Payment Splitting in Lightning Network as a Mitigation Against Balance Discovery Attacks

Gijs van Dam[*]

2023-09-11

Abstract

Bitcoin has a low throughput of around 7 transactions per second. The Lightning Network (LN) is a solution meant to improve that throughput while also improving privacy. LN is a Payment Channel Network (PCN) that runs as a peer-to-peer network on top of Bitcoin and improves scalability by keeping most transactions off-chain without sacrificing the trustless character of Bitcoin. Prior work showed that LN is susceptible to the Balance Discovery Attack that allows for individual channel balances to be revealed, threatening users' privacy. In this work we introduce Payment Splitting and Switching (PSS), a way of splitting up payments in LN at intermediary hops along the payment path. PSS drastically reduces the information an attacker can obtain through a BDA. Using real-world data in an LN simulator we demonstrate that the information gain for the attacker drops up to 62% when PSS is deployed. Apart from its potential as mitigation against BDA, PSS also shows promise for increased LN throughput and as a mitigation against jamming attacks.

1 Introduction

Bitcoin (Nakamoto, 2008) has a scalability problem that limits the amount of transactions to seven per second. This throughput is constrained by Bitcoin's blocksize of 1 MB. Increasing the block size or the rate at which blocks are generated deteriorates the security of the Bitcoin network (Sompolinsky and Zohar, 2015). Layer-two (L2) protocols (Gudgeon et al., 2020) aim to address the issue of scalability without changing the inherent characteristic of Bitcoin's low throughput. The most noticeable L2 protocol to date is Lightning Network (LN) (Poon and Dryja, 2016). LN is a p2p network of nodes that maintain payment channels with their peers. In LN, transactions are sent over these payment channels, resulting in off-chain transfer of value. These transfers are possible between untrusted parties along the payment route, as the underlying contracts are enforceable via broadcast to the Bitcoin blockchain. A Payment Channel Network (PCN) allows for multi-hop payments between nodes that do not share a channel.

Bitcoin's privacy issues are well documented (Androulaki et al., 2013; Meiklejohn et al., 2013). Because LN keeps the vast majority of transactions off-chain, it is

[*]gvandam@gmail.com

also seen as a privacy-enhancing solution. Recent literature, however, introduced new attacks on privacy that target LN specifcally (Béres et al., 2020; Kappos et al., 2020; Romiti et al., 2020). One of these attacks is the Balance Discovery Attack (BDA) (Herrera-Joancomartí et al., 2019; Tikhomirov et al., 2020; van Dam et al., 2020). The BDA reveals the individual balances of a channel by using defective payments to probe for channel balances.

Approximate differential privacy (Chan et al., 2011) has been proposed (van Dam and Abdul Kadir, 2022) as a way to mitigate BDA's, but that technique has limited effect on mitigating BDA when it comes to detecting larger trends in the flow of funds through LN and also requires locking up a considerable amount of capital. In this paper we offer a second type of mitigation that suffers from neither.

Our contributions We suggest a way of splitting up en route payments in LN at intermediary hops along the payment path, that is compatible with LN today. We coin the term Payment Splitting and Switching (PSS) for this technique and show that this concept is feasible with a plugin for `Core Lightning`[1][2]. We evaluate the impact of PSS on the information gain through Balance Discovery Attacks (BDA's) in a network simulation based on a real-world network snapshot. We show a reduction in information gain for direct probing of single channel hops of 50% and 62% for remote probing. PSS also demands a more involved BDA algorithm that has an increased time complexity ($O(n^c)$) making a BDA at network-wide scale intractible with off-the-shelve hardware.

2 Background

2.1 Lightning Network

LN enables trustless, off-chain Bitcoin payments. This means that individual transactions are not be broadcasted to the blockchain. Participants operate an LN *node* identifiable through by a public key, the *node id*. The LN nodes form a a peer-to-peer (P2P) network. Apart from communicating with its peers, an LN node also communicates with a Bitcoin node either to broadcast transactions or to retrieve information on broadcasted transactions.

An LN node starts by connecting to one or more peers. This allows it to then initiate the opening of a *channel* between it and a peer. The two peers open a channel by locking bitcoins in a 2-of-2 *multi-signature* contract. The locked coins reflect the opening balance of the channel. This transaction is called the *funding transaction*. The total amount of locked coins locked represent the channel's *capacity*, the maximum value that can be transferred. After a set number of confirmations on the blockchain, the channel is considered open.

With an open channel between them, two peers can now transact with each other. With every transaction the balance of the channel is updated to reflect the latest state of the channel. A state is specified in a *commitment transaction*. This is a normal Bitcoin transaction that has the funding transaction as its input and devides the coins according to the latest balance. During normal

[1]The code for this plugin is available at https://github.com/gijswijs/plugins/tree/master/pss
[2]A screencast showing the plugin in action is available at https://asciinema.org/a/520416

operation of the channel, the commitment transaction will not be broadcasted to the blockchain. Only when one or both peers decide to close the channel, the final state is broadcasted.

The outputs of the commitment transaction representing the balance of the payment channel are called *hash time-locked contracts* (HTLCs). One peer can spent the HTLC by providing the pre-image of a given hash while the other peer can spent the HTLC after a timeout. Imagine a scenario where Alice wants to buy a widget from Bob for an agreed price of x. Bob creates the pre-image r, which is a random number. The hash of the pre-image, the *payment hash* $H(r)$ is sent to Alice in a message called an *invoice*. Alice can now set up the payment using an HTLC. She offers an HTLC contingent on $H(r)$ for the correct amount x to Bob, spendable for him with the pre-image. He needs to do so before the timeout, because after the timeout Alice can claim the payment herself. This is what happens if Bob does not know the pre-image r or if the amount offered is not the amount agreed upon. In those cases Bob will let the timeout expire so that Alice can reclaim the payment. An HTLC is resolved after either Bob or Alice claims the payment. The next commitment transaction will contain an updated state where the balances reflect the resolved HTLC. A payment channel can have multiple unresolved HTLCs at any time during normal operation.

LN operates under source-based routing: The sender finds a route to the receiver, based on its knowledge of the network graph. It is not necessary to find a direct route to the receiver. Using HTLCs a route can be set up that uses one or multiple intermediary nodes, a so called multi-hop route. If the above scenario would play out in a multi-hop route, it would again be Alice who would offer the first HTLC but in this case to the first intermediary node. The HTLC would be contingent on $H(r)$, which the first intermediary obviously does not know. The first intermediary offers an HTLC to the next node in the route, which it can find in the information that Alice sent to it. It does so using a shorter timeout than Alice's, so that once it learns the pre-image it still has time to claim the payment from the HTLC that Alice offered. This process repeats itself until the receiver Bob is reached. Bob does know the pre-image, so he can claim the payment. In doing so he shares the knowledge of the pre-image with the node before him. This now unwinds back to first intermediary node who can claim the payment from the original sender Alice. It is important to note that payments like this are *atomic*. They either succeed, or they do not succeed at all. The intermediary nodes are incentivized to relay the payments with routing fees, a small difference between the HTLC being received and the HTLC being offered by the relaying node. The sender takes these routing fees into account when creating the route.

For finding a route to any other node that is not a direct peer, it is paramount that a node has knowledge of the network graph. LN uses its P2P network for a gossip protocol in which new nodes and channels are announced. A node can also request its peer to share its knowledge about the graph. Two nodes opening a channel can decide to refrain from announcing the channel, in which case the channel remains private. Because two peers with a payment channel do not broadcast each and every payment that is made in the channel (Malavolta et al., 2019), the ratio of actual payments versus broadcasted transactions on the blockchain is heavily in favour of the former. This is where the scalability benefit of LN comes from, and multi-hop payments amplify this even more.

Misbehavior in LN is punished with a penalty. E.g., a node might try to close a channel by broadcasting a state other than the latest, because it is beneficial to the misbehaving node for it represents a state where the balance was still more in favour of the misbehaving node. To prevent this from happening, nodes exchange *commitment revocation private keys* of the stale state while establishing a new state. The outputs of a commitment transaction that pay out to the holder of said transaction, are spendable after a time out, but there's a way to spent those outputs directly, namely by using a private key, the commitment revocation private key. At any time both channel partners hold the commitment revocation private keys for all the *old* commitment transactions the other partner holds, so when a node broadcasts an old state, its channel partner can punish this act by claiming all channel funds directly using the commitment revocation private key.

2.2 Balance Discovery Attack

Finding a route to a receiving node is a process of trial and error. Because the sender does not know the actual balances of the hops along the route, it can inadvertently select a route where a channel does not have enough liquidity to relay the payment. In these cases the relaying node will return an `InsufficientFunds` or `TemporaryChannelFailure` error. The sender will now have to find another route to the receiver, and try again. It does so until it finds a route that has enough liquidity to relay the payment. This process of finding a route is called *probing*.

In the Balance Discovery Attack (BDA) the attacker probes a route, but instead of using the payment hash of an actual invoice, it uses a random payment hash created by the attacker. When the attacker probes a route with enough liquidity, the receiver now returns an `UnknownPaymentHash`, since it receives a payment hash it did not create. This gives the attacker the information that the route has enough liquidity for relaying the payment. It tries again with a higher amount, until it receives an error that shows that a channel along the route does not have enough liquidity. The attacker chooses amounts according to a binary search algorithm, which allows it to identify efficiently the exact balance of a channel with a precision of up to one millisatoshi.

Herrera-Joancomartí et al. (2019) introduced the basic BDA, that had the potential of disclosing 89% of all public channels. van Dam et al. (2020) increased that to 98% by introducing two-way probing. Probing multiple channels concurrently (Biryukov et al., 2022; Rahimpour and Khabbazian, 2022; Tikhomirov et al., 2020) made it feasible to mount a network-wide probing attack.

2.3 Parallel channels & the generalized geometrical probing model

LN allows for peers to open up more than one channel between them. This could be warranted if a channel between two peers is depleted on one side. The channel partner on that side could open up a new channel, which would allow it to send payments again, while retaining the ability to receive payments through the other channel. This concept of parallel channels was not properly addressed in earlier BDA algorithms. Biryukov et al. (2022) tackled this and in doing so introduced the concept of the generalized geometric probing model.

The geometrical model is a model where all parallel channels in a N-channel hop are represented by an n-dimensional (hyper-)rectangle. This (hyper-)rectangle represents the result space of all possible balance vectors with one of its vertices at the origin and the other vertex at the coordinate represented by the exact balances of each of the parallel channels. A probe removes a (hyper-)square from the result space, either from the origin point or from the opposite vertex, depending on the probing direction. If the probe failed the result space is contained by the (hyper-)square (an upper bound) if it succeeded all possible points in the result space are outside the (hyper-)square (a lower bound).

In this model probing becomes a model of chipping away at the original n-dimensional (hyper-)rectangle until the smallest set of possible balance vectors remains. This does not necessarily mean that we get a set of cardinality 1. In general if $n > 1$ the set can not be shrunk to a single point.

2.4 Onion Routing with Sphinx

LN uses onion routing (Reed et al., 1998) to ensure privacy along a payment path. The encryption scheme employed by LN is called Sphinx (Danezis and Goldberg, 2009). For each node in the payment path, the Sender creates a shared secret.

The Sender starts by creating a session key, a random 256 bits bitstring. This session key *is* the first ephemeral secret key. The Sender then shares the first ephemeral *public* key with the first node along the payment path. It does so by sending it unencrypted in the onion header. Now both the Sender and the first node can create the first shared secret key using Elliptic Curve Diffie-Hellman (ECDH). To this end, the Sender uses the first ephemeral secret key and the `node_id` of the first node, which is its public key. The first node uses its secret key and the ephemeral public key it received from the Sender.

For the next node, the Sender creates a new ephemeral secret key. It does so by creating a tweak factor. This tweak factor is the hash of the concatenation of the first ephemeral public key and the first shared secret. Multiplying the tweak factor with the first ephemeral secret key gives the second ephemeral secret key. The Sender shares the second ephemeral *public* key with the second node in the payment path, again unencrypted in the onion header. Now both the Sender and the second node can create the second shared secret key. This process is repeated for each node along the payment path.

The Sender now creates the onion, encrypting each layer of it using the shared secret keys. The outermost layer is encrypted using the first shared secret key, the second layer using the second shared secret key and so on. This encapsulation in layers of encryption achieves that each node along a payment path can only "peel" away a single layer, exposing the next destination. So each node is only aware of the node from which it got the onion and the node to which the remaining layers of the onion have to be sent.

3 Method

3.1 Payment splitting and switching

The onion routing employed by LN prevents nodes in a payment path from knowing anything more than its predecessor and its successor in the payment path. It also prevents an intermediary node to change the payment path to use an alternative route to the destination, because there is no way for an intermediary node to know the destination of the payment. In that sense, LN acts as a circuit switched network.

It is that property that is leveraged in a BDA: The Sender chooses a payment path and does that so that a single hop can be targeted for probing.

We suggest a way for intermediary hops to split a payment into multiple smaller payments and send those smaller payments over alternative routes to the next hop. This is called payment splitting & switching (PSS). It works as follows:

Upon opening a channel both nodes signal that they support PSS. Let's assume that Alice and Bob have opened such a channel. Now, upon receiving a payment that needs to be relayed, Alice decides she wants to split the payment into two separate parts. By definition, one part will have to follow the original route as intended and the other part will be sent to Bob via an alternative route. Alice will start by forwarding the original onion as normal, but she will commit to an HTLC that carries an amount lower than the intended amount. Bob, upon receiving the amount, sees that the amount is insufficient to cover the amount that he needs to forward and the fees that he expects to receive for forwarding, but instead of failing the HTLC, he waits for an agreed-upon time, because Bob also supports PSS. Alice now sends a new payment to Bob for the remaining amount, using the same payment hash. Bob is the recipient of this payment, so Alice can create the entire onion, but Bob can only claim the payments once he learns of the preimage used to create the payment hash. Once Bob receives this second payment he is economically incentivized to forward the first payment. He does so by setting up the next HTLC and forwarding the remaining onion of the first payment, which is the original onion containing the intended recipient of the payment.

The above method creates a localized packet switching of sorts where a node can use any route available to the next node, to forward a payment in separate parts.

3.2 Probing of PSS hops

Assuming an attacker knows it is probing a PSS channel from Alice to Bob, it needs to change its conceptual model of what it is probing. Instead of knowing the route of a payment, it now needs to take into account that a payment can go through each and any of the possible routes from Alice to Bob at once. If Alice and Bob have other, parallel channels (Biryukov et al., 2022) then those qualify as alternative routes, but even routes via intermediary hops qualify as alternative routes. So instead of probing a single channel, an attacker is now probing the total liquidity of Alice in the direction of Bob.

If we take the probing model for parallel channels (Biryukov et al., 2022) as a starting point, we can think of each possible route as a single channel in a hop. Originally a hop referred to all the channels shared by a pair of adjacent nodes, but now we take all possible routes from two adjacent nodes into account. For clarity, we will refer to such a hop as a routing-hop.

The capacity of a route is defined by the channel with the lowest capacity of all channels in that route. The balance of a route is defined by the channel with the lowest balance in the direction of the route. This need not be the same channel. A routing-hop with n routes is defined by its route capacities $C = [c_1, ..., c_n]$ and balances $B = [b_1, ..., b_n]$. Balances are assumed to be in the direction of the node with the alphanumerically smaller ID. We define this direction as $dir0$. $dir1$ is defined as the opposite direction.

This definition collapses with the original definition for hops with parallel channels (Biryukov et al., 2022), in the case where the set of parallel channels equals the set of routes between two adjacent nodes.

There is a noteworthy difference between channels in a hop and routes in a routing-hop: If a channel i is enabled in both directions, the forwarding ability of a hop in $dir0$ is determined by its balance b_i, and its forwarding ability in $dir1$ is determined by $c_i - b_i$. In a *route* that is enabled in both directions, however, that relationship between capacity in $dir0$ on one side and balance and capacity in $dir1$ on the other side doesn't necessarily hold. The balance in $dir0$ and $dir1$ can be anything from 0 to the route capacity c_i and there need not be any relationship between the two. To fix this we model a bidirectional route as two unidirectional routes, for which we calculate the balance such that $b_i^{dir1} = c_i - b_i$ holds, but is meaningless in either $dir0$ or $dir1$, since the route is not enabled in that direction.

E^d is defined as the set of routes enabled in direction d, where $d \in \{dir0, dir1\}$. The forwarding ability of a routing-hop is set by the sum of balances of all routes enabled in a given direction, where h stands for the forwarding ability in $dir0$ and g in $dir1$.

$$h = \sum_{i \in E^{dir0}} b_i$$
$$g = \sum_{i \in E^{dir1}} (c_i - b_i)$$

Probing reveals information about h or g, and in some cases about individual balances. The attacker maintains the current lower and upper bound for all of them: $h^l < h \leq h^u$, $g^l < g \leq h^u$ and $b_i^l < b_i \leq b_i^u$. Before probing those bounds are set to

$$\begin{aligned}
h^l &= g^l = -1 \\
\forall i \in \{1, \dots n\}.\ b_i^l &= -1 \\
\forall i \in \{1, \dots, n\}.\ b_i^u &= c_i \\
h^u &= \sum_{i \in E^{dir0}} c_i \\
g^u &= \sum_{i \in E^{dir1}} c_i
\end{aligned}$$

For routes that consist of just one bidirectional channel the relationship between balance (in $dir0$) and forwarding ability in $dir1$ *is* meaningful. We can think of this forwarding ability as the balance in $dir1$ and this balance also has upper and lower bounds that the attacker can maintain when probing from $dir1$. These upper and lower bounds of the balance in $dir1$ relate to the lower and upper bounds of the balance in $dir0$ respectively:

$$\begin{aligned}
b_i^l &= c_i - b_i^{u_{dir1}} - 1 \\
b_i^u &= c_i - b_i^{l_{dir1}} - 1
\end{aligned}$$

It should be pointed out that all lower bounds are strict and all upper bounds are non-strict. Let F be the set of all possible values for B, considering the knowledge of the attacker. F is then the n-fold Cartesian product defined by $F = X_1 \times \ldots \times X_n = \big\{(x_1, \dots, x_n) | x_i \in (b_i^l, b_i^u] \text{ for every } i \in \{1, \dots, n\}\big\}$.

Assuming all available routes are used to their potential for a probe leads to the following updates of those bounds depending on whether the probe was successful or a failure:

Successful probe of amount a in $dir0$

A successful probe in $dir0$ has a direct impact on the lower bound h^l, since at least the probed amount has to be available for forwarding payments in $dir0$. Lower bounds being strict, the bound is adjusted to the amount probed minus one.

A successful probe potentially gives an attacker knowledge about the lower bounds of the balances of the possible routes. The intuition is that if we look at a single route out of all routes enabled in $dir0$, and the combined potential for all *other* routes enabled in $dir0$ (the sum of their b^u's) is not enough to relay the probed amount, then the remaining amount of the probe would have to be relayed through this single route. In that case, we can update the lower bound of the balance for that single route. We can do this for every single route in the set of routes enabled in $dir0$.

A successful probe can also be used to update the upper bound g^u. But in this case, we have to look at whether all routes enabled in $dir0$ are also enabled in

$dir1$ ($E^{dir0} \subseteq E^{dir1}$). If this is the case, then we can state that the total amount of the probe is not available for forwarding payments from "the other side", so we can update the upper limit g^u by deducting the probed amount from the total capacity of all routes enabled in $dir1$. If some of the routes enabled in $dir0$ are not enabled in $dir1$ we cannot make such a strong statement, so we resort to using the sum of upper bounds of forwarding capability in $dir1$ of all routes enabled in $dir1$, using the relationship between this upper bound and the lower bound of the balance in $dir0$, as explained above.

$$\begin{aligned}
h^l &= a - 1 \\
\forall i \in E^{dir0}.\ b_i^l &= \max\left(a - \sum_{\{j \in E^{dir0} : j \neq i\}} b_j^u, 0\right) - 1 \\
g^u &= \begin{cases} -a + \sum\limits_{i \in E^{dir1}} c_i & \text{if } E^{dir0} \subseteq E^{dir1} \\ \sum\limits_{i \in E^{dir1}} c_i - b_i^l - 1 & \text{otherwise} \end{cases}
\end{aligned}$$

Failed probe of amount a in $dir0$

A failed probe in $dir0$ has a direct impact on the upper bound h^u since the probed amount a cannot be available for forwarding payments in $dir0$. The upper bound is adjusted to the amount probed minus one.

Likewise, a failed probe can potentially be a reason to adjust the upper bounds of the balances of the possible routes. Consider the case where we look at a single route of all routes enabled in $dir0$, and the total amount of which we are certain that it is available for all the *other* routes enabled in $dir0$ (the sum of their b^l's) is not enough to relay the probed amount, then the remaining amount of the probe would have to be relayed through this single route and would have failed. So we can update the upper bound of the balance for that single route. We can do so for every single route in the set of routes enabled in $dir0$.

A failed probe can also be used to update the lower bound g^l. If we look at all routes enabled in $dir0$ that are bidirectional ($E^{dir0} \cap E^{dir1}$), we know for sure that the combined capacity of those bidirectional routes minus the probed amount increased by one, has to be on the "other side", otherwise the probe would have succeeded.

$$\begin{aligned}
h^u &= a - 1 \\
\forall i \in E^{dir0}.\ b_i^u &= \min\left(a - 1 - \sum_{\{j \in E^{dir0} : j \neq i\}} (b_j^l + 1), c_i\right) \\
g^l &= \max(-a + \sum_{i \in E^{dir0} \cap E^{dir1}} c_i, -1)
\end{aligned}$$

Successful probe of amount a in $dir1$

A probe from the other direction would lead to similar, but mirrored adjustments to the upper and lower bounds, but since we keep the balance of a route and its upper and lower bounds by definition in $dir0$ we need to adjust for this fact.

A probe in $dir1$ would lead to the following results:

$$
\begin{aligned}
g^l &= a - 1 \\
\forall i \in E^{dir1}.\ b_i^u &= \min\left(c_i - a + \sum_{\{j \in E^{dir1} : j \neq i\}} c_j - b_j^l - 1, c_i\right) \\
h^u &= \begin{cases} -a + \sum\limits_{i \in E^{dir0}} c_i & \text{if } E^{dir1} \subseteq E^{dir0} \\ \sum\limits_{i \in E^{dir0}} b_i^u & \text{otherwise} \end{cases}
\end{aligned}
$$

Failed probe of amount a in $dir1$

Finally, a failed probe in $dir1$ would lead to the following adjustments:

$$
\begin{aligned}
g^u &= a - 1 \\
\forall i \in E^{dir1}.\ b_i^l &= \max\left(c_i - a + \sum_{\{j \in E^{dir1} : j \neq i\}} (c_j - b_j^u), -1\right) \\
h^l &= \max(-a + \sum_{i \in E^{dir0} \cap E^{dir1}} c_i, -1)
\end{aligned}
$$

3.3 Threat model

We work under the assumption of a computationally efficient adversary who can run some fraction of the nodes in LN. The adversary can make those nodes diverge from the LN protocol, but all other nodes keep exactly to the LN protocol and do not leak information through side-channels. The adversary attacks off-path value privacy (Malavolta et al., 2017), which is the concept that a payment should remain hidden from nodes that do not take part in the payment route, but not from the nodes that do.

The adversary is computationally bounded (i.e., probabilistic, polynomial time) and can not break the underlying cryptographic primitives.

3.4 Geometrical model of PSS probing

We can capture the probing of PSS hops in a geometrical model. An n-route hop can be seen as an n-dimensional (hyper-)rectangle R, where the sides run parallel to the axes. Each side corresponds to one possible route. Along the i^{th} dimension, R is defined by the point $[0, c_i]$. The origin is one of the vertices of R and the point $[c_1, \ldots, c_n]$ is the vertex diagonally opposite of the origin. Each lattice point inside R corresponds to a possible balance vector. The size of the possible result set is defined by the cardinality of the set of lattice points inside R.

A probe of amount a cuts the hyperrectangle along a hyperplane orthogonal to the all-ones vector $\vec{1}$. If the probe fails all points from the opposite vertex (in *dir0*) or the origin (in *dir1*) to the hyperplane, including all points in the hyperplane itself are removed from the result set. A failed probe results in a new upper bound, and since upper bounds are non-strict, the points in the hyperplane itself should be removed from the result set. If the probe succeeds all points on the opposing side of the hyperplane, are removed from the result set. Any adjustments in the bounds for the individual balances of the routes in the hop are captured by changing the dimensions of R. Specifically for dimension i, R will be defined by $\left[b_i^l + 1, b_i^u\right]$.

Figure 1 shows a 2-dimensional case with $c_1 > c2$ and a failed probe in *dir0* of amount a. The dashed diagonal line represents the hyperplane running through all coordinates in the first quadrant of the Cartesian plane (including $[0, a]$ and $[a, 0]$) for which the sum of its coordinates equals a. Where the line intersects with R, the lattice points on the line correspond to a possible balance vector v for which $\sum_{i=1}^{n} v_i = a$. Since it is a failed probe, the upper right corner of the rectangle R is removed from the possible result set, leaving the colored polygon representing the attacker's current best estimates.

Figure 2 shows a new probe in *dir0*, but this time it is successful. this results in the lower right corner of the rectangle R being removed, again leaving the colored polygon representing the attacker's current estimates.

Finally in figure 3 we see a state of probing where the current h^l and h^u necessitate an update to b_i^l and b_i^u

Convex polytope described through h-representation

We can describe the polygon (or polytope when the number of dimensions is larger than 2) representing the attacker's current estimates as a system of inequalities. Let P be that polytope. Its system of inequalities is described by $Ax \leq b$ where $A \in \mathbb{Z}^{m \times d}$, $A = (a_{ij})$, and $b \in \mathbb{Z}^m$. Assume that we are probing a routing-hop with two routes with capacity $c_1 = 160$ and $c_2 = 100$, similar to the example used in Figure 1, 2 and 3. We are probing from *dir0* with an amount of 80. We can now describe the polytope P as follows. $P = \{(x, y) : x \leq 160, y \leq 100, x + y \leq 80, x \geq -1, y \geq -1\}$. Thus

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}, b = \begin{pmatrix} 160 \\ 100 \\ 80 \\ 0 \\ 0 \end{pmatrix}$$

Using Barvinok's algorithm (Barvinok, 1994) we can find a polynomial time algorithm f for counting lattice points in a convex polytope for any polytope in n-dimensional Euclidean space $\mathbb{R}^n$. The computer package `LattE`, contains an implementation of A. Barvinok's algorithm (De Loera et al., 2004). We use this

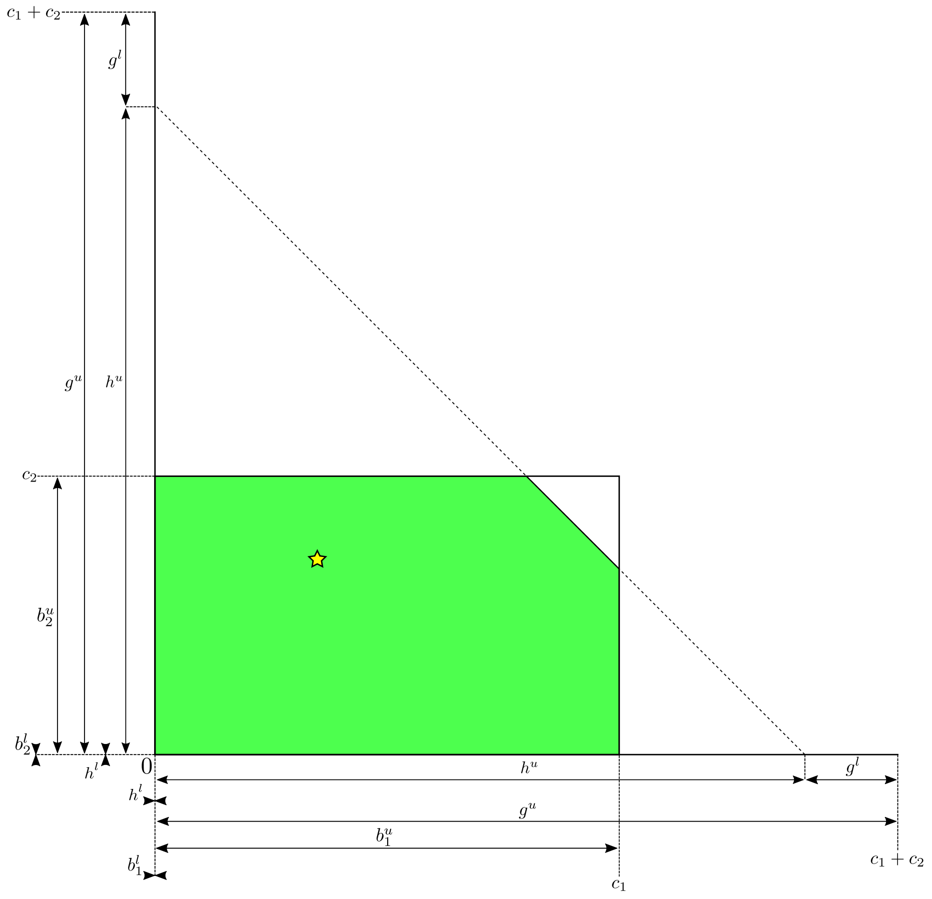

Figure 1: Failed probe in a 2-dimensional rectangle, resulting in a new upper bound.

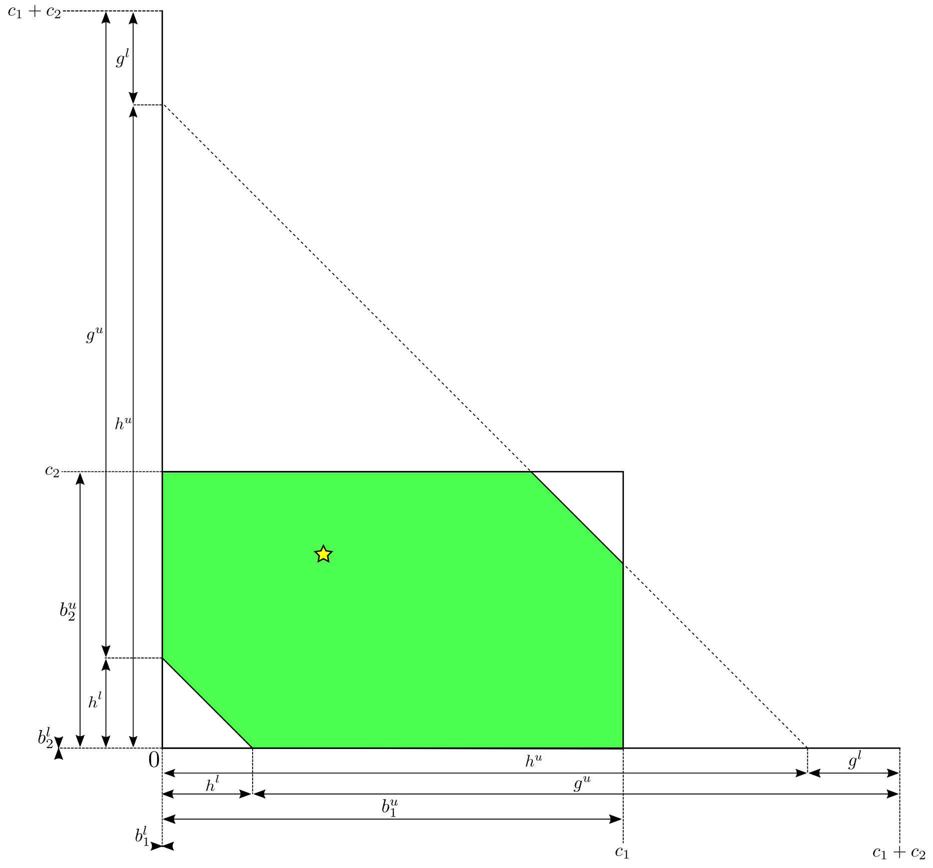

Figure 2: Successful probe in a 2-dimensional rectangle, resulting in a new lower bound.

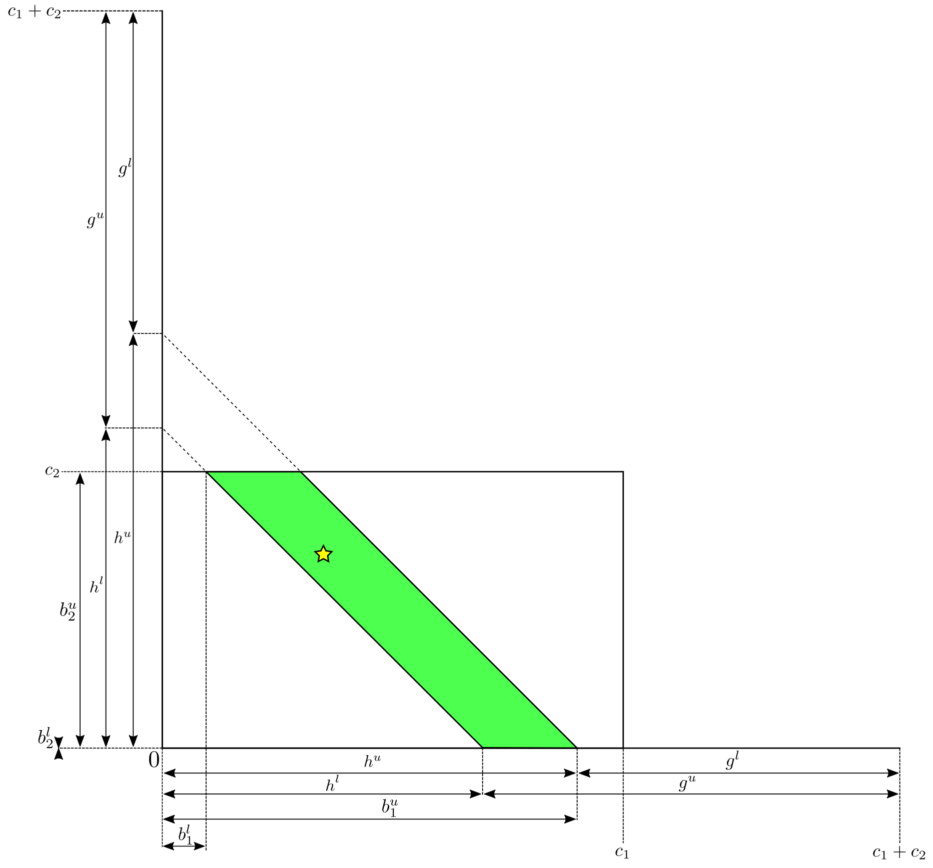

Figure 3: Probing state in a 2-dimensional rectangle, resulting in updated balance bounds for channel 1.

package to determine the cardinality of the set of possible balance vectors given by $f(P)$.

Information gain

We can measure the success of a probe by the amount of information gained through a successive series of probes. Information gain is calculated similar to Biryukov et al. (2022), by comparing the attacker's uncertainty U before and after the attack. The attacker's uncertainty is defined as the number of bits required to encode the cardinality of the possible result set. This is calculated by $log_2(f(P_i))$ where P_i is the polytope describing the attacker's current estimates after the i^{th} probe. After n probes, the uncertainty changes from $U_{before} = log_2(f(P_0))$ to $U_{after} = log_2(f(P_n))$. In case of a set of target hops T, we can sum uncertainties. The Information Gain then becomes:

$$I = 1 - \frac{\sum\limits_{t \in T} U^t_{after}}{\sum\limits_{t \in T} U^t_{before}}$$

Lightning Network Probing Simulator

The LN Probing Simulator was introduced by Biryukov et al. (2022), it uses a snapshot of the Lightning Network to recreate the actual graph at a specific time. Because the actual balances are hidden, they cannot be part of a snapshot, so the simulator samples a balance for each channel from a uniform distribution between 0 and the capacity of that channel. The simulator operates in two modes: direct probing and remote probing.

Direct probing means that the attacker opens a channel to one of the two nodes on either side of the channel being probed. Probes are then sent through a 2-hop path. Since all probes reach the target, direct probing is efficient, but it incurs on-chain fees for each channel being probed. It also requires the participation of the victim when opening a channel.

Remote probing means probing through a multi-hop path where the attacker isn't directly connected to one of the two nodes on either side of the channel being probed. By choosing wisely whom to connect to, an attacker can drastically reduce the amount of on-chain fees required to probe multiple target hops. It also allows for gathering data on the intermediary hops as well, which can aid in forming a network-encompassing view of balances Rahimpour and Khabbazian (2022). The drawback of remote probing is that there is a chance of probes not reaching their target, because of intermediary hops failing or lacking enough balance to relay the payment.

We forked the original simulator and created a version that integrated with the `LattE` software[3]. We reproduced the probing results of Biryukov et al. (2022) and compared them to probing PSS hops using the same LN snapshot. This is a snapshot created using a `Core Lightning` node on 2021-12-09. This snapshot contains 17068 nodes and 78076 channels which is in line with public network explores at the time.

[3]The repository is at https://github.com/gijswijs/ln-probing-simulator

We establish information gain for both probing methods (direct probing and remote probing) and in two contexts (PSS and non-PSS). For each combination, we simulate attacks on target hops with 1 to 5 parallel channels. From the network graph, we randomly select 20 target hops with the required amount of parallel channels and then simulate the probing. We repeat this 50 times and average the results. In total, we run $5 \times 20 \times 50 = 5000$ probes for each of the 4 combinations.

4 Results

4.1 Achieved Information Gain

When we look at the results (fig. 4) we see that non-PSS probing, both direct and remote, are exactly in line with the results from Biryukov et al. (2022). This was expected, since we replicated the exact method of probing from that paper. To be able to compare our results with theirs, we have looked at the number of channels in the original target hops. So a target hop with one single channel in the non-PSS settings would be compared to a target hop with one single channel in the PSS setting. This does explicitly not mean that there is a single channel in the *routing-hop.* A routing-hop consists of *all* possible routes from two adjacent nodes. So what figure 4 shows us is the difference between probing a hop assuming no PSS, versus probing a similar hop assuming PSS.

The results are very pronounced for both direct and remote probing. With PSS, the information gain for direct probing of a single channel hop drops from 0.98 to 0.49 ($p < 0.0001$) and for remote probing it drops from 0.91 to 0.35 ($p < 0.0001$). When the number of parallel channels in the target hop increases the information gain decreases in non-PSS and PSS probing alike making the impact of PSS relatively smaller, but the difference remains distinct and significant. With PSS, direct probing a 5-channel hop drops from 0.27 to 0.22 ($p < 0.0001$) and remote probing drops from 0.16 to 0.13 ($p < 0.0001$).

Moreover, the results for PSS-probing are upper bounds of the actual information gain. Since counting lattice points is computationally expensive, even when using Barvinok's algorithm (Barvinok, 1994), we limited the total amount of routes in a routing-hop to a maximum of 9. Potential target hops with more routes in their routing-hop were disregarded. Higher dimensionality polytopes lead to lower information gain, hence the results of PSS-probing are an upper bound.

5 Discussion

Our results show that PSS can greatly reduce the extraction of balance information from the BDA. It can be (part of) a mitigation strategy that impedes potential attackers from mounting a BDA.

5.1 Limitations

Network simulations like the Lightning Network probing simulator have certain limitations, amongst which are accuracy and validation (Rampfl, 2013). The

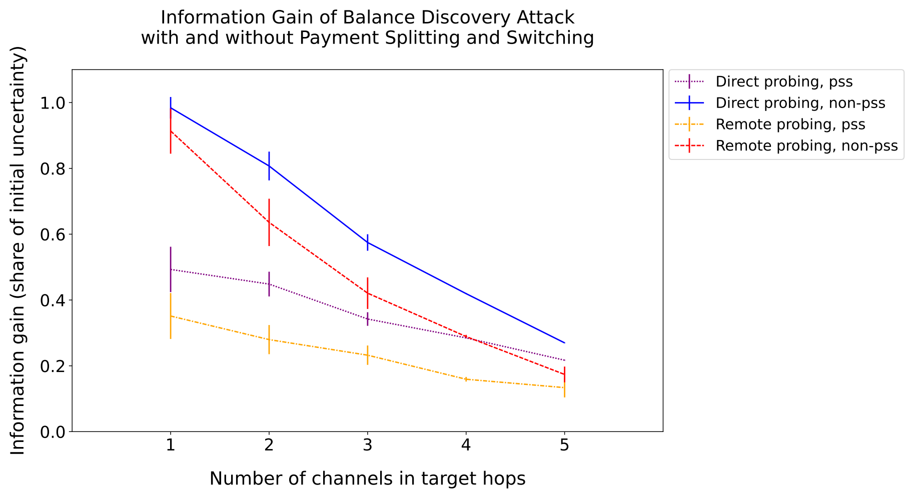

Figure 4: Information gain for direct and remote probing in PSS and non-PSS context

simulator has not been validated and certain aspects of the network are absent from the simulation, e.g. connectivity of nodes, routing policies, balance shifting between probes, in-flight payments and other factors (Biryukov et al., 2022). We also do not take into account LN routing fees.

5.2 Knowledge of PSS being employed

In our analysis, we assume the attacker has full knowledge of whether PSS is employed. But, the way we have set up our plugin, the plugin announces its support for PSS at `init`, which means it only shares its support with direct peers. The attacker does not know for sure whether PSS is employed or not. This makes for an extra factor of uncertainty. The only way for an attacker to know whether PSS is supported for a specific channel is to connect with both peers on either side of the channel and see if they both support PSS. Even then an attacker does not know whether splitting is used, because the forwarding node is free to use PSS fully, partly or not at all. Although we have confirmed the feasibility of payment splitting with a plugin, we are agnostic as to whether PSS should be an optional addition by the use of a plugin or an integral part of the LN protocol. Even if it were to be an integral part of the LN protocol, in practice an attacker will not have full knowledge of the usage of PSS.

5.3 Overlapping hops in alternative routes

A helpful mental representation of probing in a PSS context is that instead of probing the balance of a channel, e.g. the balance of a channel between Alice and Bob, the attacker is now probing the total liquidity of Alice in the direction of Bob. At first glance it might seem that an attacker would still be able to monitor liquidity over time like an attacker could with channel balances in the original BDA setting (Herrera-Joancomartí et al., 2019; van Dam et al., 2020) and deduce individual payments in LN. Overlapping alternative routes make this less tractable than it might seem at first sight. Consider Alice and Carol who have one direct channel between them and one alternative route over a single intermediary hop Bob. Now consider Dave and Carol who also have one direct channel between them and one alternative route over the same intermediary hop Bob. The alternative routes share a single hop between them, namely the channel between Bob and Carol and the balance in that single hop in the direction of Carol is the lowest balance in both alternative routes (defining the balance of those routes). The direct channels between Alice and Carol and between Dave and Carol are depleted in the direction of Carol (fig. 5). We assume a powerful adversary that can mount a network-wide BDA between every single payment. If Alice now pays 25 to Carol, this powerful adversary would detect a change in liquidity in 7 out of the 12 possible permutations of nodes (tbl. 1). Even in this toy example disambiguating the payment becomes quite hard. Disambiguating the payment becomes completely intractable except for the smallest of graphs and is considered out of scope for this paper.

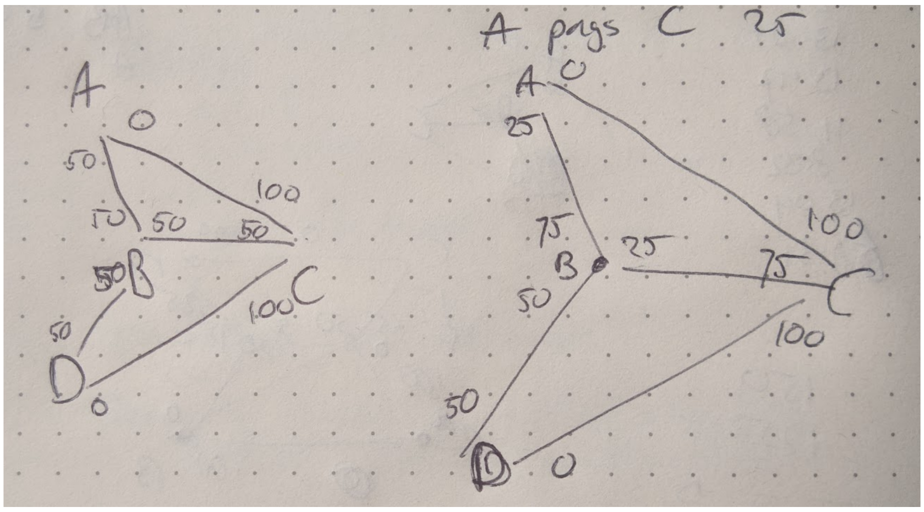

Figure 5: Detecting liquidity after a single payment between t_0 and t_1

TODO: Create the proper figure

Table 1: Changes in liquidity for all permutations of nodes in a simple graph

liquidity direction	liquidity at t_0	liquidity at t_1
AB	50	25
AC	50	25
AD	50	25
BA	100	100
BC	50	25
BD	100	75
CA	150	175

liquidity direction	liquidity at t_0	liquidity at t_1
CB	150	150
CD	150	150
DA	50	50
DB	50	50
DC	50	25

5.4 PSS as a tool for approximate differentially private payment channels

In earlier work, we have studied the feasibility of approximate differentially private payment channels (DP-channels) in LN (van Dam and Abdul Kadir, 2022). This approach made use of an additional noise payment through private channels to hide the value of the actual payment. The actual payment and the noise payment are two separate payments, and a powerful enough adversary could mount a BDA in between those two payments and render the noise payment moot. To prevent this, the node making both payments does not relay any other payments in between those two payments to force the two payments to act as if they were atomic. With PSS we can make the actual payment and the noise payment in parallel as opposed to consecutive, as they are both contingent on the same payment hash. This makes both partial payments atomic as a whole, without requiring to ignore relay requests. It does require a change to the current PSS plugin because the noise payment is circular, meaning that it returns to the sender, and the current plugin does not support that.

As we have noted in (van Dam and Abdul Kadir, 2022), if multipath payments (Di Stasi et al., 2018; Wang et al., 2019) are supported we can forego negative noise payments by having a payment that consists of a payment over the public channel for the amount of the actual payment minus the noise and a payment over the private channel for the amount of the noise. In this scenario, there is no circular noise payment needed. PSS is a new type of multipath payment that would support exactly this type of scenario. This is an obvious improvement to DP-channels, achievable with PSS, that would reduce the cost of implementing such channels.

It remains an open question whether PSS combined with DP-channels allows for stronger bounds on ϵ, δ)-differential privacy (van Dam and Abdul Kadir, 2022). It is also an open question if PSS would allow for DP-channels without the use of private channels, which would further reduce the cost of implementing DP-channels and improve the feasibility.

5.5 PSS and enhanced probing.

Enhanced probing is a type of probing where other techniques or aspects of LN are leveraged to improve the results of probing itself (Biryukov et al., 2022). There are two types of enhanced probing: jamming-enhanced probing and policy-aware probing.

Jamming is a denial-of-service attack (EmelyanenkoK, 2017; Pérez-Solà et al., 2020). The adversary sends payments to itself or to another node it controls and delays claiming them. In doing so liquidity and payment slots are locked up along the route resulting in congestion. By jamming channels, an attacker can potentially reduce the number of available channels for relaying (or in the case of PSS, available routes) in a hop to one. When we are certain that a payment is relayed through a single channel, a BDA allows for exact disclosure. PSS does not prevent jamming, but it does allow for redirecting partial payments over alternative routes. An attacker would have to take these alternative routes into account and jam them too. Jamming demands considerably more resources when the attacker has to take PSS into account. Also, as noted by Biryukov et al. (2022), this description of jamming makes a few simplifying assumptions. The biggest assumption being made is that an attacker can jam specific channels, and leave a single particular channel unjammed. In practice, there is no way to achieve that, since any node is free to decide which channel (or route) to use for relaying a payment. So jamming-enhanced probing is more difficult than is obvious at first glance to begin with and PSS adds significantly to this difficulty.

Policy-aware probing is a type of probing that tries to target specific channels by tuning certain parameters such as timeouts and fees. For instance, fee-aware probing, which is a specific kind of policy-aware probing, offers fees for relaying such that only the channel that accepts the lowest fees can forward the payment. This only works if the fee requirements of the channels in a hop are set differently. PSS does not prevent this type of enhanced probing but again, PSS does require the attacker to consider all the alternative routes, making it less likely that a single, specific channel can be targeted.

5.6 Computational cost

Introducing A. Barvinok's algorithm in the BDA algorithm increases the Big-O time complexity. The original BDA is a binary search with a time complexity of $O(\log n)$. A. Barvinok's algorithm bumps the time complexity up to $O(n^c)$. This is why we had to limit the routes in a routing-hop to a maximum of 9, to keep the running time of the simulation manageable at roughly four hours on an Intel Core i7-8850H 2.60GHz CPU running Ubuntu Linux. During a simulation 5000 hops are targetted. The total amount of hops in the LN snapshot used is 74022. Assuming PSS in the entire LN network, this would bring the runtime of a network-wide BDA on similar hardware to nearly 60 hours. This makes advanced methods such as payment inference unfeasible. The payment discovery algorithm described by Kappos et al. (2020) does not report on the success rate for network balance snapshots spaced 60 hours apart, but snapshots with an interval of 32,768 seconds (over 9 hours) allow for a success rate of less than 5% for revealing payments. This is assuming perfect balance snapshots, which is impossible to begin with if PSS is employed throughout LN.

5.7 Additional benefits of PSS

Although we have developed and analyzed PSS as a mitigation strategy against BDA, it has side effects that could be considered beneficial to other aspects of LN. The aforementioned rerouting of payments by intermediary hops is something that is explored in the Bailout rerouting protocol (Ersoy et al., 2023). But where this protocol unlocks and reroutes payments before the payment is completed but after the coins are locked, the rerouting in PSS is only possible before the actual locking of the coins. This makes PSS a less strong mitigation against jamming attacks, but a useful one nonetheless and one that is possible with LN today.

Spider (Sivaraman et al., n.d.) is a multi-path transport protocol that utilizes a packet-switched architecture. As we have mentioned previously, PSS turns the circuit-switched nature of LN into a packet-switched one, albeit less comprehensive than Spider does. Nonetheless, network throughput will improve if PSS is implemented, similar to other multi-path payment methods (Osuntokun, 2018; Piatkivskyi and Nowostawski, 2018).

6 Conclusion

In this paper we showed a new technique that allows intermediary parties in a multi-hop payment to reroute the payment to the next hop over an alternative route. We coined the term Payment Splitting and Switching (PSS) for this technique. We showed the viability of this technique by developing a `Core Lightning` plugin that works with Lightning Network today. We demonstrated the effect of PSS on the information gain that can be achieved through a BDA by an attacker. The simulations, using a real-world network snapshot, showed very pronounced results for Balance Disclosure Attacks (BDA) employing either direct or remote probing. For direct probing of single channel hops the information gain dropped by 50% and for remote probing it dropped by 62%. For multi-channels hops the information gain decrease becomes relatively smaller because the potential information gain without PSS is already smaller to begin with, but the difference remains distinct and significant. Secondly, PSS forces an increase in the time complexity of the BDA algorithm from $O(\log n)$ to $O(n^c)$. This makes the use of BDA at a scale needed for network-wide payment discovery unfeasible with off-the-shelf hardware. The Lightning Network shows tremendous promise to improve Bitcoin's scalability and privacy, but also introduces new types of attacks. With this work we proposed a practical and obtainable mitigation against on of those attacks, the BDA.

7 References

Androulaki E, Karame GO, Roeschlin M, Scherer T, Capkun S. Evaluating user privacy in bitcoin. International Conference on Financial Cryptography and Data Security, Springer; 2013, p. 34–51.
Barvinok AI. A Polynomial Time Algorithm for Counting Integral Points in Polyhedra When the Dimension is Fixed. Mathematics of Operations Research 1994;19:769–79. https://doi.org/10.1287/moor.19.4.769.
Béres F, Seres IA, Benczúr AA. A Cryptoeconomic Traffic Analysis of Bitcoin's Lightning Network. Cryptoeconomic Systems 2020:1–23. https://doi.org/10.21428/58320208.d4cd697e.
Biryukov A, Naumenko G, Tikhomirov S. Analysis and Probing of Parallel Channels in the Lightning Network. Financial Cryptography and Data Security, Springer Cham; 2022.
Chan T-HH, Shi E, Song D. Private and Continual Release of Statistics. ACM

Transactions on Information and System Security 2011;14:1–24. https://doi.org/10.1145/2043621.2043626.
Danezis G, Goldberg I. Sphinx: A compact and provably secure mix format. Proceedings - IEEE Symposium on Security and Privacy 2009:269–82. https://doi.org/10.1109/SP.2009.15.
De Loera JA, Hemmecke R, Tauzer J, Yoshida R. Effective lattice point counting in rational convex polytopes. Journal of Symbolic Computation 2004;38:1273–302. https://doi.org/10.1016/j.jsc.2003.04.003.
Di Stasi G, Avallone S, Canonico R, Ventre G. Routing Payments on the Lightning Network. Proceedings - IEEE 2018 International Congress on Cybermatics: 2018 IEEE Conferences on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, iThings/Gree 2018:1161–70. https://doi.org/10.1109/Cybermatics_2018.2018.00209.
EmelyanenkoK. Payment channel congestion via spam-attack · Issue #182 · lightning/bolts. GitHub 2017.
Ersoy O, Moreno-Sanchez P, Roos S. Get Me out of This Payment! Bailout: An HTLC Re-routing Protocol. Financial Cryptography and Data Security, 2023.
Gudgeon L, Moreno-Sanchez P, Roos S, McCorry P, Gervais A. SoK: Layer-Two Blockchain Protocols. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 2020;12059 LNCS:201–26. https://doi.org/10.1007/978-3-030-51280-4_12.
Herrera-Joancomartí J, Navarro-Arribas G, Ranchal-Pedrosa A, Pérez-Solà C, Garcia-Alfaro J. On the Difficulty of Hiding the Balance of Lightning Network Channels 2019:602–12. https://doi.org/10.1145/3321705.3329812.
Kappos G, Yousaf H, Piotrowska A, Kanjalkar S, Delgado-Segura S, Miller A, et al. An Empirical Analysis of Privacy in the Lightning Network 2020.
Malavolta G, Moreno-Sanchez P, Kate A, Maffei M, Ravi S. Concurrency and Privacy with Payment-Channel Networks. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17, New York, New York, USA: ACM Press; 2017, p. 455–71. https://doi.org/10.1145/3133956.3134096.
Malavolta G, Moreno-Sanchez P, Schneidewind C, Kate A, Maffei M. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. Proceedings 2019 Network and Distributed System Security Symposium, Reston, VA: Internet Society; 2019. https://doi.org/10.14722/ndss.2019.23330.
Meiklejohn S, Pomarole M, Jordan G, Levchenko K, McCoy D, Voelker GM, et al. A fistful of Bitcoins: Characterizing Payments Among Men with No Names. Communications of the ACM 2013;59:86–93. https://doi.org/10.1145/2896384.
Nakamoto S. Bitcoin: A peer-to-peer electronic cash system 2008.
Osuntokun O. [Lightning-dev] AMP: Atomic Multi-Path Payments over Lightning 2018.
Pérez-Solà C, Ranchal-Pedrosa A, Herrera-Joancomartí J, Navarro-Arribas G, Garcia-Alfaro J. LockDown: Balance Availability Attack Against Lightning Network Channels, 2020, p. 245–63. https://doi.org/10.1007/978-3-030-51280-4_14.
Piatkivskyi D, Nowostawski M. Split Payments in Payment Networks. In: Garcia-Alfaro J, Herrera-Joancomartí J, Livraga G, Rios R, editors. Data Privacy Management, Cryptocurrencies and Blockchain Technology, vol. 11025, Cham: Springer International Publishing; 2018, p. 67–75. https://doi.org/10.1007/978-

3-030-00305-0_5.
Poon J, Dryja T. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. vol. 18. 2016.
Rahimpour S, Khabbazian M. Torrent: Strong, Fast Balance Discovery in the Lightning Network. 2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Shanghai, China: IEEE; 2022, p. 1–7. https://doi.org/10.1109/ICBC54727.2022.9805520.
Rampfl S. Network Simulation and its Limitations 2013. https://doi.org/10.2313/NET-2013-08-1_08.
Reed MG, Syverson PF, Goldschlag DM. Anonymous connections and onion routing. IEEE Journal on Selected Areas in Communications 1998;16:482–93. https://doi.org/10.1109/49.668972.
Romiti M, Victor F, Moreno-Sanchez P, Nordholt PS, Haslhofer B, Maffei M. Cross-Layer Deanonymization Methods in the Lightning Protocol. arXiv 2020.
Sivaraman V, Venkatakrishnan SB, Ruan K, Negi P, Yang L, Mittal R, et al. High Throughput Cryptocurrency Routing in Payment Channel Networks n.d.
Sompolinsky Y, Zohar A. Secure High-Rate Transaction Processing in Bitcoin (full version). Financial Cryptography and Data Security - 19th International Conference, FC 2015 2015:507–27.
Tikhomirov S, Pickhardt R, Biryukov A, Nowostawski M. Probing Channel Balances in the Lightning Network 2020.
van Dam G, Abdul Kadir R. Hiding payments in lightning network with approximate differentially private payment channels. Computers & Security 2022;115. https://doi.org/10.1016/j.cose.2022.102623.
van Dam G, Kadir RA, Nohuddin PNE, Zaman HB. Improvements of the Balance Discovery Attack on Lightning Network Payment Channels. In: Hölb M, Rannenberg K, Tatjana W, editors. IFIP Advances in Information and Communication Technology, Springer International Publishing; 2020, p. 313–23. https://doi.org/10.1007/978-3-030-58201-2_21.
Wang P, Xu H, Jin X, Wang T. Flash. Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies, New York, NY, USA: ACM; 2019, p. 370–81. https://doi.org/10.1145/3359989.3365411.
Zhao Z, Ge C, Zhou L, Wang H. Fully Discover the Balance of Lightning Network Payment Channels. In: Liu Z, Wu F, Das SK, editors. Wireless Algorithms, Systems, and Applications, vol. 12937, Cham: Springer International Publishing; 2021, p. 159–73. https://doi.org/10.1007/978-3-030-85928-2_13.