applied sciences

MDPI

Article

Evaluating Methods for the Identification of Off-Chain Transactions in the Lightning Network

Mariusz Nowostawski *,† and Jardar Tøn †

Computer Science Department, NTNU – Norwegian University of Science and Technology, Teknologivegen 22, 2815 Gjøvik, Norway; jardar.ton@ntnu.no

* Correspondence: mariusz.nowostawski@ntnu.no

† These authors contributed equally to this work.

Received: 17 April 2019; Accepted: 14 June 2019; Published: 20 June 2019

Abstract: Payment channels and off-chain transactions are used to address blockchain scalability. Those mechanisms rely on the blockchain proper, as the resolution mechanism. They allow for high transaction throughput due to the pure peer-to-peer nature of the transaction exchange that happens directly between the peers, without the involvement of the blockchain transactions. The transactions are not mediated through the blockchain but happen off-chain. The transactions in such overlay networks are not included in the blockchain, they nevertheless leave some data traces in a public ledger. We have used the Bitcoin mainnet and testnet blockchains together with the Lightning network node to explore what can be inferred from the underlying blockchain in the context of Lightning transactions, channel setup, and channel teardown. The main purpose of this study is to identify what methods, transaction signatures, and data can be used to understand the non-visible publicly off-chain transactions. We have proposed heuristics for identifying the setup and teardown transactions, quantified and analyzed the effectiveness of our proposed methods. Using the data from the Bitcoin blockchain, as well as the data from the Lightning network to link related information we have found when parsing the blockchain, we generate network graph representations showing the relationships between the Lightning network channels identified on the blockchain. This study is significant from the personal data and privacy perspectives, as well as from forensics. We have established that at least 75% of all P2WSH transactions are Lightning transactions, and some of the channels can be deduced from the blockchain analysis. The synthesized results demonstrate that our methods are viable for identifying a subset of transactions and that only partial topology of the payment channels can be obtained from the data left in the blockchain.

Keywords: Blockchain; Bitcoin; Lightning; Lightning network; LN; off-chain transactions; forensics

1. Introduction

Bitcoin is a peer-to-peer electronic cash system. It relies on cryptography and a distributed ledger [1] to record all transactions done in the system to provide a consistent global state without a single trusted third party or central authority managing that state. The blockchain is maintained collectively by the network of peer-to-peer nodes. New transactions are added to the ledger by spending computational power; making the transactions practically irreversible as the blockchain grows and more computational power is used. Such a decentralized cash system, used as a payment solution, does not scale. The Bitcoin blockchain can handle at best around 7 transactions per second [2]. This looks slow compared to centralized payment systems, such as Visa, which can handle up to 47,000 transactions per second [3]. One of the ways to solve the scalability is to use an overlay network to process payments. Such an overlay network, or payment channels network, supports the direct exchange of payments between the peers, which effectively means ease of scalability. Such a network allows users to do transactions without the actual publishing of the transaction in the Bitcoin ledger itself. The transactions are not included in the blockchain, which is why they are referred to as off-chain transactions. This makes it possible for more transactions to be done without increasing the capacity of the Bitcoin system itself. It can be thought of as an overlay layer on top of the Bitcoin system since it requires the Bitcoin system for verification and conflict resolution. One of such overlay networks is the Lightning network. The Lightning network, as described by Poon and Dryja [2], is a payment channel network currently deployed on top of the Bitcoin network. A payment channel is a one-to-one channel allowing two participants to exchange funds between each other. The Bitcoin blockchain has been used in the past in research projects focusing on anonymity and privacy of users [4,5]. Because the data in the blockchain provides a complete record of how funds are moved between entities, it is relatively easy to analyze the data from the blockchain and determine user involvement in transactions and the flow of funds. To achieve this, we group and contextualize the transaction information to provide understanding for past events (e.g., for forensic purposes) as well as to understand the implications for user privacy.

The use of the second layer network allowing transactions to be done off-chain makes traditional blockchain analysis difficult as all the transactions within the Lightning are invisible to the blockchain proper. However, as the Lightning network is built on top of Bitcoin, there is a need for on-chain transactions to manage the payment channels. These on-chain transactions from the LN are recorded on the blockchain just like any other Bitcoin transactions and the data contained in them is validated the same as any other transactions; they, however, do not provide the explicit record of how funds have moved inside the LN, unlike the Bitcoin transactions do for the Bitcoin network. This means one can still analyze the ledger (data in the blockchain) to see how funds moved inside the Bitcoin network, but one cannot do this to the same degree for the Lightning network. Jordi Herrera-Joancomartí and Cristina Pérez-Solà [6] point out that the methods used for blockchain analysis in earlier research is no longer effective for payment networks. If systems such as LN are widely adopted and used, the majority of transactions would be done off-chain and the only on-chain transactions would be the ones required for the payment channels to operate. Giulio Malavolta et al. [7] identifies some possible privacy implications of the on-chain transactions from payment channels networks, so there might be possibilities of using blockchain data related to payment channels networks to reveal something about users or transactions.

Research questions. In this work, we are interested in the kind of data that leak from the off-chain network through to the blockchain. What can be deduced and inferred about the layer-two overlay network based on the transactions recorded in the ledger? A second aspect is to explore how unique the transactions related to the Lightning network are.

2. Background

2.1. Bitcoin Transactions

In the Bitcoin blockchain, public key cryptography is used to sign transactions and to transfer ownership of value; meaning a transaction is tied to, or owned by, the owner of the private part of a public-private key pair. To transfer control of a transaction and therefore its value to another entity, the owner locks the transaction using one of the three typical output scripts: Pay-To-PublicKey (P2PK), Pay-To-PublicKey-Hash (P2PKH) and Pay-To-Script-Hash (P2SH). In the P2PK, the public key of the recipient is used directly, and in the P2PKH, a hash of the key is used instead. Note, even with P2PKH the public key is only hidden from the ledger until the transaction is unlocked, i.e., the output is spent. The unlocking script will reveal the owner's public key. The cryptographic lock placed on the transaction by the previous owner, makes the recipient in control of the transaction. Everyone can verify the transfer, because the previous owner generated the signature using their private key when locking the transaction to a new public key, and therefore showing that they had control of the key pair controlling the transaction at the time of the transfer. The transaction as a data structure, represents one transfer of value. For each transfer a new transaction is created. This results in a series of transactions, each representing a change in control of the value, and these are related to each other as they contain the same value, but represent different transfers. This is the chain of signatures (transactions) we introduced at the start of this section. By following the chain of transactions and verifying the cryptographic signatures in each transaction, everyone can verify the current ownership of value locked in the UTXOs. Only the newest transaction in the chain can transfer the value to another key pair. Previous transactions in the chain are only historical states, allowing for the current state to be verified. This transaction chain can fork and merge, enabling the value to be split into different transactions, or merge the value of many into a single transaction. Therefore, the structure is not a linear chain, but a directed acyclic graph (DAG). No transactions can be spent twice. With the possibility of splitting the value in a transaction, it no longer makes sense for a single key pair to control the entire value of a transaction. Instead, we should define value ownership in terms of controlling transaction outputs, which together with transaction inputs are the two main parts of a transaction. In short: outputs is the link forward in the transaction chain, while the input is the link backwards. The output of a transaction is where the creator of the transaction locks the value to a public key, and with multiple outputs can give different keys control of different outputs, each with a portion of the value of the transaction. The input of transactions, is the output of previous transactions in the chain, indicating which value the transaction transfers. The transaction graph Figure 1 depicts a small part of a transaction graph with outputs and inputs connecting transactions and transferring control of value. The transaction graph contains all information about how value has been transferred. The leaf nodes in the graph represent the current owners of the value. This graph structure (DAG) means that the value of a transaction must be used in its entirety when transferring it, or it will be lost, as its transaction is no longer a leaf node. If the creator of a transaction does not wish to transfer the entire value to another entity, they can transfer it back to themselves. This is done by creating transaction output and locking it to their own public key. Similarly, if a user has no output with sufficiently large value needed for a transfer, they can use multiple outputs for inputs to the transaction. By doing so will reach the desired value of the transaction. An example is depicted in the right upper transaction in Figure 1.

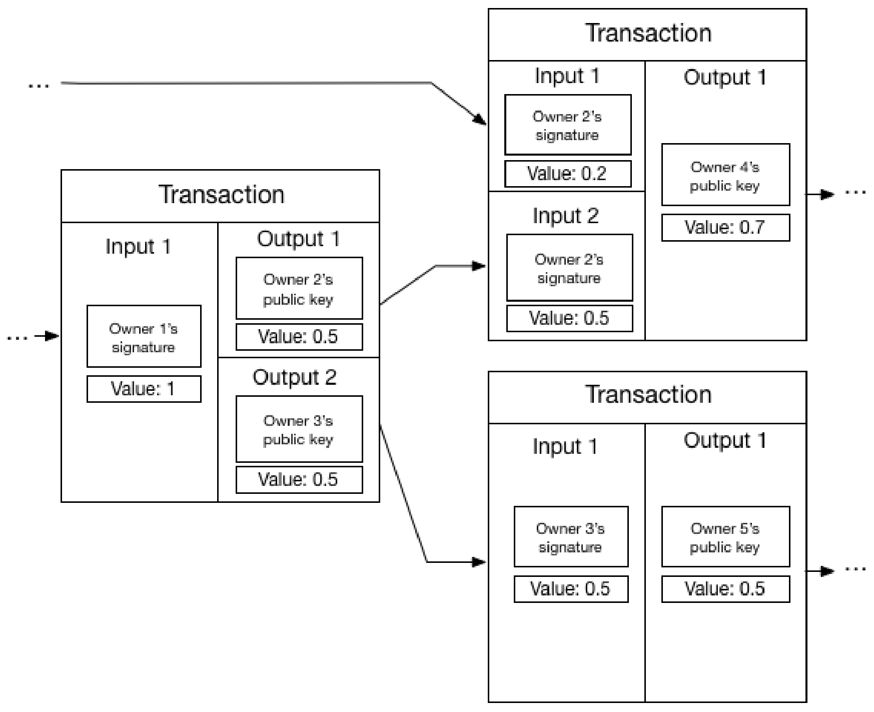


Figure 1. Section of a transaction graph.

2.2. Transaction Scripts

Bitcoin transactions use a stateless, stack-based, non-Turing complete language called simply `script` [8]. The outputs of transactions are locked using scripts that place requirements for spending the outputs; inputs contain scripts that meet the requirements placed by their respective output scripts. The language has a predefined set of operations that can be done, called `opcodes`, which limits the possibilities of scripts. An example of a common locking script placed in outputs are:

```
OP_DUP OP_HASH160 <receivers public key hash> OP_EQUALVERIFY OP_CHECKSIG
```

This script is an example of Pay to Public Key Hash (`P2PKH`) script. The sender can use this script to lock the output to the receiver public key hash. The receiver can spend this output by supplying their $< signature >< publickey >$ in the input, which will meet the requirements placed by the output script and unlock the funds. The unlocking script will be combined with the locking script and executed to ensure that it is valid.

```
<receivers signature> <receivers public key> OP_DUP OP_HASH160
        <receivers public key hash> OP_EQUALVERIFY OP_CHECKSIG
```

The goal of this script is to check if the `<public key>` in the unlocking part hashes to the `<public key hash>` of the locking part, and then if the `<signature>` in the unlocking part matches the public key. When the script executes, the execution pointer will move left to right pushing data to the stack and do the operations specified. In the case of `P2PKH` scripts, the `<signature>` will be moved to the stack followed by the `<public key>`, `OP_DUP` duplicates the top item on the stack, in this case the `<public key>`. Then the `OP_HASH160` will hash the top item on the stack, which is the top `<public key>`. The `<public key hash>` given in the locking part of the script will then be pushed to the stack and `OP_EQUALVERIFY` will compare the two top items, which now are the `<public key hash>` and the hash of the `<public key>` created by the `OP_HASH` operation previously, if they are equal they are both removed from the stack. The final operation `OP_CHECKSIG` takes the two last items on the stack: `<public key>` and `<signature>` and checks if the signature matches the public key, which means it was created with the corresponding private key.

There are two standard transaction types commonly used presently: one is the pay to public key hash (`P2PKH`) already described, and the other is pay to script hash (`P2SH`) [9]. While users are free to create any locking script they wish within the limits of the language, these types are recognized by most wallet software and allow for senders to transfer value by locking it in known ways. With `P2PKH` the receiver can just send a hash of their public key, and the sender can create the locking script as outlined above and just insert the hash. `P2SH` transactions allow for the same process but using the hash of a script instead of the hash of a public key. This allows for complex scripts to be created and hashed by the recipient. The hash value is then used for creating the locking script, resulting in:

```
HASH160 <script hash> EQUAL
```

The unlocking script will contain the script that was used to create the hash as one stack item only. When this is executed the entire original script, known as the redeem script, will be pushed to the stack, and then hashed using `OP_HASH`. Then the script hash from the unlocking script is pushed to the stack, and finally the two hashes are compared. The unlocking and locking script for this is shown below:

```
<redeem script> HASH160 <script hash> EQUAL
```

This hashing and comparing of hashes will ensure that the script supplied is the original one. After it is confirmed in the manner explained above, the redeem script itself is executed to check if it evaluates to true. The conditions for the redeem script must also be satisfied meaning that potential signatures or other data must also be supplied in the input unlocking an output. We can therefore say that that P2SH has two executions, one being to check if the redeem script given by the redeemer matches the hash in locking script as previously explained, and then the redeem script itself is executed. One reason for this is to move the responsibility for supplying these script from sender to receiver. The receiver can create whatever script they wish without the need for the sender to recreate the script when forming the transaction and locking the output. The sender receives a hash of a script and can create the standard P2SH locking script with the format given above. The receiver needs to know the script to unlock the output, but the sender does not need to know the script to transfer value; it therefore makes sense to place the burden of supplying the redeem script on the receiver instead of the sender. A typical use case for P2SH, is multisig (multi-signatures) redeem scripts:

```
2 <public key 1> <public key 2> 2 OP_CHECKMULTISIG
```

These scripts require multiple signatures and therefore multiple key pairs to be valid, which implies multiple users are needed to spend an output. In the case above two signatures are required for the two keys, indicated by the number 2 before the keys and the 2 after. This will be the redeem script which is hashed and given to the sender for them to create the P2SH locking script with this hash. The sender does not need to know the public keys required to spend the output or that it is a multisig script in the first place. The receiver, however, needs to know as they are the ones who will claim the value and will therefore likely control at least one of the public keys required in the redeem script.

2.3. Segregated Witness

Segregated Witness (segwit) is a change introduced to the Bitcoin consensus layer defined in Bitcoin Improvement Proposals (BIPs), BIP141 and BIP143 [10,11]. Segwit changes the structure of the validating data used to verify transactions and where this data is stored. Transactions outputs have a locking script and an unlocking script; the witness is the unlocking script, which enables us to check the validity of the output spent by using it with the locking script. Segwit places the witness or unlocking script in a witness data structure which is no longer stored inside the transaction. One of the main reasons this is done is to fix transaction malleability, which is the ability to change the transaction id of a transaction. This id is a hash of all the properties of a transaction including the unlocking script/witness. The only malleable part of a transaction is the unlocking script because a transaction is unique, and outputs can only be spent once—i.e., if the choice of inputs, choice of outputs, or the value changes, it is no longer the same transaction. Therefore, when the transaction has been created most of it is non-malleable. However, the unlocking script can be modified in different ways without invalidating the transaction core elements. Adding operations to the unlocking scripts or zero padding the numbers used can achieve this effect, and thus change the id, while the transaction is for all intents and purposes the same [12]. With segwit, the transaction hash will be calculated without the witness data, and therefore this removes possibilities for malleability. As the witness data is only used to validate the transaction itself, it is not necessary to determine the state of the blockchain and can, therefore, become optional for lightweight nodes wishing to save bandwidth.

A segwit transaction output is constructed such that a non-segwit enabled node will interpret the output as being spendable by everyone—i.e., redeemable by an empty signature; this means a non-segwit node cannot correctly validate a segwit transaction but will not consider it invalid [8].

Segwit has two standard transaction types that mirror the functionality of the existing ones: pay to witness public key hash (P2WPKH), and pay to witness script hash (P2WSH). The main change is how the locking scripts in the outputs are structured; instead of including the script in the output, there is only a version byte with the value 0 to 16 followed by 20 bytes containing a public key hash, or 32 bytes containing a script hash. The amount of bytes fund after the version byte allows us to determine if the output is a P2WPKH or P2WSH. Currently, only version 0 is used, with the rest reserved for future versions [10]. An example of a P2WPKH locking script is:

```
0 <20 byte public key hash>
```

The unlocking script is found in the witness structure which is a stack containing the required data to validate the transaction. In the case of the `P2WPKH` the witness stack will contain two elements: the `<signature>` and the `<public key>` which is required to validate the transaction. The mechanism is exactly the same as in the old `P2PKH` transaction type. In the case of the `P2WSH` type, the witness stack can contain a varying number of items depending on the requirements of the redeem script. If the redeem script is a multisig script the stack will contain the required amount of signatures, with the last item of the stack always being the redeem script itself.

2.4. Payment Channels in the Lightning Network

A payment channel allows for two people to connect and transfer funds between themselves. When many users create channels with each other, a network of channels is formed. Here we focus on how a channel allows for two people to transact in a trustless manner. The payment channels use Bitcoin transactions for both managing the channels and to send funds inside the channel. However, not all the transactions are broadcast to the Bitcoin network. All non-broadcasted transactions are therefore not included in the blockchain. At a high level, the channel is used by two participants to keep track of how funds are distributed between the stakeholders. This means that the channel has a balance showing how much funds each of the two participants has. The balance is updated as funds are sent between them. This is the basic premise of payment channels: when two people send funds between themselves, it should not be necessary to broadcast each transfer, instead only the final result, e.g., Alice and Bob have a channel with a value of one coin, with the initial distribution being 0.5 to each; Alice sends Bob 0.1 coins in three separate transactions, resulting in a final distribution of 0.2 to Alice and 0.8 to Bob; this result is broadcast while the three 0.1 transactions are not. Using the example above we can see that only publishing the result requires one transaction to be broadcast, in comparison to three transactions if all intermediary transactions were broadcast. The transactions that are broadcast are called `on-chain transactions`, since they will be included in the blockchain. `Off-chain transactions` are the intermediary transactions not broadcast, and therefore not found on the blockchain. While payment channels allow for multiple transactions to be done off-chain, they do require some on-chain transactions for managing the channels. One of these on-chain transactions is the funding transaction, which is used to create the channel and make a common starting point for the users in the channel. The funding transaction will contain an output locked using a multisig script, which we discussed earlier in Section 2.2. The multisig redeem script used for the output creating the channel has the form [13]:

```
2 <public key 1> <public key 2> 2 OP_CHECKMULTISIG
```

The multisig script requires multiple signatures from different keys to be spent. The script above is a 2of2 multisig script when we denote it in the form of *n* of *m*, where the number *n* is the required and *m* is the potential number of keys. Using 2of2 multisig scripts for the payment channel output in the funding transaction means that both parties in the channel must sign the transaction to spend it. This collectively controlled output in the funding transaction located on the blockchain, is the starting point of their channel, enabling them to do off-chain transactions. However, to avoid the value of the channel being stuck in the funding transaction because one or both parties are uncooperative, as spending the output on-chain requires both signatures, a new transaction is created with both signatures, spending the output and refunding the value. This is done before the funding transaction is published, such that either party can close out the channel by publishing the refund transaction. The process of setting up a channel starts with the parties creating a funding transaction with a 2of2 multisig output, but they will not exchange signatures for this transaction before they have created the refund transaction. The exchange of signatures is done in a specific order where first the refund transaction is signed by both sides, and only then are signatures exchanged for the funding transaction, and it is then published to the blockchain. The channel has now been established in a trustless manner, and it also allows for refunds initiated by either party by publishing the refund transaction and closing the channel.

The off-chain transactions done between the parties inside the channel are known as commitment transactions. They contain the current distribution (balance) in the channel, which is how the total value of the channel will be split between the parties. Each time a transfer is done inside the channel a new commitment transaction is created reflecting the new balance between the two parties, e.g., the total value of the channel is 10 and the balance is 5 to Alice and 5 to Bob, which is reflected in the most recent commitment transaction; Alice decides to send Bob 1 coin; now the balance is 4 coins to Alice and 6 to Bob and a new commitment transaction is created to reflect this change. The refund transaction can be said to be the first commitment transaction since it describes the initial balance in the channel. This initial balance will be whatever each party contributed to the input of the funding transaction. When the balance in the channel needs to be updated a new commitment transaction is created, which spends the output of the funding transaction again. While it is not possible to spend output twice, the commitment transaction has not been published, so the output is not really spent. Off-chain transaction consists of creating new commitment transactions for each transfer, reflecting the new balance in the channel.

Alice and Bob exchange signatures for each newly created commitment transaction. This is to satisfy the multi-signature condition on the funding transaction output, allowing any one of them to publish the commitment transaction to the blockchain at any time if they wish. As a transaction output can only be spent once, so any commitment transaction published to the blockchain will spend the funding transaction, making any other commitment transaction invalid (unspendable). This will also close the channel, meaning it can no longer be used—its funding transaction is spent. Channels are closed by publishing a transaction spending the output of the funding transaction. Therefore, to be able to do many transfers between the parties in the channel the commitment transactions should ideally not be published immediately. Only when something goes wrong, or both/one of the parties wishes the channel to close, should the newest commitment transaction representing the current balance be published to the blockchain, and thereby closing out the channel and distributing the value. This results in only two on-chain transactions: the funding and closing transaction, with a potential of many off-chain commitment transactions done in-between through the channel—those are invisible in the blockchain.

The payment channels should be trustless and operate in a potentially hostile environment. There are mechanisms that ensure cooperation—e.g., the refund transaction and funding transaction signing order, which makes locking the funds used in the channel impossible. With the newest commitment transactions reflecting the current balance in the channel, older ones will contain a different and therefore wrong balance, and should not be published. Using the earlier example where a channel between Alice and Bob had the value of 10 coins and the balance was 5 to each of them. If Alice does several payments to Bob and the balance ends up being 0 to Alice 10 to Bob, she could publish the commitment transaction where the balance was 5 to each and get all her money back. To deal with this problem, there is a penalty to anyone who publishes any other commitment transaction than the latest. The penalty is that whoever published the older commitment transaction will lose all the funds to the other person on the channel. As both can publish an old commitment transaction, we must determine which of the two published; this is described in the Lightning paper as the problem of ascribing blame [2]. When the two parties exchange signatures for a commitment transaction, we end up getting two different versions of the same transaction. Bob signs one and sends it to Alice, and Alice signs one and sends it to Bob. This means both sides end up with a half-signed transaction which only needs their own signature before it is valid and can be published to the blockchain. The multi-signature requirement means that each of them can only publish the one they received from the other party (the one already signed by the other participant). If Bob is the one who published a commitment transaction we can find out because he must publish one of those he received from Alice containing her signature. As each side receives their version of the commitment transactions from the opposing party, the opposing party can create an insurance clause on the output to the party receiving the commitment. This means that when Bob signs a commitment transaction for sending to Alice, he can place an insurance clause on the output for Alice, punishing her if the commitment is published when it is outdated. This means that the commitment transactions for the two parties are not exactly the same, because they will contain this insurance mechanism, but the transaction pair will still have the same balance, spending the same outputs and have the same outputs.

The need to wait for confirmations to spend the output is enforced on every set of created commitment transactions. It provides time to enforce a punishment, that is, it allows the other party to spend the output before the timelock expires. However, the punishment only applies to old commitment transactions, so the punishment is only made possible when a commitment transaction pair is invalidated by creating a new one. The invalidation or revocation of old commitment transactions is done by creating a Breach Remedy transaction which spends the same output as the timelocked/RSMC output. Each of the parties sign this transaction spending the timelocked/RSMC output of their own commitment transaction, and then gives it to the other party to revoke it. This enables both of the parties to spend the timelocked output belonging to the other party if they should publish a revoked commitment transaction—e.g., Alice creates a Breach Remedy transaction spending her output in the old commitment transaction and sends it to Bob, and he does the same; if either of the two parties now publishes an old commitment transaction, they cannot spend their output immediately because of the RSMC/timelock, but the other party can publish the Breach Remedy transaction, and spend their own output and thus get all funds in the channel. However, spending the timelocked output should be done before the timelock expires as the party which published the old commitment can now also spend that output. Both parties should, therefore, observe the blockchain so they can see if an old commitment transaction is published, allowing them to use the Breach Remedy transaction before the timelock expires.

The channel can also be cooperatively closed if both parties agree to do so. The parties must simply create a new transaction spending the funding transaction with the outputs reflecting the balance in the latest commitment. They exchange signatures and it can be published on the blockchain. The outputs of this transaction will not have any time-locks, so each party will be able to claim their funds when they wish. This is because they both must agree to close a channel this way, if they cannot agree, one party can simply publish their newest commitment transaction and close the channel that way, but they will need to wait for the timelock to release the funds. Closing a channel cooperatively means that the total number of transactions published to the blockchain will be exactly 2: one funding and one closing transaction. This is the simplest and the most common case. Otherwise, the extra timelocked (or potential Breach Remedy) transactions will be recorded in the blockchain.

2.5. Networked Payment Channels

A single payment channel is used by two entities to exchange funds with each other. The Lightning network is formed by having many such interconnected channels between different nodes. Thanks to such a network, users can pass funds to those that they do not have a direct channel with. For this to work, there are additional mechanisms that allow for payments to be routed across other users' payment channels. As explained in the previous section, the payment channels used in the LN do not require trust between the two endpoints of the channel. There are also mechanisms that make sending payments across channels not requiring the sender to trust the intermediary nodes along the path. The construction that enables this is called Hashed Timelock Contracts (HTLC). It creates contracts where intermediary nodes have a guarantee that they can get funds from the sender if they first transfer it to the receiver. By having the intermediary nodes first pay the receiver and then get the funds from the sender, one stops the intermediary nodes from not passing the funds along to the receiver. If the intermediary was sent the funds first, they could simply not pay the receiver and keep the funds to themselves. HTLC uses a hash function with an input/preimage `R`; the receiver inputs `R` into the function and gets a hash `H` which is given to the sender. The sender then promises to pay the intermediary if they can provide `R` that generates `H`. The intermediary then makes the same promise to pay the receiver if they can provide a `R` that generates `H`. Now a chain of promises has been created, each promise guarantees to pay an amount to the receiver that can reveal the input `R` which generates `H`. Because it was the receiver who created `H` using `R`, they know `R` and can, therefore, give it to the intermediary and get the funds from them. `R` has now been disclosed to the intermediary, which in turn can use it to get the funds from the sender as previously promised. This can easily be extended to include more intermediaries.

The promise to pay someone in exchange for disclosing `R` generating `H` is not simply a promise. It is a part of a transaction which needs `R` to be valid. When the sender promises to pay the intermediary, the sender is no longer in control of the funds because a transaction containing this clause has been created and given to the intermediary. Thus, the funds are in a way transferred in a manner and direction one would expect (from sender to intermediary, then intermediary to receiver), but with the clause of providing `R` generating `H` to be claimed. A timelock is used to ensure that if no R is provided, the funds can be returned to the creator of the transaction, i.e., the one who made the promise to pay if provided the correct `R`.

3. State of the Art in Off-Chain Transactions Analysis

The existing research regarding blockchain transaction analysis does not consider the off-chain transactions and Lightning network since most of the research was done before the inception of the LN. Some of the approaches and methods can nevertheless be relevant for off-chain tx analysis. The fact that the blockchain is a public distributed data structure containing all transactions done by every Bitcoin user means that it is theoretically plausible to track all users and all transactions. Analyzing the blockchain should provide us with all records of all transactions related to a given single individual; this will include the amount sent and received, the time and date this was done, and which addresses these transactions were going to or coming from. The addresses are encoded public key hashes, and keys can be generated at will, so users are free to use new addresses whenever they want. For improved privacy, it is considered best practice to generate a new address for each transaction. Doing that results in transactions between different addresses each time. We will refer interchangeably to keys and addresses since an address is just a hashed and base48 encoded public key. The practice of not reusing keys means there is no longer a 1-to-1 mapping between users and keys. This use of new keys for each transaction gives the users Pseudonymity, which is defined by Pfitzmann and Köhntop [14] as: "Pseudonymity is the use of pseudonyms as IDs.". They also define pseudonymity in relation to linkability, which depending on the context has different levels of anonymity. The relevant aspect for us is the `transaction pseudonym`. Each transaction will have a different pseudonym that is used and this makes different transaction pseudonyms hard to link. The transaction pseudonyms are exactly the mechanism that the Bitcoin system uses to provide privacy for its users. Having a new pseudonym for each transaction makes it difficult for observers to link the activity of users. Linkability, therefore, is defined as the ability to find a relationship between two items in a system. With respect to key reuse, a key is obviously related to itself, but there is also be the relationship between the transactions the keys are used in, allowing us to also link the transactions based on this. According to Pfitzmann and Köhntop unlinkability is one of three main concepts of anonymity and privacy. The other is anonymity, which is defined as such: "Anonymity is the state of being not identifiable within a set of subjects, the anonymity set." [14], and unobservability which means that no message is any different from random noise, i.e., encryption. The latter is not applicable to the Bitcoin system as it is an open system with a public record of messages (TXs) which is observable to the same degree by every participant. The system is also not anonymous as the subjects are distinguished by their keys.

Pseudonymity and unlinkability are the two privacy concepts used to avoid full transparency of activity within the system. Androulaki [15] suggests two privacy notions adapted to Bitcoin: activity unlinkability and user profile indistinguishability. The former refers to the inability of an observer to link keys or transactions belonging to a single user of their choosing, and therefore revealing all the user's activities. This is a more specified unlinkability notion describing the lack of relationship between keys and transactions for a single user. User profile indistinguishability provides stronger privacy guarantees. A user profile can consist of keys or transactions which are linked, meaning they are discovered as belonging to the same user. User profile indistinguishability means that an observer should not be able to link keys or transactions to the right user, and thus to construct correct user profiles. The linking of keys allows us to say that a set of keys belong to the same user, while the linking of transactions means that a user has participated in that set of transactions. As noted by Androulaki et al. [15] key linking entails transaction linking because keys are contained within transactions; still there is room for transaction linking because other information might make us able to link transactions without being able to link keys.

The research focused on linking information in the blockchain demonstrated that it is possible to use heuristics and link pseudonyms to a single user. There is growing literature on how to create user profiles [4,5,15,16]. To create profiles, we need to link related transactions and link related keys. A simple way of linking transactions is to use the fact that they are naturally related through the output-input structure, creating a DAG as shown in Figure 1 and in the background of Figure 2. This graph depicts the relationship between the transactions (nodes). However, this network will only show how transactions are related and not much about the users themselves, due to the use of pseudonymous keys. This problem is illustrated in Figure 2 where we have overlaid the keys and the flow between them. We have in total seven keys in this transaction sub-graph, and with one key being used twice, the unique number of keys is six. Without linking keys, the number of possible users will be the same as unique keys, as we cannot easily determine if two keys belong to the same user or do they represent different users. Linking keys, or grouping them into clusters, create a set of keys controlled (potentially) by the same user or entity, which means we have reduced the set of possible participants in the transaction graph. If we could link all keys related to a user, we would be able to find all activity that the user participated in. Using the set of keys related to each user, we can create a user network similar to the transaction network, but where the nodes is users in addition to transactions, and the edges is the output-input pair from the transactions—i.e., showing the flow of Bitcoin from one user to another as illustrated by Figure 3. This user network is shown in Figure 3 allows us to discover how funds are moved between users. This is the ultimate goal of de-anonymization techniques. Reid and Harrigan [4] created such networks in their paper analyzing Bitcoin privacy. In a paper reviewing Bitcoin privacy papers, Herrera-Joancomartí [6] states that this user network allowed Reid and Harrigan do user-centered analysis, giving context to the data in the blockchain. Reid and Harrigan also created an ancillary network when linking keys (nodes being a unique key and edges being a relationship between two keys). This network would contain the key sets fund for each user, each set represented by a maximally connected component in the network. A different type of user network was created by Meiklejohn et al. [5]: not structured as the transaction graph as the one in Figure 3, but instead showing users outside the context of the transaction graph. Each node is a user (linked key set) and the edges were any transaction between the two, showing us if a pair of users had done any transactions at any point in time. These networks illustrate the main goal of linking information: user profiles. A set of linked keys representing a user allow us to see the activity of that user.

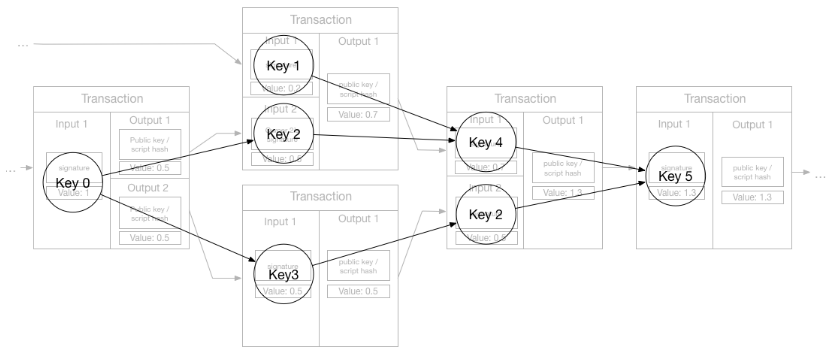


Figure 2. Transaction network with keys shown.

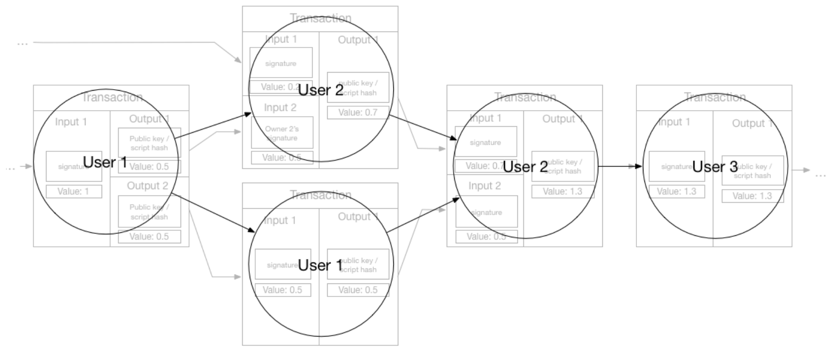


Figure 3. User network after linking addresses.

To link keys and create user profiles, there have been proposals for the use of heuristics. There are two well-known heuristics for linking keys. The first one has been used by several projects [4,5,15,16] and was also pointed out by Satoshi Nakamoto in the Bitcoin original whitepaper [1]. Keys used in transactions with multiple inputs can be said to belong to the same entity. As we explained in Section 2.2: spending outputs requires signatures in the input, from the private key paired with the public key used to lock the output. A transaction with multiple inputs means that all keys must have been available to sign the inputs, and so it is very likely that the creator of the transaction controls all keys used. Note that *very likely* does not mean certainty. If we consider Figure 2 we have two transactions with multiple inputs, and we can therefore link keys 1 and 2, and keys 2 and 4. As noted by Meiklejohn et al. [5] this effect is transitive, which means that by linking the keys from Figure 2 as done above, we can also link key 1 and 4, because both are related to key 2 which is reused in the two transactions, so they must also be related to each other. By doing this we have a set of three unique keys controlled by one user; we can now use this set to define a user (user 2) in the user network as shown in Figure 3. The second heuristic uses `shadow addressees`, first explored in [15] and then later refined in [5]. Because an output must be spent in its entirety, a user wishing to transfer a value less than the value of available outputs must send the remainder back to themselves. A real-world example of this is receiving change when paying with a cash bill with a higher value than the price of the item bought. Most Bitcoin software implementations known as wallets, allowing users to do transactions, create outputs to transfer change back to sender automatically with a new key each time. Transactions doing this have multiple outputs: normal ones sending the value to another entity, and an output giving back the change to the sender. The creation of this shadow address (key) and the locking of the output to this key is handled internally by the software, and users would not necessarily know the address, which makes it unlikely it is given to others for receiving transactions. This heuristic relies on the fact that these keys will only be used to receive the change in one transaction, and not in any other transactions. It also relies on people not using a single transaction to send the value to multiple people, as the linking consists of finding multiple output transactions and assuming the shadow address is a key not used before in the transaction graph. Meiklejohn et al. [5] also note that this relies on the implementation of wallet software, and is not an inherent property of the Bitcoin system itself. Applying this heuristic to the transaction graph with keys overlaid in Figure 2 we will find one multi-output transaction with funds going from key 0 to key 2 and 3; key 3 has not been used before or after in the transaction graph while key 2 has, so it is very likely that the output to key 3 is change returned to the same user controlling key 0. Figure 3 shows the final result of the user graph, where we now have reduced the set of possible users from six to three by applying the two heuristics for linking keys.

The two heuristics discussed above for linking keys use information or characteristics from the Bitcoin system itself. However, one can also link keys by using off-network information—i.e., sources outside the Bitcoin system. Reid and Herrigan [4] used the approach: exchanges allowing users to sell, buy, withdraw, and deposit Bitcoin will have access to the keys users use for depositing and withdrawing; which allows the exchanges to link those keys based on the user profile on that exchange. However, this information is not publicly available, but the method can be used for other types of services or data sources. People publicly disclosing multiple keys so they can receive Bitcoin is one such way. Reid and Herrigan [4] provide another example that is publicly available. A Bitcoin faucet is a webpage where people can donate Bitcoins and other users can receive a small amount of. The log of the redistribution containing IP-address of receivers is published to prevent abuse. The external information used for linking outlined above exists publicly on the Internet, but one can also link keys by actively gathering this data. This was done in the study by Meiklejohn et al. [5], where they participated in mining pools, withdrew and deposited funds on exchanges or wallet services, used Bitcoin gambling sites, and purchased goods from vendors. By doing this they were able to collect keys related to the different services and entities they did transactions with. This shows that engaging with other entities or users multiple times will allow us to link the keys used in those transactions. An important remark about this, done both by Meiklejohn et al. [5], and Reid and Herrigan [4], is that a big entity in the network which does transactions with many other separable users will have greater opportunity to identify or track users, because they can use their central position to link keys. Such an entity could be a big exchange where people trade other currencies for Bitcoin, for example, or a large online wallet provider.

Herrera-Joancomartí et al. [17], discuss privacy in relation to Bitcoin scaling solutions and point to several problems with blockchain analysis. One central issue is the increasing size of the blockchain as it constantly grows. Herrera-Joancomartí et al. state that at the time of writing that paper, the blockchain was 72 GB. Herrera-Joancomartí et al. also point out that the linking heuristics used in previous studies are no longer applicable for off-chain transactions facilitated by networks such as the LN. This is simply because the transactions will not be recorded on the blockchain. Herrera-Joancomartí et al. suggests that using payment channel networks will increase the privacy of users with regards to blockchain analysis, but that the existing methods may be adapted to analyze the payment channels in a network such as the LN. It has been suggested that identifying funding transactions on the blockchain can imply the network topology [18] of the LN. This is because all channels in the LN must be anchored in the blockchain with a funding transaction signed by both parties in the channel. If one were able to identify funding transactions on the blockchain, and had some way of linking channels (represented on the blockchain as funding transactions) to each other, it would reveal how the LN is structured as the linked channels would form the same network as the LN. This possibility is also mentioned in the LN paper itself [2] were they state that by monitoring the blockchain for funding transactions, a routing map of the LN can be theoretically built. Similarly, by linking the funding transaction and closing transaction of a channel, we get closed channels, so linking it with other channels would give us the structure of the LN in a previous state.

Herrera-Joancomartí et al. [17] suggest that the typology of the LN will affect privacy. They propose that a possible typology for the LN is one where there are some highly connected core nodes that have big capacity channels with each other, while the end users connect to one of these when they want to do transactions. In contrast to the Bitcoin network which is a decentralized p2p network, the LN with this topology will exhibit properties of a centralized system. The number of core nodes and how many connections they have will of course play a big role in the degree of centralization. As a countermeasure, onion routing has been proposed and is being implemented for LN. With this, in the payment path, a node will only know where it received the payment from and where the funds were forwarded. It cannot know the actual source and destination.

The recent paper by Malavolta et al. identifies some privacy notions specifically for payment channel networks [7]. The two notions are value privacy and relationship anonymity. Value privacy is that users outside the payment path cannot learn the value of the payment. A payment path is the route the payment travels inside the network. The notion is specified to only be off-path meaning it only considers nodes that are outside the path, so the nodes in the path will learn the value. Relationship anonymity is a more general anonymity concept which relates to likability. Pfitzmann and Köhntop [14] also defines this concept and explain that it relates so sender and recipient anonymity but is a weaker anonymity concept. Relationship anonymity means that the relationship between sender and receiver is unlinkable; who sends a message can be discovered and similarly who receives a message, but their relationship cannot. Malavolta et al. [7] suggests that this concept is a relevant property for on-path payments in payment channel networks. This means that the nodes that are in the payment path in the network will not be able to link the sender and receiver of a payment.

4. Methodology and Implementation

For the LN transaction analysis based on the transactions recorded in the blockchain, we can only see transactions: to open, to close, or to claim founds related to a channel. In this study, we will not use the information in non-LN-related transactions, such as address clustering, as those techniques are covered by other research. Instead, we limit our analysis only to LN.

The information found in the LN can be recorded and stored by participants, but the system itself does not keep such data records. The data in the blockchain can be verified by any full node while data recorded from the LN could not be verified in the same manner. There is simply no record of data related to LN. This means that if we want verifiable historical data about the LN or to verify data collected from the LN for study validation, we must collect it ourselves. The blockchain-based data will have limited and less informative content than what can be found in the LN directly. However, the advantage of blockchain-based data is its verifiability. It has the properties of being verified and automatically recorded by the Bitcoin system.

Our study has two main components. The first is to obtain full reference data by observing activities within LN. This provides us with the *ground truth*. The second is to analyze blockchain and see how much can be inferred from the data stored there. We start with the blockchain-only analysis.

4.1. Blockchain Analysis

The transactions in the blockchain are linked with outputs-inputs forming a DAG. Parsing the blockchain involves linking these transactions to form the full transaction graph. When applying the heuristics discussed in Section 2, one can use the results of this to provide additional context to the graph—i.e., when users are defined by key sets, the pseudonymous properties of using new keys for each transaction is removed, and so user activity is revealed. Because we are interested in transactions related to the LN only, we do not create a complete transaction graph containing all transactions on the blockchain; instead, we only link transactions related to a single LN channel, which creates many small transaction graphs, each representing a LN channel, and each representing a sub-graph of the complete transaction graph.

We will refer to these graphs as channel graphs because they contain the most common on-chain transactions related to a LN channel. By only creating channel graphs and not the complete transaction graph, we do not need to keep track as much data during parsing of the blockchain. We use two methods for locating possible channel graphs: using timelocked redeem script, and using 2of2 multisig redeem scripts. They are different. The multisig method gets all potential channels; however, it also gives false positives. The timelocked method is more precise in terms of identification, but will not be able to identify as many channels.

4.1.1. Using 2of2 Multisig for Identification

Channel graphs created with the use of our method contain the funding and closing transactions as shown in Figure 4. This is important for all types of channel graphs, as these transactions (funding and closing) are the key data points that can be extracted from the blockchain proper, and relate to LN channels.

To create the channel graphs when parsing the blockchain, we must be able to differentiate between on-chain LN transactions and other Bitcoin transactions that are not LN-related. Therefore, we must find some characteristics needed or only found in on-chain LN transactions. In Figure 4 we can see some characteristics of the funding and closing transactions. The output in the funding transaction used for the channel must be `P2WSH`, and the closing transaction will always have one input which will be the 2of2 multisig redeem script as explained in Section 2.4. The different characteristics have different levels of uniqueness, but using more unique characteristics will often impact the number of channels that can be identified. There is a trade-off between precision and sensitivity. An example of this is the `P2WSH` output type, which is not unique, so it provides false positives, but it improves sensitivity (all LN funding transaction use P2WSH). It allows the detection of both closed and open channels. The 2of2 redeem script is more unique, as it is a specific redeem script and must therefore also belong to a `P2WSH` output-input pair. In the funding transaction we will only be able to check if it has any `P2WSH` outputs, while the closing transaction allows us to check for the 2of2 multisig characteristic, which the presence of also entails the `P2WSH`. Using closing transactions limits us to closed channels, but the transactions matching our characteristics are more likely to be LN-related. While the 2of2 multisig redeem script is present in all closing transactions for LN channels, they can also be used for other purposes. We, therefore, cannot use this to determine with full certainty if a transaction is a closing transaction or not, but it will rule out all transactions without this characteristic. By locating all transactions that have inputs with such redeem scripts, we have all potential closing transactions, and by following the input-output connection we can also locate the funding transaction, providing us with a channel graph with the transactions as shown in Figure 4.

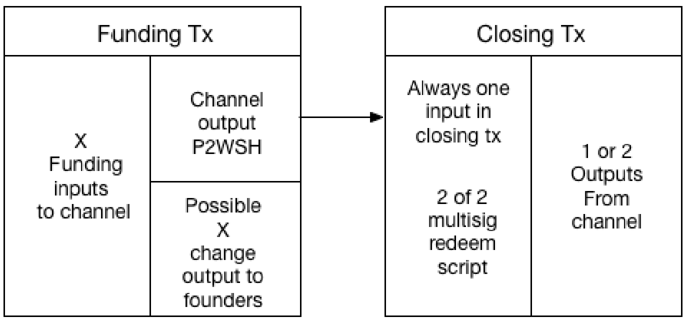

Funding Tx
X Funding inputs to channel
Channel output P2WSH
Possible X change output to founders
Closing Tx
Always one input in closing tx
2 of 2 multisig redeem script
1 or 2 Outputs From channel

Figure 4. Channel graph with funding and closing transactions.

Creating the channel graphs as explained above, enables us to identify the transactions in the channel graph in the reverse-order in relation to their creation, i.e., we identify the closing transaction first, and use it to find the funding transaction. All the transactions form a DAG, meaning they are located chronologically in the blockchain, with the newer transaction being found towards the tip of the chain. For the transactions in our channel graphs, this means that the closing transactions will be found before the funding transaction, as the closing uses the output of the funding. We must, therefore, parse the blockchain in reverse, beginning with the most recent block, and working towards the genesis (first) block.

Our software parses the blockchain and identifies relevant transactions, using both the multisig and timelocked identification methods. We use the `btcd` [19] Bitcoin implementation which is written in `Golang`, to synchronize and store the blockchain data. Our software, which is also written in Go, uses libraries from `btcd` to read this data from disk and make it available to us as a convenient data structure. At a high level our software finds the latest block from the blockchain stored on disk and uses it as a starting point; it then iterates over the transactions in the block, and checks if they are relevant, i.e., they are part of a channel graph. If that is the case the transactions are stored in a data structure representing a channel graph. After all transactions in a block are parsed, the hash of the preceding block is used to fetch it from the disk, and the same process is repeated. For each transaction parsed, the input part is checked for the presence of a 2of2 redeem script, if that is the case, a channel graph is created, and the transaction is stored as a closing transaction. As explained in Section 2.3 each transaction has an id, which is the transaction hash. Transaction inputs reference the hash of the transaction containing the output they spend, so when finding a potential closing transaction, we get the hash of the corresponding funding transaction. This is stored in a list, which we can use to recognize funding transactions when we encounter them in the blockchain. We can always check the hash of each transaction we parse, to see if it is the funding transaction for a closing transaction we have already found. If there is a match, we have found all transactions for a single channel graph. This is the main algorithm for the software as can be seen in Figure 5, where the process described above is illustrated.

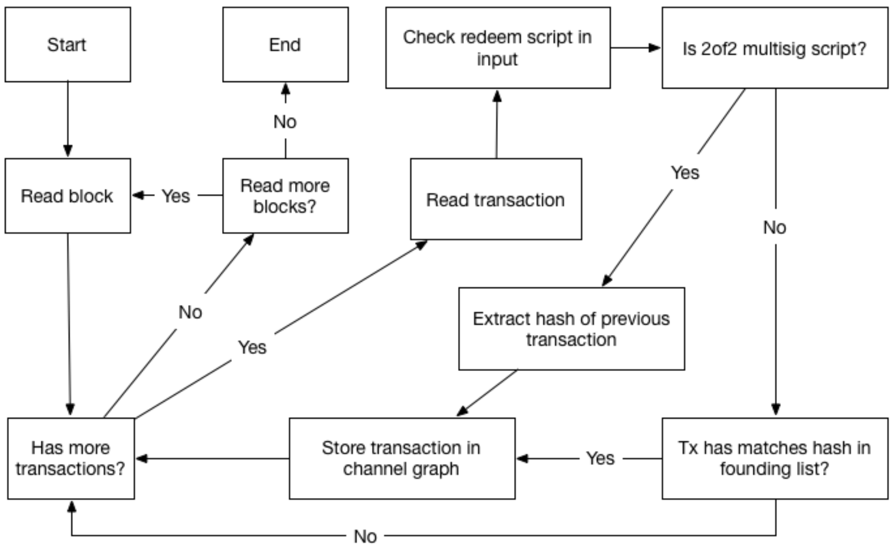

Start
End
Check redeem script in input
Is 2of2 multisig script?
No
Read block
Yes
Read more blocks?
Read transaction
Yes
No
No
Yes
Extract hash of previous transaction
Has more transactions?
Store transaction in channel graph
Yes
Tx has matches hash in founding list?
No

Figure 5. Algorithm for parsing blockchain and creating channel graphs using the multisig method.

Below we provide the 2of2 multisig redeem script used to identify possible closing channels. As an example, we have also included the byte version of the script. This is exactly how we get the script when extracting it from the input of a transaction. We use the byte representation when identifying the script.

```
2 <public key 1> <public key 2> 2 OP_CHECKMULTISIG
```

82 33 2 211 153 245 240 225 125 95 140 116 20 99 81 38 139 135 136 59 14 125 34 181 148 47 67 16 42 24 147 28 144 61 33 33 2 215 1 70 141 233 112 91 253 252 202 27 73 158 254 234 159 125 98 30 78 159 235 6 46 167 103 105 239 180 125 168 66 82 174

The opcodes and their ordering in the script dictate the functionality of the script. To identify a type of script we need to look for scripts with the same format and operations. The script consists of two numbers denoting the required and total number of keys required to create signatures, the public keys for the key pairs that can be used, and the `OP_CHECKMULTISIG` operation. The required and the total number of keys required is, in our case, 2, as we are only looking for 2of2 scripts. As shown above, the script starts with the required number of keys which must be used for signing, in this case 2, being 82 in the byte version. Then there is the first public key of the key pair, able to sign. In the byte version, the second byte is 33, which indicates how many bytes to push to the stack; this is the length of a compressed public key with the prefix. Bitcoin uses elliptic curve cryptography, where a public key is two coordinates representing a point on the curve. The prefix for compressed keys can be 02 or 03, indicating if the y coordinate is even or odd [8]. So, the first byte in a compressed public key, and the third byte in this type of script will always be 2 or 3, and the 32 next bytes will be the rest of the compressed key `<public key 1>`. Then we have the other public key `<public key 2>` in the same format, followed by 2 or 82 in byte format, telling us the total number of keys which can be used for signing. At the end we have the opcode 174 `OP_CHECKMULTISIG`, which checks signatures against the public keys. To identify this type of script we check if the opcodes are present at their expected locations in the script, and the total length of the script. This means we check if certain indexes in the byte array contain the data we expect—e.g., we would expect the last index in the byte array to contain 174.

When using 2of2 multisig redeem script, which also entails using `P2WSH` for identifying potential channels, our results contain all actual LN channels, but likely also many that are not. The precision of this will depend on the current use cases for 2of2 multisig transactions besides on-chain LN transactions, and how commonly they are used by other Bitcoin users. To determine the current effectiveness of this method, we have compared data from the blockchain with data gathered through the LN itself, and with results from our other method. This allowed us to quantify the actual number of channels identified using this method, compared to the number of false positives. The Section 4.2 contains a more in-depth discussion on the precision and sensitivity of the heuristics used.

4.1.2. Identification Using Timelocked Redeem Scripts

The main method for identifying LN channels based on Blockchain data relies on timelocked redeem scripts. While more unique than the 2of2 multisig redeem scripts, it only exists on channels that are unilaterally closed—i.e., the channel is closed by one of the parties, and not cooperatively. In Figure 6 we can see how a channel graph is structured when this method is used. We still have the funding and closing transactions, but the closing transaction is also a commitment transaction, as it was used to close the channel. When a commitment transaction is published, the output to the entity closing the channel will be timelocked, as we explained in Section 2.4. This is to enable the other party to spend the output using the revocation key, in the case of the commitment being revoked—i.e., any commitment other the most recent. This is implemented using a `P2WSH` output, which has a redeem script containing a timelock, and a clause for the revocation key. As redeem scripts are only available in the input of the transaction spending the `P2WSH` output, similarly as with the 2of2 multisig redeem script, we find this script in the transaction spending the timelocked output. We refer to this transaction as the timelocked transaction, which can be seen on the right in Figure 6.

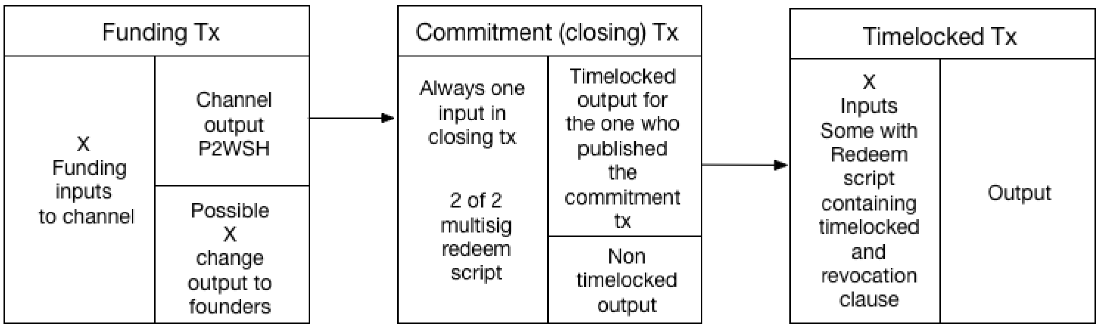


Figure 6. Channel graph of unilaterally closed channel.

While the timelocked transaction containing the redeem script is not present in channels closed cooperatively, the uniqueness of the script ensures the transactions that are identified using this characteristic are likely to be actual timelocked transactions from the LN. When a timelocked transaction is identified, we can use its input where we found the redeem script to get the hash of the closing/commitment transaction. We can, subsequently, use the input of the closing/commitment transaction to get the funding transaction. This way we can get all transactions in the channel graph. We can also use the fact that the closing/commitment transaction has the 2of2 multisig redeem script. By checking that the closing/commitment transaction has this script in its input, same as we did in the multisig method, we are further ensuring that we have found a LN channel. Even with the timelocked scripts being unique, because of its specific use case, users are free to create the scripts they wish, so the presence of such a script will not guarantee that we have found a LN timelocked transaction. While also LN-related, we see in Section 4.1.3 how scripts with the same purpose are used in different transactions. By checking that the closing transaction has the characteristics we expect, we can avoid mistaking instances where a timelocked script is used in other scenarios, rather than in the LN context.

This whole process can be seen in Figure 7, which is similar to the one in Figure 5 (but more complex).

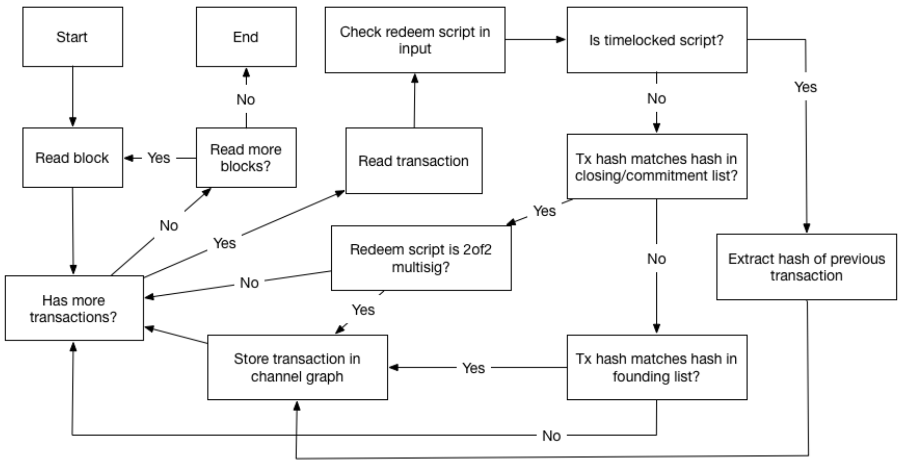

Start
End
Check redeem script in input
Is timelocked script?
Yes
No
Read block
Yes
Read more blocks?
Read transaction
Tx hash matches hash in closing/commitment list?
No
Yes
Yes
Redeem script is 2of2 multisig?
No
Extract hash of previous transaction
Has more transactions?
No
Yes
Store transaction in channel graph
Yes
Tx hash matches hash in founding list?
No

Figure 7. Algorithm for parsing blockchain and creating channel graphs using the timelocked method.

The timelocked redeem script defined in [13], is the redeem script used to identify the timelocked transactions, and can be seen below:

```
OP_IF
    # Penalty transaction
    <revocationpubkey>
OP_ELSE
    ‘to_self_delay‘
    OP_CSV
    OP_DROP
    <local_delayedpubkey>
OP_ENDIF
OP_CHECKSIG
```

The script contains an `if-else` clause, with the `TRUE` clause being the one which allows the opposing party to spend the output in the case of the commitment transaction being revoked. By supplying `<revocation_sig> 1` with a valid signature, the script will evaluate to true. The constant 1 will make the script evaluate the first clause, and then the signature will be matched to the `<revocationpubkey>` using `OP_CHECKSIG`. The else clause is the timelocked part of the script. It will contain the timelock, enforced by the `CHECKSEQUENCEVERIFY` operation defined in [20], which will terminate the script if the specified delay has not passed. The delay is then removed from the stack with `OP_DROP` such that a signature and public key are the only two items on the stack. To make the script use the else clause with the timelock, and check the signature against the `<local_delayedpubkey>` the witness data should be: `<local_delayedsig> 0`. Similar to the revocation clause, we have a signature, and the constant 0 will make the script not use the if clause. Meaning the `OP_CHECKSIG` will be used to verify the `<local_delayedsig>` against the `<local_delayedpubkey>`, as long as the timelock has expired. This means we can easily check if a timelocked script was revoked. By checking the witness data provided to the script for 0 or 1, we can determine which clause was used, and, as it is included in the blockchain, the script has been evaluated and found to be valid.

Below is the raw byte representation of a specific redeem script of the type shown above. We depict it because our implementation handles the byte version of the scripts when identifying them. The byte version has been formatted in the same manner as the clear-text version, to allow for easier comparison.

```
99
    33 3 251 83 243 198 231 109 204 252 217 94 44 221 0 255 185 86 106 105 161 141 254 96     167 77 48
16 57 146 128 4 80 1
103
    82
    178
    117
    33 2 159 6 236 212 233 63 48 147 59 52 201 11 15 138 165 248 118 100 188 234 227 215     108 160
135 22 57 37 117 250 172 130
104
172
```

The first byte seen in the byte script above is 99 which is the opcode for the `OP_IF` operation. The next is 33, which indicates how many bytes to push. Same as explained earlier for the 2of2 multisig script: this is the length of a compressed public key with a prefix. So, the next 33 bytes are the `<revocationpubkey>`. After the public key, we have the `OP_ELSE` with the opcode 103, followed by the 82 'to_self_delay' which is the delay used for the `OP_CHECKSEQUENCEVERIFY` (`OP_CSV`) operation, with opcode 178. Next byte is 117 which is the `OP_DROP` operation, and after that, we find the `<local_delayedpubkey>` in the same format as `<revocationpubkey>`; 33 indicating the number of bytes to push to the stack followed by the prefix and the rest of the key. After that, we have 104 which is the opcode for `OP_ENDIF`, and 172 for `OP_CHECKSIG`. To identify this type of redeem script in timelocked transactions we check if the opcodes can be found at the expected locations, e.g., index 0 of the byte array should be 99 for the `OP_IF`, and 103 for the `OP_ELSE` operation should be at index 35, allowing space for a public key between it and the `OP_IF` operation. Each opcode found in this script type is used for recognition, meaning all opcodes covered there must be present in the script for it to be identified as a timelocked redeem script, and they must also be found at their expected location relative to the others.

Because transactions in a block can have any order, and both funding and closing transaction can be placed in the same block, we had to conduct two passes over transactions in a block, o iterate over the transactions in the block a Tsecond time if we found any LN-related transactions, to make sure we did not miss the preceding transaction earlier in the block. Even with parsing some blocks twice the software parsed blocks fairly quickly. We could achieve parsing rates of 100 million transactions in under 90 min. The memory consumption was also low, as our software only stored channel graphs and the hash lists for transactions we were currently looking for.

4.1.3. HTLC On-Chain

If the HTLC cannot be resolved off-chain, the channel will be closed and the HTLC will be handled on-chain. This will be a `P2WSH` output from the commitment/closing transaction, with a redeem script allowing users to spend it in two ways: by either supplying the preimage R used to create the HTLC or by waiting for the timelock to expire, which will create an additional timelock. There are two versions of HTLC redeem scripts used in on-chain transactions, defined in the BOLT rfc [13]. Which of the two types is used depends on the fact if the party that published the commitment transaction to the blockchain is sending or receiving funds using the HTLC. These scripts are even more complex than the timelocked scripts, which would make them good at identifying LN-related transactions. Below is the script used for the HTLC output when the publisher of the commitment has sent an HTLC to the other party:

```
# To remote node with revocation key
OP_DUP OP_HASH160 <RIPEMD160(SHA256(revocationpubkey))> OP_EQUAL
OP_IF
    OP_CHECKSIG
OP_ELSE
    <remote_htlcpubkey> OP_SWAP OP_SIZE 32 OP_EQUAL
    OP_NOTIF
        # To local node via HTLC-timeout transaction (timelocked).
        OP_DROP 2 OP_SWAP <local_htlcpubkey> 2 OP_CHECKMULTISIG
    OP_ELSE
        # To remote node with preimage.
        OP_HASH160 <RIPEMD160(payment_hash)> OP_EQUALVERIFY
        OP_CHECKSIG
    OP_ENDIF
OP_ENDIF
```

The script first checks if it was provided with the public revocation key. If this is the case, the first `if` clause will be used to check if the signature provided matches the public key. This will only happen if the HTLC was part of a revoked commitment transaction. If this is not the case the else clause is used, which will check if the HTLC public key of the party not publishing the commitment was provided. If this key is provided it means that the party can use the preimage `R`. The publisher can make the HTLC timeout by using the `notif` clause.

Our software can recognize these scripts using similar methods as discussed in Sections 4.1.1 and 4.1.2. They are, however, not used for identification of channels, and thus channel graph creation. Our focus has been on timelocked transactions as the main method for identification, as it provides a good mix of precision and sensitivity. Our hypothesis was that HTLC outputs, while being unique, are not nearly as frequently found on the blockchain as timelocked transactions. Additionally, the transactions containing redeem script such as the one above would spend commitment/closing transactions we would likely find using timelocked transactions. By implementing the basic recognition of the scripts, we could check how often they occur, and by comparing their inputs to our already identified closing transactions we can determine how many of the HTLC transactions would identify channels we could find using the timelocked method.

The transactions spending an HTLC output will have an additional transaction with a timelock if the party that published the commitment spends the HTLC. This timelock will use the same script as the timelocked transaction spending a commitment output. After identifying a timelocked transaction, we would check if the preceding transaction had a 2of2 multisig script, indicating it was a closing transaction. Not doing this, we would create channel graphs where the timelocked transaction was an HTLC timelocked transaction, the closing transaction an HTLC transaction, and the funding transaction a closing transaction. This example demonstrate that a specific script does not necessarily mean the transaction is the one we expect.

4.1.4. LN Information on the Blockchain

If we can successfully identify channel graphs containing transactions related to the LN, the data in those transactions will allow us to extract some information about the channel. The value in the output-input pair used for the channel will tell us the total value of the channel. Timestamps are included in blocks when they are created, so by checking the timestamp of the blocks where the funding and closing transactions are located we can see how long the channel was active. The number of inputs in the funding transaction and the number of outputs in the closing is also interesting; a funding transaction with a single input shows that a single user funded the channel, whereas multiple inputs can indicate that multiple users funded the channel, but there is also a possibility that the channel is still funded by a single user, with multiple smaller outputs to get the desired value of the channel. It is difficult to determine how funds have moved inside the channel based on the value of the inputs to the funding transaction, and the value of the outputs from the closing transaction—i.e., it is hard to extract any information about any off-chain transactions by analyzing the outputs and inputs of the on-chain transactions. The reason for this is that normally we cannot determine which input/output belong to which user in the channel. We discussed in Section 2.2 how keys are used to lock outputs, and signatures using those keys are used in inputs to unlock, meaning a key pair is related to an input-output. In Figure 8 we have a funding transaction with two inputs (keys), a multisig output-input with two keys, and a closing transaction with two outputs (keys). If we assume each user funded the channel with one input we can determine the initial balance between the parties by checking the value of those inputs. Comparing that balance to the one in the two outputs will tell us how it has shifted from start of the channel to the end, but as discussed in Section 3 the pseudonymity provided by different key pairs makes it impossible to see which of the inputs corresponds to which output. We can also see in Figure 8 how the value is initially spread in two outputs then merged in the channel and then spread out again in two outputs, so simply following a specific value will not work. Linking the keys used in these transactions as is done in previous work discussed in Section 3 will make it possible and is something we will discuss in Section 4.3. The one thing we can determine by looking at inputs and outputs of a channel, without any key linking, is that an off-chain transaction has taken place in the channel, but this is only the case if there is one input to the funding transaction (single funded), and there are multiple outputs, meaning we know the channel started with all value belonging to one party and it ends with the parties splitting the value.

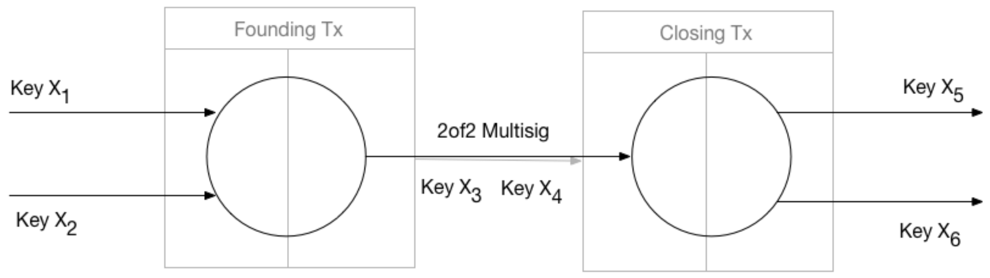


Figure 8. Keys in a channel graph.

We discussed in Section 4.1.1 how using `P2WSH` outputs would allow us to identify possible open channels, as all outputs used for LN channels are of this type. With those types of transactions fairly common in the blockchain, we are most likely to get many false positives, but we can nevertheless use this to determine the maximum size (number of channels) of the LN, i.e., establish an upper bound. By counting the number of unspent `P2WSH` outputs we can get the maximum number of open channels. This can also be done to get the maximum number of LN channels at any historical point of the blockchain; by counting all `P2WSH` outputs and subtracting the spent ones up to a height on the blockchain will get us the unspent outputs at that point. While this might not be accurate in terms of the actual number of channels (size) for the LN, it can provide a concrete upper limit to its size, both current and at any point in the past. We can also use data collected directly from the LN to correlate between the number of unspent `P2WSH` outputs and open channels in the LN at different points in time. This will indicate how closely the number of `P2WSH` outputs reflects the number of LN channels.

4.2. Lightning Network Data Collection

To collect ground truth data, we have modified LN daemon to store public advertisements about the topology of the network. This data is more complete than what we can find on the blockchain and can thus provide us with a more accurate picture. By comparing blockchain data with data directly from the LN, the effectiveness of our methods can be measured. In Section 4.1.1 we have discussed our multisig detection method. We stated that it would create some channel graphs which might not be the actual LN channels. By comparing the potential channels, we find using this method with the ones gathered directly from the LN, we can determine how many of our results are false positives. The data from the LN can also provide us with the ideal result of linking information found on the blockchain. This is because some relations are explicitly defined in the LN data, but only implicitly or not at all on the blockchain, which makes our linking efforts on the blockchain data also measurable.

To collect data from the LN we used one of the implementations following the BOLT specification [21], called LND [22], which is an open source project written in Golang. We modified the implementation to collect the data we were interested in. LND maintains a view of the current LN by storing a graph containing active channels and nodes. This is continuously updated as new channels are announced through the network, and closing transactions are found on new blocks on the blockchain. It requires a Bitcoin instance running to interact with the Bitcoin network, which is used to publish on-chain transactions and monitor the blockchain. New channels are discovered by announcements within the LN, while channels closing is found by locating closing transactions in new blocks from the Bitcoin network, i.e., transactions spending outputs of funding transactions. We modified the version 0.4.1-beta of LND to create a copy of its database each time a new block notification was received from the Bitcoin software. This was not done immediately but within a minute so the changes from the block could be applied to the graph data. Doing this gave us a set of databases containing the state of the network at each block, essentially a set of snapshots of the Lightning network, as seen from within the network.

The snapshots contain the graph describing the state of the Lightning network as known to our node at the time of the snapshot, so by comparing the graphs of different snapshots, we can see how the network evolves over time. Each graph in the snapshots is essentially a set of channels active at that time. Consider two sets of channels representing the LN at different points in time, κ being the older and τ the newer. The channels not present in κ but present in τ would be new channels, i.e., the relative complement of κ in τ, $\tau \setminus \kappa$. Similarly, the channels closed would be the ones present in κ but not in τ, i.e., the relative complement of τ in κ, $\kappa \setminus \tau$. Doing this for each snapshot we can get a set of channels closed and opened during the data collection interval, and because we collect snapshot at each new block, we can easily contextualize the results with different block heights, e.g., create a list of closed channels for each block or number of active channels for each block height. This also makes it easy to compare data from the LN to data from the blockchain as we can use the block height as an index, ensuring we compare correct data. Using some libraries from the LND implementation, we created an additional piece of software to read the data generated by the node software, compare the snapshots, process the data, and produce outputs with the desired information.

An additional modification we have made to the LND implementation besides generating snapshots for the LN network state was removing the process of pruning the *zombie channels* from the graph. According to the BOLT RFC [23], nodes can remove channels if the last channel update is older than two weeks. This is done to avoid open but abandoned or unusable channels to remain in the graph and be propagated to others. The LND implementation follows this recommendation and does this regularly. Our reasoning behind disabling this for our node was to keep data from the LN consistent with the blockchain data. Pruning channels would make changes to the LN graph, making it seem like channels were closed whereas from the blockchain data they appear open. Because we are focusing on the blockchain-centric LN analysis, we chose to keep these channels in the graph.

4.3. Linking and Key Reuse

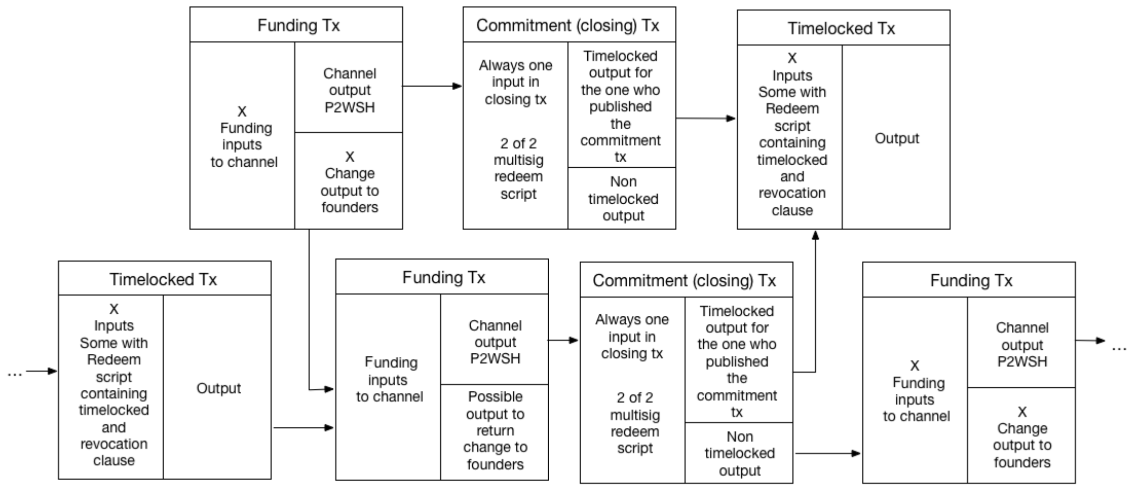
Linking and clustering information that is related has been the focus of much of the related work described in Section 3. The main goal of clustering is to reveal non-explicit information, which, in the context of the Bitcoin transaction graph, would be constructing user profiles. This is done by linking keys controlled by the same user, which will de-anonymize the pseudo-anonymous properties of the Bitcoin system. For this work, we are interested in the subset of Bitcoin users that is also LN users, in addition to information about the LN itself. Our methods for locating the relevant information have been outlined in Section 4.1, the result of which is channel graphs containing the on-chain transactions related to one LN channel. As we do not take the entire Bitcoin transaction graph into consideration we are limited to using the data found inside channel graphs for linking purposes. The main focus of linking in this project is to find relationships between channel graphs, allowing us to link them based on shared user participation. We rely on the fact that to be a usable network, the LN needs nodes/users to have multiple channels to different users—i.e., some vertices will need to be of a degree higher than one. This relationship between channels is clearly visible from the LN perspective, as the view of the network will contain this relationship, but this is not the case for the blockchain. As we stated in the start of this chapter, the Bitcoin system is unaware of the existence and state of the LN. There is no explicit relationship based on common node/user participation for channel graphs on the blockchain. As the on-chain LN transaction are mainly for managing channels, we are limited to linking information about channels or users within the channels. The off-chain transactions are not available to use for any linking.

While we can link channels if the same user is participating in both, we cannot determine which of the two users in either channel creates this relationship. The reason for this is the same as discussed in Section 4.1.4, where we pointed out the problems of matching inputs to outputs of a channel, for the analysis of changes in the balance between the users. In previous research (discussed in Section 3) users are defined by control of a key pair. Essentially, a user is defined as the ownership of a private key. Linking keys reduces the number of users by redefining users as a set of key pairs, e.g., a transaction graph with four keys, would mean there were four potential users, unless we managed to link keys. For our case with the channel graphs, a problem arises because we know in advance how many users are involved in the graph. There are two users involved in a channel, but usually many more keys present, so we have a reduced user count, but this is not due to linking keys. Defining a user in this setting is harder, as we cannot simply choose an arbitrary key for each of the two users. In Figure 8 we see how there are 6 keys which normally would represent a user each. In our setting we know there are only two users. If we could link the keys, resulting in two sets of keys, then these sets would enable us to distinguish the users. However, without two key sets, when linking channels based on a property in the channel graph, we cannot determine which of the users this property belongs to as we have no clear definition of the users based on the keysets from LN.

We have created three heuristics for linking channel graphs. These are only for linking channel graphs based on common user participation; two of them are related to the heuristics discussed in Section 3 used in previous work. The heuristics employed are:

1. **Key reuse**
   Check if the same keys are present in different channel graphs. Any key found within a channel graph is sure to belong to one of the two users participating in the channel, so if the same key can be found in another part of the graph, the user owning the key must participate in both channels. This heuristic allows us to link sets of transactions—i.e., transaction graphs representing channels. This heuristic reveals user involvement in channels. Because we know there is only two people participating in transactions within each channel, keys from any transaction inside the graph can be checked for reuse. The keys used in the funding and closing transactions clearly belong to the participants, but with the timelocked transactions, we cannot be certain. The BOLT RFC [24] strongly recommends the owner to spend the timelocked output when it expires to transfer the value to a convenient address, this is because of the advanced redeem script required to spend the timelocked output. This advanced script must be kept until the output is spent, which is inconvenient compared to having the value available in a normal `P2WPKH` output. Assuming all LN implementations follow this recommendation, the timelocked transaction and the keys found should only contain keys related to the user which claimed the timelocked output. We should, however, note that the key reuse needs to happen in different channels, as key reuse within a single channel does not allow us to link channels.
2. **Graph connections**
   This heuristic is based on channel graphs being directly connected with each other by output-input pairs. Such a connection can be found in any outputs going to transactions outside the graph. This means all outputs in the funding transaction except the 2of2 multisig used for the channel, the non-timelocked output in the closing transaction, and the outputs of the timelocked transaction can all be used for linking. In Figure 9 we can see the connections—e.g., the non-channel output of a funding transaction is used for input in another funding transaction. The reason we can determine shared user participation in connected channel graphs is that they are directly connected. By inferring the purpose of outputs from the channel graph we can determine that the outputs are controlled by the participants in the channel, so if they connect to another graph we can say there is common user participation. This concept is the same as the one used in previous work with heuristic 2 discussed in Section 3, where some outputs from a transaction were assumed to be the change for the sender. The non-channel outputs of the funding transaction should be treated as the change back to the participants, as they may not want to use the entire value of the inputs in their channel. It is unlikely that the outputs are transfers of funds to another entity unrelated to the channel, so we can safely assume that the outputs are controlled by one of the entities in the channel, meaning if they are used as input in another channel, it would tell us that the same user is participating in both. For the non-timelocked output in the closing transaction, we know it must be controlled by one of the users. The control of the outputs of the timelocked transaction is based on the recommendation from the BOLT RFC [23] which was discussed in the previous heuristic.

3. **Graph overlap**
The third heuristic is when channel graphs overlap with each other, meaning a single transaction is part of multiple graphs. This can only occur with the timelocked transactions, as funding and closing transactions are used for controlling a single channel. We can see this overlap illustrated in Figure 9 where the timelocked transaction in the top right has inputs from two commitment/closing transactions and will thus be part of both channel graphs. This relates to heuristic used in previous work as outlined in Section 3, where transactions with multiple inputs were used to link keys used in those inputs. By being part of multiple graphs, a timelocked transaction will have multiple inputs. Which according to the old heuristic 2 tells us that the same user is in control.



Funding Tx
X Funding inputs to channel
Channel output P2WSH
X Change output to founders
Commitment (closing) Tx
Always one input in closing tx
2 of 2 multisig redeem script
Timelocked output for the one who published the commitment tx
Non timelocked output
Timelocked Tx
X Inputs Some with Redeem script containing timelocked and revocation clause
Output
...
Timelocked Tx
X Inputs Some with Redeem script containing timelocked and revocation clause
Output
Funding Tx
Funding inputs to channel
Channel output P2WSH
Possible output to return change to founders
Commitment (closing) Tx
Always one input in closing tx
2 of 2 multisig redeem script
Timelocked output for the one who published the commitment tx
Non timelocked output
Funding Tx
X Funding inputs to channel
Channel output P2WSH
X Change output to founders
...

Figure 9. Connected channel graphs.

Linking channels based on shared user participation allows us to create a channel network depicting how the channels are related to each other. In this graph, the vertices represent channels and the edges represent users/nodes in the LN. While the transaction graph for Bitcoin allows us to see how funds are moving by giving us the structure of the transaction graph, the channel network can provide us with a visualization of how funds can flow between channels based on their connections. As the LN is a network, it has a natural graph structure where vertices are users/nodes and the channels are edges between the users. We will refer to it as the LN graph. Note, LN graph uses a reverse notation in terms of vertex-edges compared to our channel network. Another difference between the LN graph and our channel network is that the LN graph requires a distinction between users/nodes as the vertices in that graph are specific users; in our channel network the edges are any of the two users participating in the channel.

Having a clear definition of users in the context of channels identified on the blockchain, as discussed earlier, allows us to differentiate the users. We can determine which channels a specific user participates in. This allows us to create a network using channels we have identified on-chain, with the same structure as the standard LN view—i.e., vertices representing users and edges being channels. Including the temporal aspects of the channels would allow one to recreate the structure of the LN at different points in time. One approach for achieving a clear distinction between the two users in the channel graphs is to disregard keys and related transactions; if we only use the 2of2 multisig output-input constituting the channel, we will only have two keys, one for each user. While this removal of information gives us a clear definition of a user in the channel, it also removes many possibilities for linking the channel to other channels. The only heuristic still viable if we only consider the channel output-input is key reuse of the keys used to define the users. This means we are not able to link as many channels as previously, where we would have more keys to check for reuse. Another possibility is to link all keys within a channel graph, resulting in two sets, each representing the user as we discussed previously. This approach would allow us to use the heuristics presented above and differentiate user participation. A hybrid between the two would also be possible: linking keys to the extent we are able to do and then disregarding information related to the keys we are unable to link to any set. The heuristics could still work using this approach, depending on the extent we are able to link keys, and which keys we are able to link. Ultimately, linking keys should be done by using the entire transaction graph. Doing this provides more information for linking possibilities than taking into consideration the small number of transactions inside the channel graphs.

5. Results

Comparing data from the LN with our data from the blockchain, provided us with results about the extent our identification methods can identify LN channels on the blockchain. There were three sets of data used in these comparisons:

- The set α, containing channels from the LN closed during the capture interval.
- The set β, containing channels from the blockchain identified using timelocked redeem scripts.
- The set γ, containing potential channels identified on the blockchain using 2of2 multisig scripts.

Because the on-chain channel transactions must be of the specific type, the γ set is guaranteed to include all LN channels closed in the block interval used for the search. It also means that the two other sets will be the subsets of γ. Thus, the relationships between the sets should be as follows:

$$\alpha \subseteq \gamma, \quad \beta \subseteq \gamma, \quad \beta \subseteq \alpha \tag{1}$$

5.1. First Data Collection Interval

The first collection interval on the mainnet had a length of 1151 blocks. It was between Bitcoin block 517,855 and 519,005. Our modified LND implementation collected data from the LN and produced the set α containing 715 channels which were closed during this interval. The same interval on the blockchain was parsed two times, once identifying channels using timelocked redeem scripts, and the second time getting all potential channels using multisig identification. This resulted in a β set containing 322 channels and a γ set with 1265 channels shown on the left in the Venn diagram Figure 10. We can see how both β and α are a subset of γ, but $\beta \nsubseteq \alpha$. The intersection $\beta \cap \alpha$ is the number of channels we identified using the timelocked redeem script which was also found through the LN node. On the mainnet, this intersection was 190 which is 36% of the total channels in α. This is reasonable, as the method only discovers a channel if it has been unilaterally closed as we explained in Section 4.1.2, and not if a channel is closed cooperatively which happens more frequently (64% of the time, to be precise). Taking the ideal world scenario $\beta \subseteq \alpha$ discussed above, into account for this data, we have a large $\beta \backslash \alpha$ relative complement of α in β of 132, which is 40% of the channels in β. With a false positive this high is unlikely because the uniqueness of the timelocked redeem scripts, so we assumed that these are actually LN channels which we have not been able to capture in our interaction with the LN. The reason we made this assumption was that it is more likely that not every channel is propagated successfully to our single LN node than there being this many instances of timelocked redeem scripts unrelated to the LN on the blockchain. Additionally, as we explained in Section 4.2 at least every node running the LND implementation will prune their LN graph of *zombie channels* regularly, so these will eventually no longer be propagated through the network. This means even if our node does not prune its graph of the network, the channel will never be received in the first place. These are the likely causes of our timelocked channels not being found in the LN channel set. Taking this into account and assuming the union $\alpha \cup \beta$ are all LN channels, the total number of LN channels closed in this interval would be 847. This would entail that 67% of 2of2 multisig transactions in our interval are lightning channels.

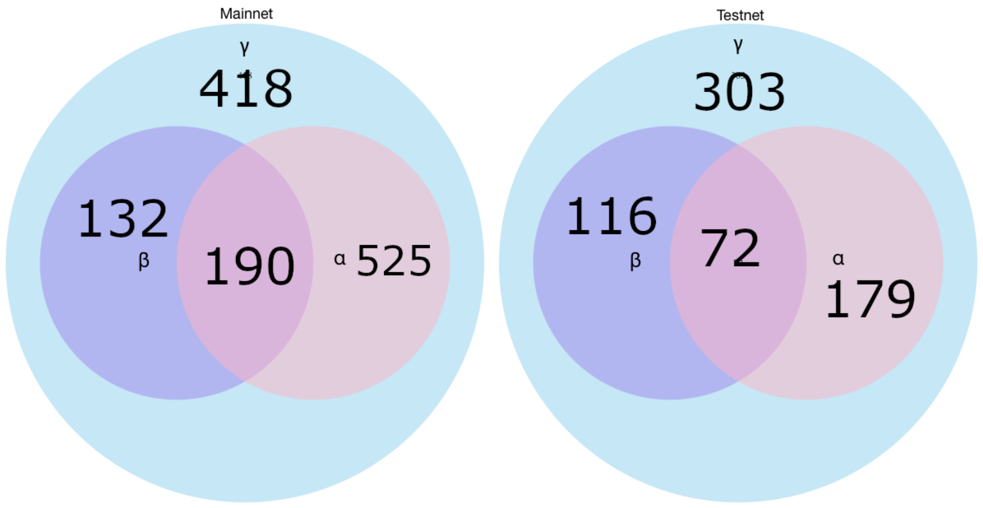


Figure 10. Venn diagram of channel sets in interval one.

For the testnet, our LND node collected 838 snapshots starting on block 1,290,011 and ending on block 1,289,174. The results are similar to the mainnet, with the intersection $\beta \cap \alpha$ being 27% of α, and the $\beta \setminus \alpha$ relative complement of α in β being 62% of channels in β. Again, assuming the union $\alpha \cup \beta$ are all LN channels, 55% of 2of2 multisig transactions are lightning channels. This is however not an upper limit, as our $\alpha \cup \beta$ union is the channels unilaterally closed on the blockchain and channels that have been propagated to our node in the LN. Our LN node might not be able to get all channels, so there should also be channels not closed unilaterally and therefore not discovered on the blockchain, which we similarly have not been able to collect through the LN, which would make the total channel count higher, and more of the 2of2 multisig transactions being channels than indicated in our results for both the networks.

5.2. Second Data Collection Interval

For our second data collection interval, we only collected data from the mainnet. We have tweaked the collection slightly. Instead of only peering with the one assigned node when starting the LND software, we added more, up to a total of 10 peers for the duration of the interval. We also ran the node for a few days before starting the collection, so the node would have opportunity to synchronize. A total of 1151 snapshots of the LN was created, giving α 446 closed channels. The results are summed up in Figure 11. The timelocked identification on the blockchain found 168 channels for β, while the multisig identification resulted in 667 channels for γ. For this interval, the intersection $\beta \cap \alpha$ was 26% of α, which is similar to the first interval, with 36% and 27% for the mainnet and testnet, respectively. However, the $\beta \setminus \alpha$ relative complement of α in β, was 30% of channels in β, compared to 42% and 63% in the first interval. Also, if the union $\alpha \cup \beta$ all is LN channels, it would entail 75% of the 2of2 multisig transactions are lightning channels. This would indicate we have been able to collect more channels from the LN, which the size of α compared to γ also confirms: in this interval α is 67% of γ, while in the first interval this was between 55–56% for both networks. This supports our suggestion made in the previous subsection about there being several channels which we are unable to collect through the LN. Ensuring that the collecting node can synchronize as much as possible is important. This would also mean that more than 75% of 2of2 multisig transaction are likely LN channels.

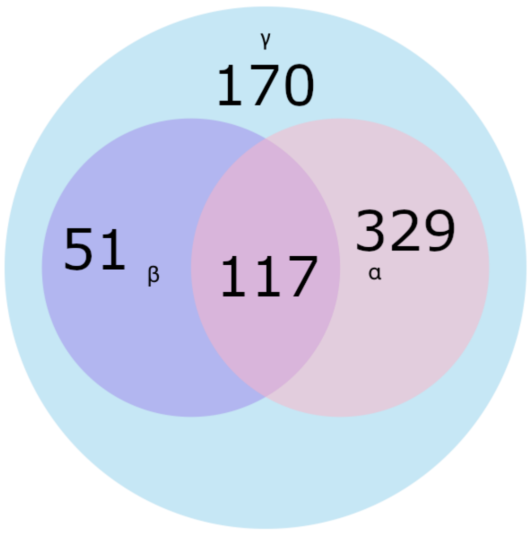


Figure 11. Venn diagram of channel sets during interval two.

5.3. Lightning Network Size

While parsing the blockchain we have counted all unspent `P2WSH` outputs for each block height, in both the mainnet and the testnet. The results of this can be found in Figure 12, as a graph showing unspent `P2WSH` outputs for different block heights. We can see a distinct difference between the two networks: the mainnet has a more organic growth as a result of real-world use, while the testnet graph is characterized by the testing which the network is used for. The sharp decline in unspent outputs on the mainnet at around block 505,000 can be explained by the sharp decline in fees for doing transactions at that time [25], which incentivized users to consolidate their unspent outputs into fewer bigger ones.

While the unspent output count provides a concrete upper limit on the number of channels and therefore the size of the LN, we also wanted to know to which degree the amount of unspent `P2WSH` transactions was correlated with the size of the LN. It is clear that changes in the size of the LN will impact the number of `P2WSH` outputs/inputs, but there might be many other transactions not related to the LN having a larger impact. From the data we collected from the LN itself we constructed a total channel count for each block height. Similarly, we isolated the same block interval with the `P2WSH` output count from the blockchain. These two variables allowed us to compare the changes in the number of channels in the LN and number of unspent `P2WSH` outputs. We used the Kendall rank correlation coefficient on this data for the mainnet, which gave us a correlation coefficient of 0.73 indicating a strong positive correlation. The scatter plot for this can be seen in Figure 13. We did the same for the interval of testnet data we collected, which resulted in a coefficient of 0.45 which is only a moderate correlation. The scatter plot for the testnet can be seen in Figure 14. Both tests have the same *p*-value: $p < 2.2 \times 10^{-16}$. While the two are correlated naturally due to the need for channel outputs to be `P2WSH`, the results here indicate there is indeed a strong statistical correlation. The testnet showing a weaker correlation can be due to the unnatural use taking place on the testnet, as was apparent from Figure 12 graph showing unspent output for different block heights. Additionally, the correlation was limited to the amount of data from our collection intervals, meaning we could only correlate the two for around 1000 blocks and less.

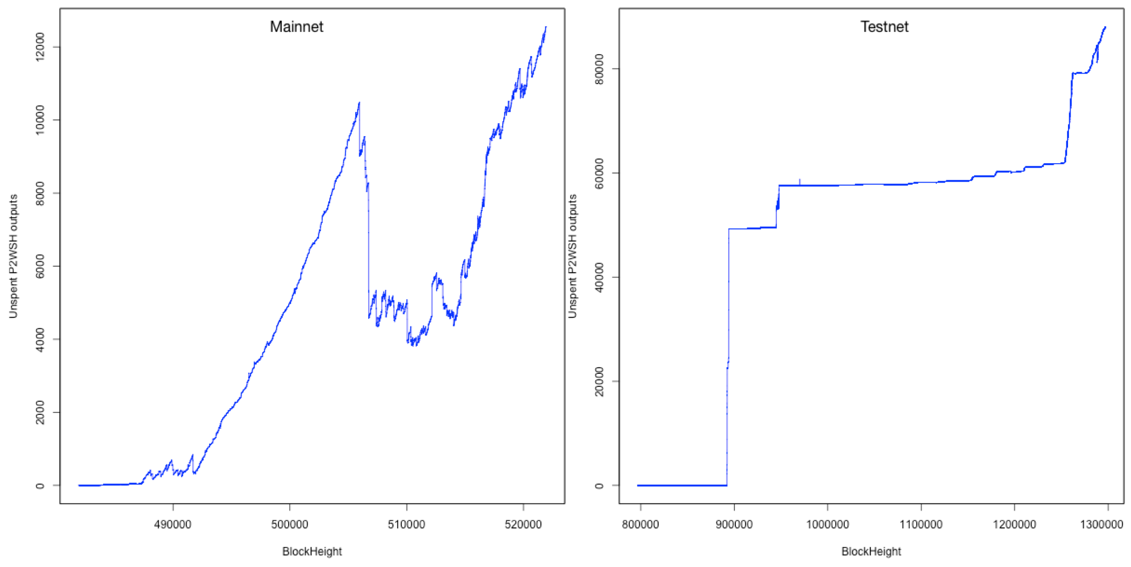


Figure 12. Maximum size of the LN based on unspent P2WSH outputs on the blockchain.

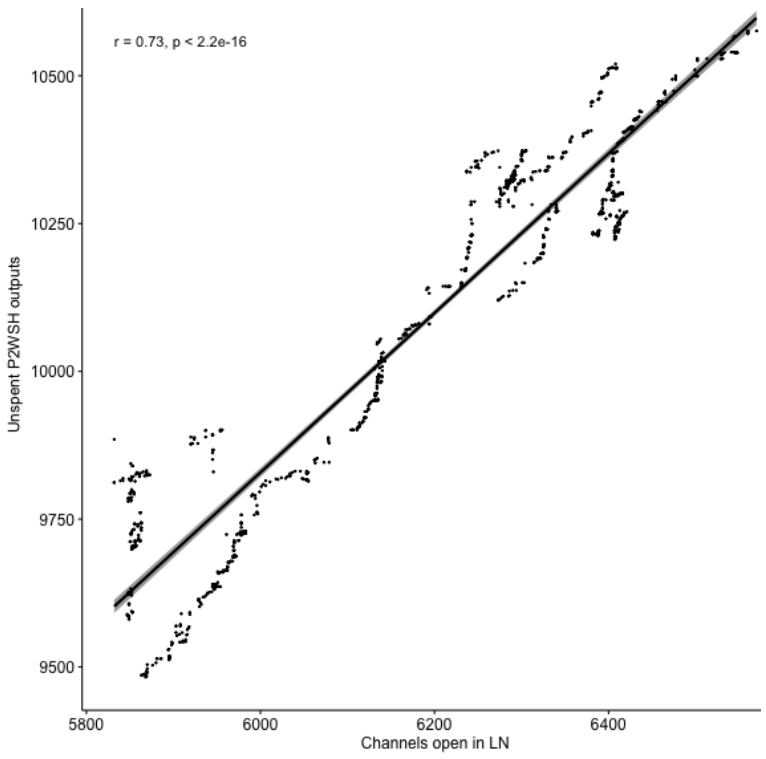


Figure 13. Correlation between unspent P2WSH outputs and channels open in the LN mainnet.

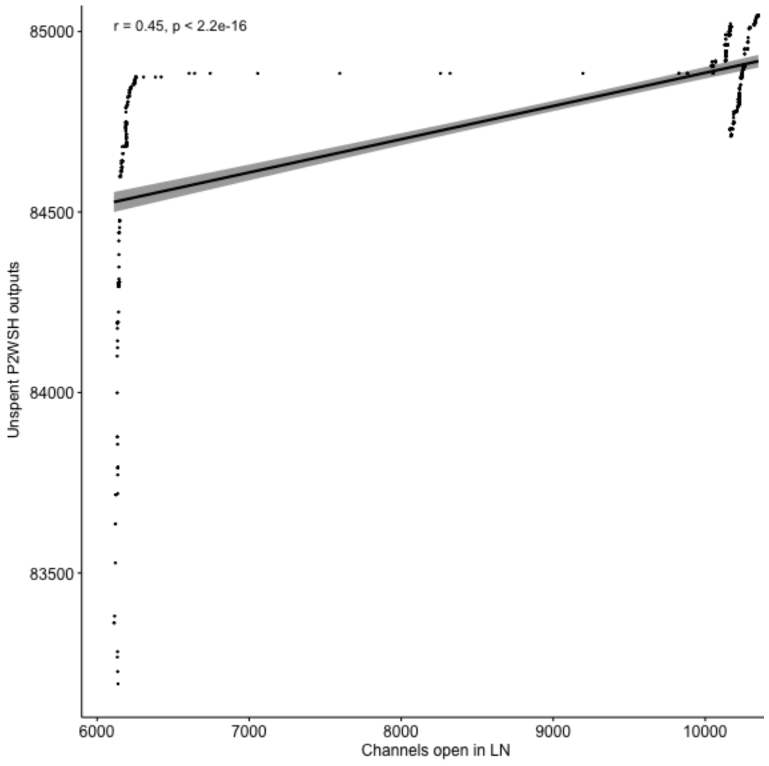


Figure 14. Correlation between unspent P2WSH outputs and LN channels on the testnet.

5.4. Stats from Complete Runs

We also tried to run our software with the timelocked redeem scripts method for as long as there were any `P2WSH` transactions on the blockchain, to collect as many channels as possible. This method is precise and can find a fair amount of channels. The run resulted in a set of channel graphs for every unilaterally closed channel that has existed up to the point we started parsing. On the mainnet, we started on block 524,816 and ended on block 464,816 resulting in a total of 3809 channels found. For each timelocked redeem script we also checked how the script was evaluated, telling us if the revocation or timelocked clause were used. On the mainnet we found that 163 timelocked outputs were spent with the revocation key, meaning at least this many old commitment transactions were published. Users could spend unconfirmed transactions, resulting in the transactions being found within the same block. We found a total of 29 such instances, only checking the transactions within our channel graphs. Regarding HTLC frequency on-chain, we found a total of 163 transactions containing HTLC redeem scripts. 101 of these turned out to be spending outputs from closing transactions of graphs we already had identified, which means we could have identified 62 more channels by also using this method. Doing the same for the testnet resulted in 8047 channels found, starting on block 1,316,196, and ending on block 816,196. Here we found 38 timelocked outputs spent with the revocation key, and 430 of our transactions being found within the same block. On the testnet, we found 566 HTLC transactions, with 143 of them not being connected to any of the channel graphs we had identified. Using these two sets of channel graphs, we extracted stats about the channels as discussed in Section 4.1.4. This gave us data on the value used in each channel as the amount of Bitcoin (BTC), and the lifetime of channels in terms of the number of blocks between opening and closing the channel. The results of this can be seen in Table 1.

Table 1. Value and lifetime for channels.

	Mainnet	LN Mainnet	Testnet	LN testnet
Sample size (channels)	3809	715	8047	251
Average Channel value	0.00439556	0.0023375	0.07086336	0.07758942
Median channel value	0.001	0.0005	0.005	0.06
Standard Deviation Channel value	0.01384113	0.00586706	0.07067473	0.07191958
Average channel lifetime	1600	1940	2441	1269
Median channel lifetime	693	1836	433	312
Standard Deviation channel lifetime	2183	1791	5631	4059

In Table 2 we have shown the distribution for the number of inputs and outputs to channels. These results are fairly similar for both networks. We should, however, note the percentage of channels with two outputs in relation to the percentage with only one input. For the mainnet, the percentage of channels with one input is 70%, and the percentage of channels only with one output is 71%. As discussed in Section 4.1.4 we cannot easily match input-output pairs to users in the channel, so these numbers either mean that many channels do no off-chain transactions at all, meaning the single input will be outputted back to owner, or that all funds in the channel end up at one of the entities as a result of off-chain transactions. In the testnet column, we can see how the single output percentage is lower compared to the single input percentage. Note also, the double output percentage is higher than the double input. This indicates that many single funded channels end up with splitting the value when they close, and as we stated in Section 4.1.4 this enables us to say that off-chain transactions have taken place inside the channel. This difference is also present when we check how many channels had the same number of inputs as outputs. On the mainnet 57% of channels found have the same number of inputs as outputs, while on the testnet this is 46%. Using the HTLC transaction heuristic we found which were the outputs from closing transactions in our channel graphs. We can determine how often they occur: 2.6% of channels on the mainnet and 5.2% on the testnet would have one HTLC output. Compared to the Table 2 we can see how this can make up all the channels with two outputs, and the ones with three or more as channels certainly can have more HTLC outputs.

Table 2. Percentages of input-output count for channels.

	Mainnet	Testnet
Channels with one input	69.3	76.4
Channels with two inputs	23.2	16.8
Channels with three inputs	4.5	3.8
Channels with more than three inputs	2.8	2.9
Channels with one output	69.2	50.5
Channels with two outputs	29.6	46.3
Channels with three outputs	0.8	2.1
Channels with more than three outputs	0.2	0.8

5.5. Linking

The linking of channels was done, both on the set of channel graphs found on the blockchain during the LN collection intervals, and the sets of channels found when doing complete runs as described in Section 5.4. In Table 3 we can see the findings in regards to the key reuse when checking the channels graphs for testnet and mainnet found during the complete runs. Using the 3809 channels we found on the mainnet, we extracted a total of 26,824 unique keys from the transactions in the graphs, with 1370 or 5.1% of these being found multiple times. These keys combined where reused 2850 times, which if the key reuse was evenly distributed in the channel graphs, each channel graph would have 0.75 keys used in another channel graph. For the 8047 channels found on the testnet, we got 50,050 unique keys. The results here are very similar to the mainnet, with 6.5% of the keys found were reused, and 0.9 reused keys per channel graph if they were equally distributed.

Table 3. Key reuse in channel graphs found in complete runs.

	Mainnet	**Testnet**
Unique Keys found	26,824	50,050
Unique keys reused	1370	3250
Instances of key reuse	2850	7240
Maximum number of reuses for a single key	23	171
Average reuse of keys	2	2.2

Our two other heuristics were also used on these channel sets, allowing us to determine how many channels we were able to link. The results of this are shown in Table 4. For heuristic one, where we found directly connected graphs trough outputs-inputs, we have separate results for each possible output which can be the source of such a relationship. We discovered a clear difference between the different outputs: the outputs from the funding transactions make up over 65% of the connections found using heuristic one, both on the testnet and mainnet, compared to 2.1% or less for the outputs from the closing transactions. The outputs from the timelocked transactions make up 30% and 33% of the connections on the mainnet and testnet, respectively. One possible explanation for this difference has to do with the value of the outputs: closing timelocked outputs are used to give one party their value from the channel, meaning the value of these outputs is limited to the channel balance of the participant owning them; outputs from the funding transaction, on the other hand, are used to transfer change resulting from the output spent to fund the channel, which has no size limit, meaning an output many times the value of the channel can be used. Unless multiple inputs are used to fund a channel, creating a new channel using the outputs from another channel, limits the value of the funding input to the balance in the previous channel. A possible reason for the difference between closing and timelocked outputs could be the recommendation to spend timelocked outputs discussed in Section 4.3. This means if one is closing many channels unilaterally at the same time, consolidation of all timelocked outputs into one timelocked transaction is a natural thing to do; this would cause graph overlap as defined in heuristic three, but also create a high-value output from the timelocked transaction which could be used to fund new channels. Heuristic 3 which used graph overlap or multi-input timelocked transactions provided 12% on mainnet, and 24% on the testnet, of the total connections, found using both heuristic two and three.

Table 4. Relationships found between channel graphs using heuristic one and two.

	Mainnet	**Testnet**
Total relationships found	1586	5715
Founding output relationships	931	2824
Closing output relationships	30	2
Timelocked output relationships	435	1499
Graph overlap relationships	190	1390

If we combine our results of using all three heuristics on the channel sets, in the mainnet we get a total of 4436 relationships between the channel graphs. This enabled us to create 2182 links between channels. The reason for this difference is redundancy of connections and key reuse within a single channel graph. This means that 2254 of our relationships, or about 50% where redundant, but only 36 of them being the actual reuse of the same key in the same channel graph. The set of linked channels can be seen visualized in Figure 15, were we have created a channel network as discussed in Section 4.3. The network consists of 2324 smaller components, with each node having an average degree of 1.1. While this graph does not distinguish users, we can reveal additional information about the users participating in the channels. As edges represent at least one common user being present in both nodes, we can create the property: for adjacent nodes, there can be at most three different users and a minimum of two users. This means if the distance between vertices is more than one, we can no longer be sure if there is common user participation, but we cannot rule it out either. More importantly, it has an impact on complete components—i.e., every node is connected to every other node. Because the property we defined must hold for every node in such a complete component, there are only two possibilities with regard to user participation: there can be at most three total users in the component, or that one user participates in all nodes, but with a different second user in each. We can see a few examples of such components in Figure 15, and some components with a similar structure but with additional nodes which are not part of the complete core of the component.

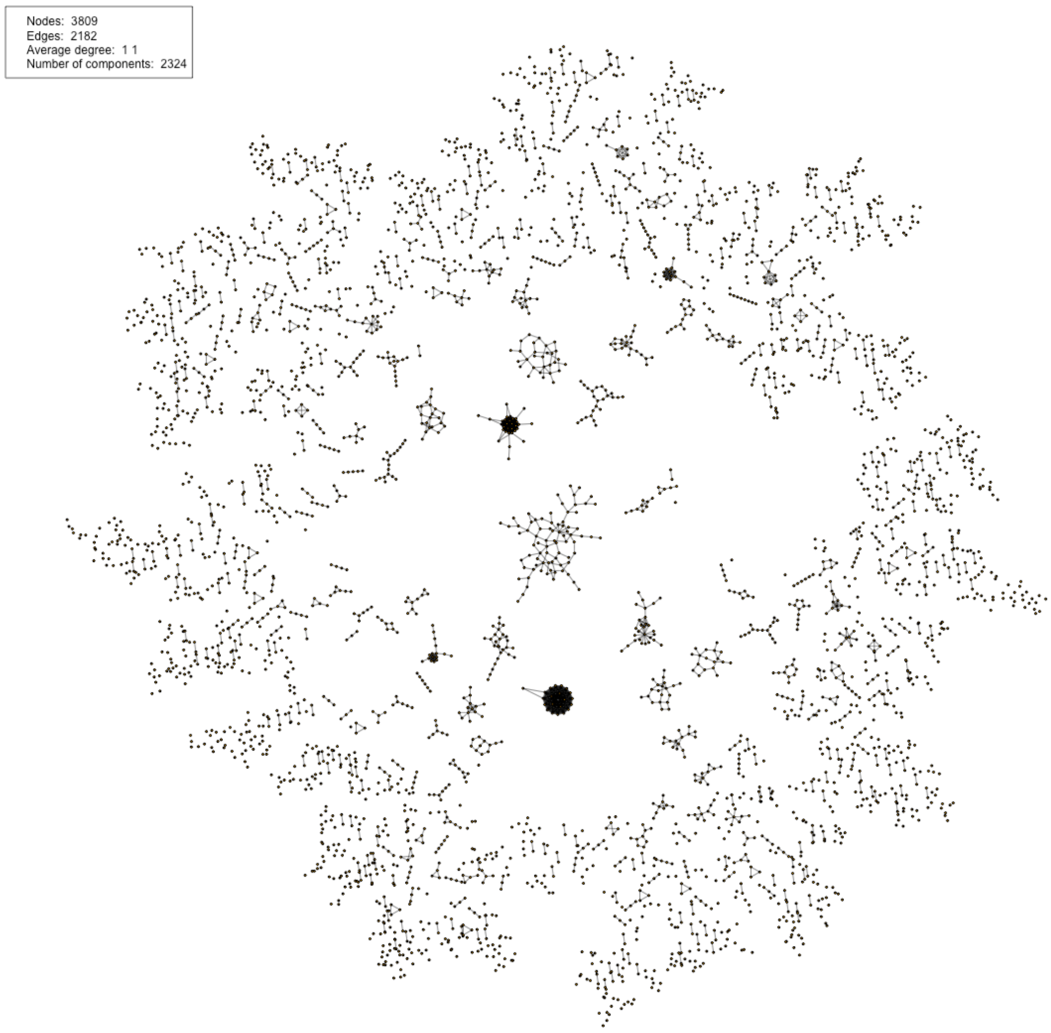


Figure 15. Channel network for mainnet.

We did the same for the set of channel graphs from the testnet. The total number of relationships using all three heuristics was 12,955, while the number of links we could create was 7084. Again, 45% of the connections we found where redundant for actually linking the channels. Here the number of reused keys within a single channel graph was 113, which is relatively small compared to the 5758 other redundant relationships. The channel network is shown in Figure 16 is more connected than the mainnet in Figure 15, with the testnet network having over twice as many nodes, but only having 617 more components. We can see this difference clearly with the large component in the middle of the testnet graph. As the LN is one network, it should consist of one dynamic component, meaning a historic network graph should also ideally have one large component. While we can see this beginning to emerge in our testnet channel network, it is very clear in the channel network constructed using LN data as seen in Figure 17.

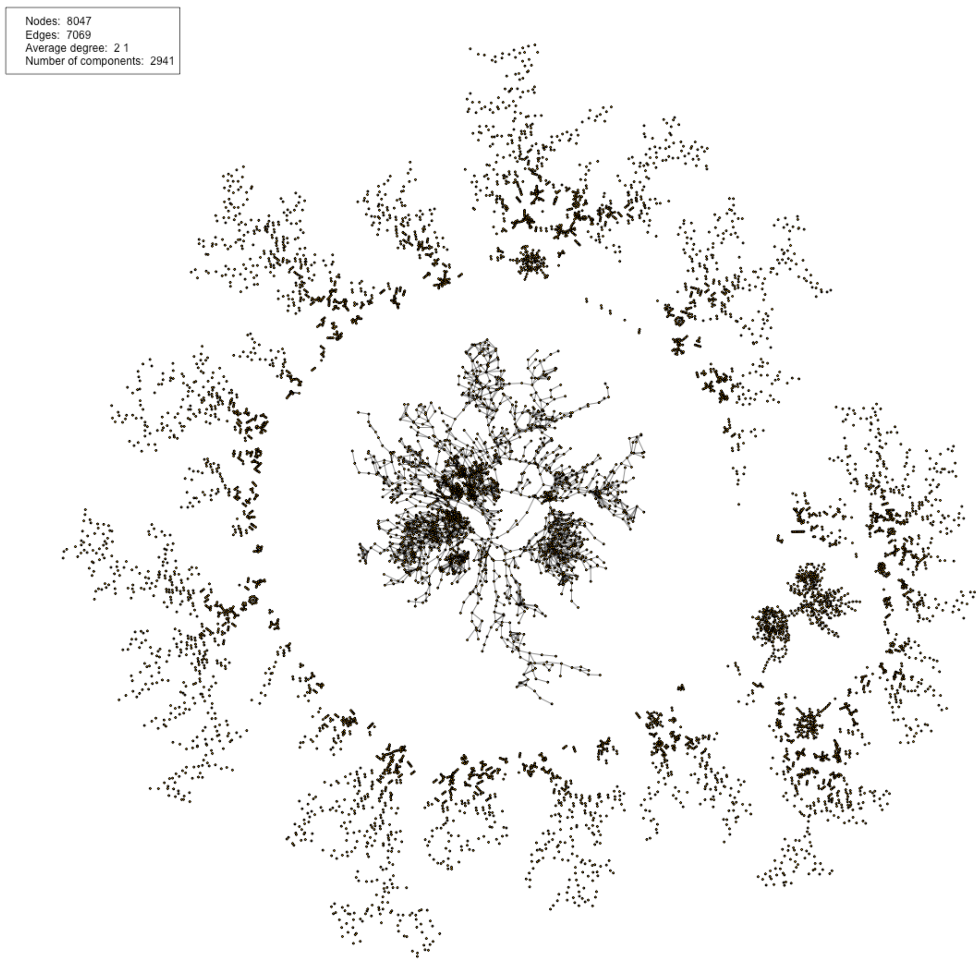


Figure 16. Channel network for testnet.

The channel network in Figure 17 based on LN data from interval 1, was created so we could compare it with the networks we created by linking blockchain information. This allowed us to see how connected a channel network should ideally be if all relationships were available to us in the blockchain. In the LN data, we collected during the intervals we used the node id, which identifies a node/user, to link the channels that nodes participated in. While users are easily separable by this id, we ignored this to create the same type of network as we did with the use of the blockchain data, meaning edges only indicate common user participation, not a specified user. As shown in Figure 17 the network is highly connected with the 715 nodes having an average degree of 58. We can see that almost all nodes are a part of the main component, with only 6 nodes not being connected to it. Figure 18 depicts the histogram for the node degrees in this network. In this histogram there is one node that sticks out, it has a degree of over 200, compared to the other highly connected nodes which have a degree of around 150. There is also a high frequency of nodes with a degree between 100 and 150. In Table 5 we have taken the top five nodes in terms of degree, and calculated some centrality measures for them. The highest connected node with a degree of 239, has also the highest score in the other measures. It can be seen in Figure 17 as the red node. For the other nodes, there is some variation in the scores, but all five nodes are in the top ten for all nodes in every measure, except node or channel 569069835159470080 colored blue in Figure 17. For closeness centrality, this node was ranked 31, while scoring high on the degree and betweenness centrality. Comparing this network to the channel network in Figure 19, which we cratered using the channels identified on the blockchain in the same interval, and linked using our three heuristics, we can see how limited our linking capabilities really are. With almost six times as many nodes as edges and an average degree of

0.1, we can see how most channels identified have no relation to others we could find, within such a short interval.

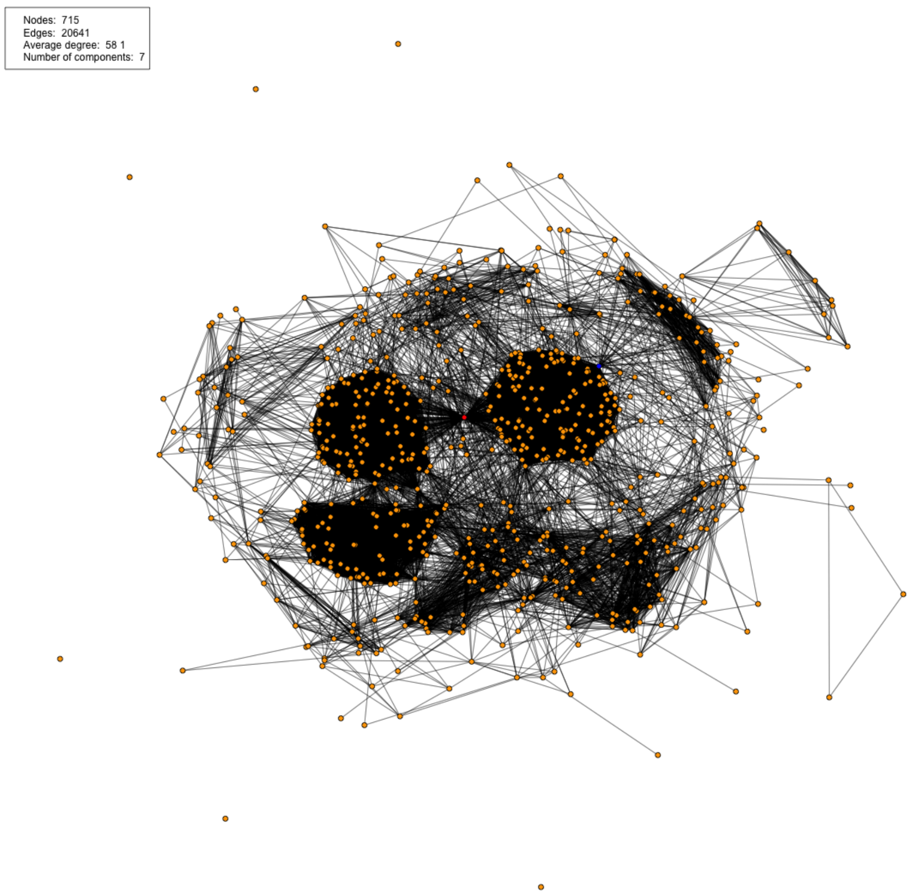


Figure 17. Channel network from the LN mainnet, first interval.

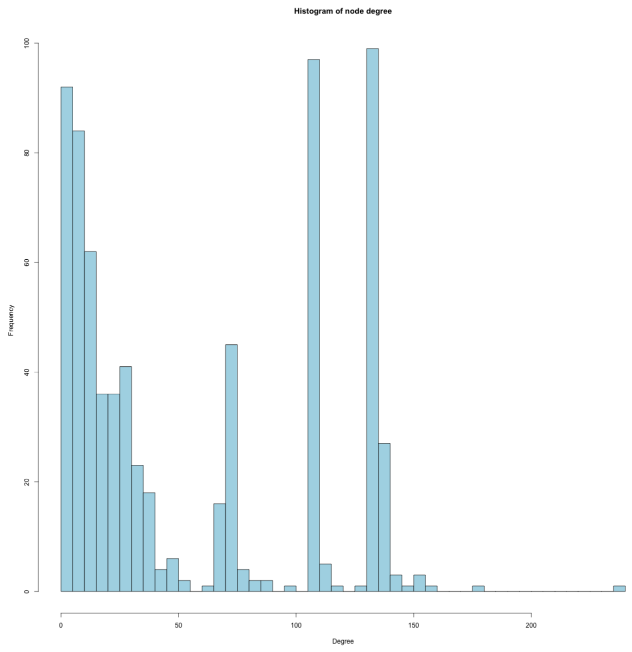


Figure 18. Histogram of node degree for the network in Figure 17.

Table 5. Centrality measures for the highest degree nodes for the network in Figure 17.

(short) Channel Id	Degree Centrality	Closeness Centrality	Betweenness Centrality
565863659278368768	239	0.0001790831	23,084.374
565758106242187265	176	0.0001743679	10,285.468
569069835159470080	159	0.0001737619	11,597.068
566469490153553920	153	0.0001763668	5733.209
567496434090377216	152	0.0001756543	6881.286

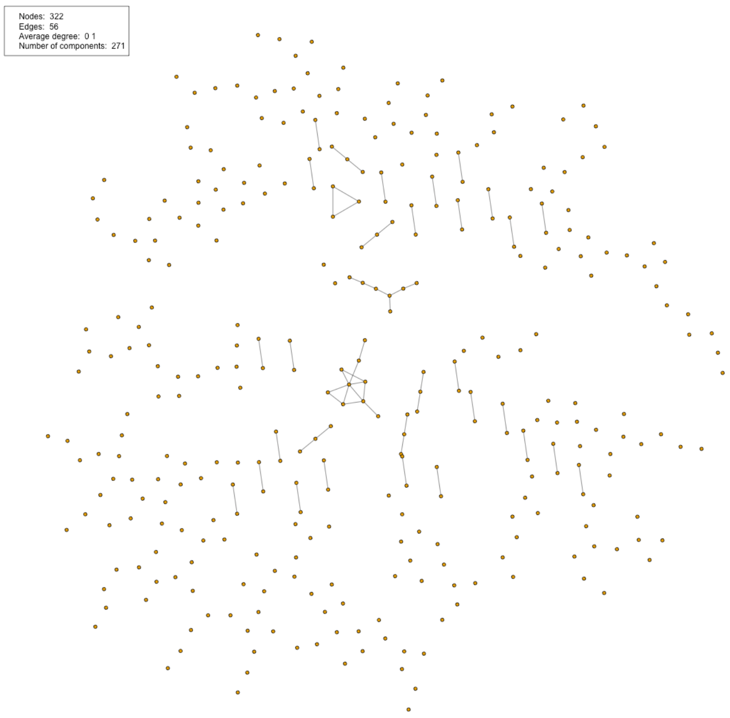


Figure 19. Channel network from the blockchain, mainnet, first interval.

6. Conclusions

We have conducted the analysis of Lightning network (LN) from the data stored in the underlying blockchain. We have observed, when attempting to locate on-chain LN transactions, that one can never be certain that a transaction is LN-related if one only uses the information available in the blockchain. The different characteristics used to identify potential on-chain transactions have a different level of uniqueness and will have an impact on the sensitivity of the method. Transaction uniqueness makes it likely that the transaction is LN related, but as stated previously: users are free to create whatever scripts and transaction chains they wish in the blockchain, and this naturally obfuscates the LN-related information. The identified channels also contain a limited amount of information allowing us to infer further LN properties, especially in terms of off-chain transactions. As the on-chain transactions relate to the channels in the LN, most of the information we can gather from the blockchain is related to the channels and users participating in the channels. This information allows us to determine some of the structure of the LN, in terms of which channels are connected. It is hard or impossible to determine anything regarding off-chain transaction flows. We can, therefore, conclude, from the blockchain analysis perspective, that doing transfers within the LN provides much better privacy than doing transfers on-chain in Bitcoin proper.

In our data collection interval comparing the channels found with our two methods of identification, and channels collected through the LN, we observed that more than 75% of all 2of2 multisig transactions are, in fact, LN channels. This was due to the possible limits of our LN node to collect all channels, as other nodes will remove inactive channels, and they will therefore not be propagated. This might become apparent if one were to use longer data collection intervals or run the collection node longer before starting to collect, allowing the node to receive many channels that will eventually be pruned by other nodes. The timelocked identification provides a good middle ground between precision and sensitivity, i.e., how many channels it will be able to find, with the results from the second interval indicating that around 25% of channels in LN are unilaterally closed. As shown in our correlation between `P2WSH` outputs and number of channels on the LN, the `P2WSH` outputs can be used as an indicator of the number of channels, at least providing an upper limit on active channels at any point in time. It is important to note, however, that these numbers are reliant on the current use cases of the things used for identification and correlation. For example, if 2of2 multisig is used commonly for other purposes than LN channels, identification based on it will become notably unreliable.

Using new keys for each transaction providing the users with pseudo-anonymity continues to be important for providing privacy in channels graphs. In addition to enabling us to link channels containing the same keys, it also enables linking of keys within the channel graph itself. We discussed previously how to distinguish users within a graph would enable us to determine which users provide a link between channels, allowing us to recreate parts of the structure of the LN more accurately. The best way of doing this is to link all the keys—for this, the entire transaction graph should also be used. Our linking results demonstrate that the information about LN available in the blockchain proper, and our heuristics are quite limited. When comparing the channel network created by using information from the LN itself, with the one created using channel graphs extracted from the blockchain in the same interval, we can see a real clear difference in the number of edges. The network using LN information has an average degree over a hundred times larger than the channel network reconstructed from the blockchain data. The effectiveness of our heuristics is dependent on user behaviour, and how LN implementations operate regarding transaction structure. For example, using a single transaction to spend many timelocked outputs is simple and convenient, but allows for channel linking.

It is apparent that the LN information available within LN itself is better to transaction information available in the blockchain proper. All LN channels need to be anchored on the blockchain. This provides verifiable historical data, and as such it is an important source of information with regards to the LN. Additionally, blockchain analysis can be used in conjunction with other methods of analysis on the LN, providing elegant data and information supplement.

Author Contributions: Conceptualization, methodology, M.N.; software, J.T.; validation, J.T.; formal analysis, M.N.; investigation, J.T.; writing—original draft preparation, J.T.; writing—review and editing, M.N.; visualization, J.T.; supervision, M.N.; project administration, M.N.; funding acquisition, M.N.

Funding: This research received no external funding, as it has been conducted as part of MSc project work in the University of Science and Technology, NTNU, Norway.

Acknowledgments: The authors would like to acknowledge the NTNU Digital Forensics research group, as well as Department of Computer Science, IDI, NTNU for making this research possible.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

LN	The Lightning Network
TX	Transaction
BTC	Bitcoin currency token

References

1. Nakamoto, S. *Bitcoin: A Peer-To-Peer Electronic Cash System*; CoinDesk: New York, NY, USA, 2008.
2. Poon, J.; Dryja, T. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. 2016. Available online: https://lightning.network/lightning-network-paper.pdf (accessed on 1 June 2018).
3. Trillo, M. Stress Test Prepares VisaNet for the Most Wonderful Time of the Year. 2013. Available online: https://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html (accessed on 28 May 2018).
4. Reid, F.; Harrigan, M. An analysis of anonymity in the bitcoin system. In *Security and Privacy in Social Networks*; Springer: New York, NY, USA, 2013; pp. 197–223.
5. Meiklejohn, S.; Pomarole, M.; Jordan, G.; Levchenko, K.; McCoy, D.; Voelker, G.M.; Savage, S. A fistful of bitcoins: characterizing payments among men with no names. In Proceedings of the 2013 Conference on Internet Measurement Conference, Barcelona, Spain, 23–25 October 2013; pp. 127–140.
6. Herrera-Joancomartí, J. Research and challenges on bitcoin anonymity. In *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*; Springer: Cham, Switzerland, 2015; pp. 3–16.
7. Malavolta, G.; Moreno-Sanchez, P.; Kate, A.; Maffei, M.; Ravi, S. *Concurrency and Privacy with Payment-Channel Networks*; ACM: New York, NY, USA, 2017.
8. Antonopoulos, A.M. *Mastering Bitcoin: Programming the Open Blockchain*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2017.
9. Andresen, G. Bitcoin Improvement Proposal 16. 2012. Available online: https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki (accessed on 28 March 2018).
10. Lombrozo, E.; Lau, J.; Wuille, P. Bitcoin Improvement Proposal 141. 2015. Available online: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki (accessed on 2 April 2018).
11. Lau, J.; Wuille, P. Bitcoin Improvement Proposal 143. 2016. Available online: https://github.com/bitcoin/bips/blob/master/bip-0143.mediawiki (accessed on 2 April 2018).
12. Wuille, P. Bitcoin Improvement Proposal 62. 2014. Available online: https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki (accessed on 1 June 2018).
13. Lightning Network Specifications. BOLT 3: Bitcoin Transaction and Script Formats. 2018. Available online: https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md (accessed on 24 April 2018).
14. Pfitzmann, A.; Köhntopp, M. Anonymity, unobservability, and pseudonymity—A proposal for terminology. In *Designing Privacy Enhancing Technologies*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 1–9.
15. Androulaki, E.; Karame, G.O.; Roeschlin, M.; Scherer, T.; Capkun, S. Evaluating user privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 34–51.
16. Ron, D.; Shamir, A. Quantitative analysis of the full bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 6–24.
17. Herrera-Joancomartí, J.; Pérez-Solà, C. Privacy in bitcoin transactions: New challenges from blockchain scalability solutions. In *Modeling Decisions for Artificial Intelligence*; Springer: Cham, Switzerland, 2016; pp. 26–44.
18. Russell, R. Routing on the Lightning Network? 2015. Available online: https://lists.linuxfoundation.org/pipermail/lightning-dev/2015-July/000019.html (accessed on 1 June 2018).
19. Roasbeef. BTCD—An Alternative Full Node Bitcoin Implementation Written in Go (Golang). 2018. Available online: https://github.com/roasbeef/btcd (accessed on 26 April 2018).
20. BtcDrak; Friedenbach, M.; Lombrozo, E. Bitcoin Improvement Proposal 112. 2015. Available online: https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki (accessed on 26 April 2018).
21. Lightning Network Specifications. Lightning Network In-Progress Specifications. 2018. Available online: https://github.com/lightningnetwork/lightning-rfc (accessed on 26 April 2018).
22. Lightningnetwork. Lightning Network Daemon. 2018. Available online: https://github.com/lightningnetwork/lnd (accessed on 22 May 2018).
23. Lightning Network Specifications. BOLT 7: P2P Node and Channel Discovery. 2018. Available online: https://github.com/lightningnetwork/lightning-rfc/blob/master/07-routing-gossip.md (accessed on 22 May 2018).

24. Lightning Network Specifications. BOLT 5: Recommendations for On-chain Transaction Handling. 2018. Available online: https://github.com/lightningnetwork/lightning-rfc/blob/master/05-onchain.md (accessed on 24 April 2018).
25. Hoenicke, J. Johoe's Bitcoin Mempool Statistics. 2018. Available online: https://jochen-hoenicke.de/queue/#1,all (accessed on 11 May 2018).