerful adversaries will fail at properly retrying accurate balances between two payment channels points of the Lightning Network.

The absorption of negative balances will act as a networking countermeasure to handle periodic (adversarial) probing to identify balance capacity flows, i.e., to hide and avoid accurate collection of channel node balance capacities. This shall lead to anonymity provable protection (i.e., protection with anonymity guarantees that can be proven in a formal way) like the one in [4], and whose goal is to mask transactional information flows in payment streams. The underlying techniques can rely on rechargeable swapping channels between Lightning Network nodes, used to mask channels' balance by adding or subtracting resource capacities (e.g., by increasing or decreasing the real bandwidth of each payment channel). The goal is to establish strong anonymity guarantees in the sense of differential privacy [13]. To achieve such anonymity guarantees in realistic settings, further work must be conducted, e.g., to establish the influence of, and the interplay between, capacity and throughput bounds that real payment channels of Lightning Network nodes must face.

The overall solution shall provide integrated methods based on cascading noise, allowing for payment channel on-the-fly recharging functionality, able to mask capacity and throughput of Lightning Network nodes with either discrete and continuous time constraints. We shall also make sure that the addition and subtraction of masking resources holds the minimal possible impact to the payment network, e.g., in terms of service disruption. Of course, this solution will require an important effort for its implementation in current payment networks such as the Lightning Network. It might require to redesign how payment networks work and provide means to allow the absorption of negative balances somehow by interested parties. A more straightforward approach could be the use of private channels between nodes (channels not announced to the whole network) that allow for such negative balance to be privately compensated.

6 Related Work

There are a number of studies that investigate adversarial issues, in terms of privacy and information disclosure, in current cryptocurrencies and routing literature. We survey next some examples, structuring the existing work in such two main research lines.

6.1 Cryptocurrencies Literature

Traditionally, anonymity in Bitcoin-like cryptocurrencies relies on pseudonymity, i.e., users creating any number of anonymous account addresses that are used later on, to identify the transactions. However, an underlying, non-anonymous, Internet infrastructure, together with the availability of transactions meta-data stored in a blockchain, allows the development of deanonymization tools. We follow four main recent literature classifications [10, 17]: (1) blockchain analysis, (2) network monitoring, (3) attacks to mixing protocols, and (4) balance disclosure.

In terms of blockchain analysis, and since any blockchain includes, by definition, all the transactions performed by the system, deanonymization may take advantage of such information. A simple analysis of the blockchain may provide information such as *from which Bitcoin addresses the money comes from, and to which Bitcoin addresses it goes to* [29]. Since users in the Bitcoin ecosystem can create any number of addresses, a more powerful tool to deanonymize transactions is to cluster all addresses in the blockchain that belong to the same user. Different proposals exist to conduct address analysis, such as clustering and similar techniques [2, 22, 30, 34]. From shadow address analysis to multiple input clustering and also behavior-based clustering techniques like k-means and Hierarchical Agglomerative Clustering can be used to enhance the cluster creation. Once the clustering set for one user is large enough, deanonymization becomes possible by using external information on Bitcoin addresses (e.g., posts, forums, markets, and market exchanges) that could identify at least one of the addresses.

With regard to traffic analysis, we recall that transactions in Bitcoin are transmitted through a P2P network [12]. Hence, metadata such as TCP or IP headers, which can be obtained by using traditional network analysis tools, can also be used as an underlying base for novel deanonymization tools [20]. Transaction eavesdropping can be performed to, e.g., discovering the IP addresses corresponding to those nodes that are broadcasting a transaction for the first time. To match an IP address with a Bitcoin address, the problem can be modeled as an evaluation of association rules, identifying the corresponding confidence scores and the support counts for the rule. Deanonymization of Bitcoin transactions can also benefit from the existence of Bitcoin sessions, initiated by nodes that get unreachable after a given period of time [21], e.g., nodes that are hidden behind NAT (Network Address Translation) and TOR (The Onion Router) connections, by identifying transaction patterns via fingerprinting techniques to characterize and highlight weaknesses in terms of pseudonymity models for Bitcoin users.

Mixing protocol attacks exist in the literature [6, 7, 23, 25]. Mixing protocols are tools designed to enhance the anonymity of Bitcoin transactions by, e.g., shuffling the information in order to hinder the relation between the input and the output values of the transactions. Bitcoin users send Bitcoins from one address to a mix service and receive from the mix service the Bitcoins to another address that could not be linked with the original one. The effectiveness of

such systems has been analyzed by different authors. They found a clear structure that allow understanding how this services work and may be used for deanonymization. Modeling and analysis of the P2P Bitcoin networking stack in term of anonymity properties, by providing source inference over graphs via epidemic source finding [24], has identified that the real Bitcoin P2P network topology offers a low degree of anonymity [15].

The work in [1] addresses the risk of leakage w.r.t. the currency balance of Bitcoin addresses and claim the necessity of hiding transaction values as well as address balances in Bitcoin, e.g., for those users who opt-out from exchanging assets. The work builds upon the assumption that Bitcoin traders may end accepting transactions immediately without waiting the necessary confirmations. This exposes them to traditional risks in terms of double spending, even for clients that are not miners [19]. The feasibility of such kind of attacks relies on directly broadcasting to the seller double-spending transactions, but in a different location of the network, within a similar time window frame. Countermeasures to the attack lead to new designs in which much more control about the precise balances of every node in the network is guaranteed. This additional control has as collateral consequences a higher likelihood in terms of deanonymization of Bitcoin users. The authors developed additional means to address the collateral consequences by hiding the balance of transactions by moving to alternative cryptocurrencies like Zerocash [33], with much stronger privacy guarantees in terms of anonymity, via zero knowledge proof techniques, while guaranteeing functionality and efficiency.

Finally, the feasibility of deanonymizing Internet privacy services such as Tor, due to leakage information of Bitcoin technologies has also been discussed in the recent literature. Authors in [18] provide deanonymization techniques to show that using Bitcoin as a payment method may leak sensitive information to disclose Tor hidden services. The techniques rely on the possibility of an adversary to link those Bitcoin users who publicly share their Bitcoin addresses on online social networks, with hidden services, and which publicly share the mapping about their Bitcoin addresses on their onion landing pages.

6.2 Routing Literature

In the previous section, we have listed existing related work in cryptocurrencies literature. An increasing number of research work deals with privacy issues to cryptocurrencies users, such as traceability of both application and network-layer data, and the limitation of mixing services. Beyond the realm of cryptocurrencies, a large number of existing literature on different areas, from distributed computing to ad hoc and sensor networking literature, shows similar issues to those uncovered in this paper. Transaction meta-data disclosure, e.g., unauthorized reporting of balance accounts on distributed networks, beyond the scope of Bitcoin and the Lightning Network, deal with similar findings and problems. Secure routing techniques have been reported vulnerable to similar adversarial probing attacks. Attacks to routing in ad hoc and sensor networks, like *black-holing* and *vampire-like attacks* [36], aim at draining the battery life of autonomous sensor nodes, to affect the routing capabilities of the whole system. In such attacks, the adversary perpetrates similar learning and discovery phases as the ones discussed in our work, prior to conducting the final attack. For instance, the adversary conducts a discovery phase to learn the accumulative energy dynamics of the network, by probing and estimating the level of available battery on each individual node of the system. Countermeasures to the attack, like those in [16, 27] are similar to some of the ideas discussed in Section 5.

Likewise, energy-driven literature on smart-grid and smart-metering environments share similarities to the issues and solutions discussed in our work. The privacy-preserving techniques discussed in [4], related to smart-metering privacy scenarios, show similar adversaries trying to estimate user energy consumption and billing functionalities by collectively collecting smart-metering metadata. Solutions are proposed in order to get adversaries concealed by new data-processing functionality. The approach relies on the addition of synthetic computing and storage functionalities, to avoid fine-grained collection of users' information. This kind of solutions can moreover be randomized as the approaches presented in [3, 35]. In the end, the goals is to increase the difficulty of adversarial monitoring tasks, to retrieve information about consumption channels. Identification and learning techniques by the adversary can still be put in place, by transforming the process in a two-stage transformation of data, hence starting with a fine-grained processing; then, moving to a coarse-grained processing.

7 Conclusion

In this paper, we have addressed privacy issues related to the Bitcoin Lightning Network. Today, to preserve users' privacy between two channel payment points of the Bitcoin Lightning Network, the precise balance (i.e. the bandwidth of the precise channel points, in each direction), is kept secret. Since the balances are not announced, second-layer nodes probe routes iteratively, until they find a successful route to the destination for the amount required, if any. Such feature makes the routing discovery protocol less efficient, but preserves the privacy of channel balances. Publicly disclosing the updated balance of channels each time they are updated would allow users to efficiently discover routes in the network, without having to probe a route in order to ensure the channels do have enough balance to support a payment. However, it would also allow adversaries to trace payments through the network, by observing how balances fluctuate. On the contrary, not providing any information about a channel's state would provide privacy to users, but

would render the network unusable to route multihop payments. Public disclosure of balances implies thus a trade-off between payment privacy and route finding efficiency in the network (and therefore usability).

Our work uncovers a balance discovery attack that can be used by Lightning Network adversaries, in order to deanonymize network payments. We have presented an analysis, complemented by experimental results that validate our claims. We have also performed a responsible disclosure to the developers of the Lightning Network, and discussed some potential countermeasures to handle the problem uncovered by our work.

Acknowledgements — The authors gratefully acknowledge financial support from the BART (Blockchain Advanced Research & Technologies) initiative (cf. http://bart-blockchain.fr/).

References

[1] E. Androulaki and G. O. Karame. Hiding transaction amounts and balances in bitcoin. In *International Conference on Trust and Trustworthy Computing*, pages 161–178. Springer, 2014.

[2] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun. Evaluating user privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 34–51. Springer, 2013.

[3] M. Backes, I. Goldberg, A. Kate, and E. Mohammadi. Provably secure and practical onion routing. In *2012 IEEE 25th Computer Security Foundations Symposium*, pages 369–385. IEEE, 2012.

[4] M. Backes and S. Meiser. Differentially private smart metering with battery recharging. In *Data Privacy Management and Autonomous Spontaneous Security*, pages 194–212. Springer, 2014.

[5] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–46, 1985.

[6] G. Bissias, A. P. Ozisik, B. N. Levine, and M. Liberatore. Sybil-resistant mixing for bitcoin. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 149–158. ACM, 2014.

[7] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In *International Conference on Financial Cryptography and Data Security*, pages 486–504. Springer, 2014.

[8] S. Brânzei, E. Segal-Halevi, and A. Zohar. How to charge lightning. *CoRR*, abs/1712.10222, 2017.

[9] S. Castillo-Pérez and J. Garcia-Alfaro. Onion routing circuit construction via latency graphs. *Computers & Security*, 37:197–214, 2013.

[10] M. Conti, S. Kumar, C. Lal, and S. Ruj. A survey on security and privacy issues of bitcoin. *IEEE Communications Surveys & Tutorials*, 2018.

[11] C. Decker and R. Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer, 2015.

[12] J. A. D. Donet, C. Pérez-Sola, and J. Herrera-Joancomartí. The bitcoin P2P network. In *International Conference on Financial Cryptography and Data Security*, pages 87–102. Springer, 2014.

[13] C. Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[14] Elements Project. c-lightning – a lightning network implementation in C. https://github.com/ElementsProject/lightning, 2019.

[15] G. Fanti and P. Viswanath. Deanonymization in the bitcoin P2P network. In *Advances in Neural Information Processing Systems*, pages 1364–1373, 2017.

[16] E. Gelenbe and Y. M. Kadioglu. Energy life-time of wireless nodes with network attacks and mitigation. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2018.

[17] J. Herrera-Joancomartí. Research and challenges on bitcoin anonymity. In *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*, pages 3–16. Springer, 2014.

[18] H. A. Jawaheri, M. A. Sabah, Y. Boshmaf, and A. Erbad. When a small leak sinks a great ship: Deanonymizing tor hidden service users through bitcoin transactions analysis. *arXiv preprint arXiv:1801.07501*, 2018.

[19] G. O. Karame, E. Androulaki, and S. Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 906–917. ACM, 2012.

[20] P. Koshy, D. Koshy, and P. McDaniel. An analysis of anonymity in bitcoin using P2P network traffic. In *International Conference on Financial Cryptography and Data Security*, pages 469–485. Springer, 2014.

[21] I. D. Mastan and S. Paul. A new approach to deanonymization of unreachable bitcoin nodes. In *International Conference on Cryptology and Network Security*, pages 277–298. Springer, 2017.

[22] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM, 2013.

[23] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[24] C. Milling, C. Caramanis, S. Mannor, and S. Shakkottai. On identifying the causative network of an epidemic. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pages 909–914. IEEE, 2012.

[25] M. Moser, R. Bohme, and D. Breuker. An inquiry into money laundering tools in the bitcoin ecosystem. In *eCrime Researchers Summit (eCRS), 2013*, pages 1–14. IEEE, 2013.

[26] J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2015.

[27] C. Pu and S. Lim. A light-weight countermeasure to forwarding misbehavior in wireless sensor networks: design, analysis, and evaluation. *IEEE Systems Journal*, 12(1):834–842, 2018.

[28] A. Ranchal-Pedrosa, M. Potop-Butucaru, and S. Tucci-Piergiovanni. Lightning factories. Cryptology ePrint Archive, Report 2018/918, 2018. https://eprint.iacr.org/2018/918.

[29] F. Reid and M. Harrigan. An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*, pages 197–223. Springer, 2013.

[30] D. Ron and A. Shamir. How did dread pirate roberts acquire and protect his bitcoin wealth? In *International Conference on Financial Cryptography and Data Security*, pages 3–15. Springer, 2014.

[31] A. Samokhvalov, J. Poon, and O. Osuntokun. Lightning Network In-Progress Specifications. BOLT 11: Invoice Protocol for Lightning Payments, 2018.

[32] A. Samokhvalov, J. Poon, and O. Osuntokun. The lightning network daemon, https://github.com/lightningnetwork/lnd, 2018.

[33] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy (SP)*, pages 459–474. IEEE, 2014.

[34] M. Spagnuolo, F. Maggi, and S. Zanero. Bitiodine: Extracting intelligence from the bitcoin network. In *International Conference on Financial Cryptography and Data Security*, pages 457–468. Springer, 2014.

[35] D. Sy, R. Chen, and L. Bao. Odar: On-demand anonymous routing in ad hoc networks. In *Mobile Adhoc and Sensor Systems (MASS), 2006 IEEE International Conference on*, pages 267–276. IEEE, 2006.

[36] E. Y. Vasserman and N. Hopper. Vampire attacks: draining life from wireless ad hoc sensor networks. *IEEE transactions on mobile computing*, 12(2):318–332, 2013.