Cerberus Channels: Incentivizing Watchtowers for Bitcoin

Georgia Avarikioti[1], Orfeas Stefanos Thyfronitis Litos[2], and Roger Wattenhofer[1]

[1] ETH Zürich
{zetavar,wattenhofer}@ethz.ch
[2] University of Edinburgh
o.thyfronitis@ed.ac.uk

Abstract. Bitcoin and similar blockchain systems have a limited transaction throughput because each transaction must be processed by all parties, on-chain. Payment channels relieve the blockchain by allowing parties to execute transactions off-chain while maintaining the on-chain security guarantees, *i.e.*, no party can be cheated out of their funds. However, to maintain these guarantees all parties must follow blockchain updates ardently. To alleviate this issue, a channel party can hire a "watchtower" to periodically check the blockchain for fraud on its behalf.
However, watchtowers will only do their job properly if there are financial incentives, fees and punishments. There are known solutions, but these need complex smart contracts, and as such are not applicable to Bitcoin's simple script language. This raises the natural question of whether incentivized watchtowers are at all possible in a system like Bitcoin.
In this work, we answer this question affirmatively, by introducing Cerberus channels, an extension of Lightning channels. Cerberus channels reward watchtowers while remaining secure against bribing and collusion; thus participants can safely go offline for an extended period of time. We show that Cerberus channels are correct, and provide a proof-of-concept implementation in the Bitcoin script language.

Keywords: Bitcoin · Security · Payment channels · Payment network · Lightning network · Watchtowers · Collateral · Incentives

1 Introduction

1.1 Motivation

Since its inception, Bitcoin [16] is the leading cryptocurrency in terms of market capitalization. Unfortunately, Bitcoin suffers from limited transaction throughput due to its underlying consensus mechanism. Specifically, Bitcoin handles at most seven transactions per second [5], while current digital monetary systems, such as Visa, handle tens of thousands. This is a major obstacle on the wide adoption of Bitcoin.

Payment channels are the foremost solution for scaling decentralized blockchain systems such as Bitcoin. Payment channels allow transactions between two parties to be executed off-chain, while maintaining the security guarantees of the blockchain. Specifically, if two parties want to execute multiple transactions, they open a channel with a single on-chain transaction and then execute transactions privately and off-chain on this channel. The blockchain is only used to close the channel or in case of dispute.

Although payment channels offer a simple and efficient solution to the limited transaction throughput of blockchain systems, they have a major drawback. The correct operation of a payment channel depends on all parties of the channel being active and in sync with the blockchain. Otherwise, a party of the channel can close the channel in a wrong state, *i.e.*, a party can publish an outdated version of the distribution of the channel's funds. This outdated state will be considered final, unless the counterparty disputes it within a specific time period. This dispute time is specified when the channel is initiated on-chain; after t blocks, a fraudulent transaction cannot be disputed anymore. Hence, to maintain the security of the payment channel, both parties must be online at least once every t blocks.

A natural solution to relieve the channel parties from this necessity is outsourcing the dispute process to third-parties, known as watchtowers [1, 8]. Watchtowers on Bitcoin Lightning network [17] mainly focus on maintaining privacy; however, the current design does not provide incentives for participation for the watchtowers. In particular, the party that hires the watchtower pays it only when fraud happens. However, the watchtower knows that rational parties never commit fraud, thus there is little incentive for it to become a watchtower in the first place. Additionally, a rational watchtower can benefit from unintentional broadcasting of revoked updates and thus may lobby for buggy or misleading channel software. A naive alternative would be for the hiring party to pay the watchtower a small fee regularly, *e.g.*, every time a transaction is executed in the channel. In this case, however, a rational watchtower would avoid the cost of storing the hiring party's data and monitoring the blockchain and would thus fail to act upon fraud.

In this paper, we introduce Cerberus channels, where watchtowers are (i) incentivized to participate in the system, and (ii) penalized in case they do not act upon fraud. In particular, each party has the option to employ a watchtower as a service provider. The watchtower is paid for every transaction executed on the channel and locks collateral on-chain as guarantee for its honest behavior. In case the watchtower misbehaves and does not dispute an outdated state, the cheated party can claim the watchtower's collateral. Hence, rational watchtowers are incentivized both to participate and act upon fraud. In our construction, the parties can go offline for an extended period of time and need only be online to penalize the watchtower. This way we weaken the availability requirement for the parties of a payment channel. More importantly, Cerberus channels build upon and extend Lightning channels and only require timelocks and additional transactions.We also provide a proof-of-concept implementation (`https://github.com/OrfeasLitos/cerberus-script`).

1.2 Related Work

Payment channels were originally introduced by Spilman [18] as unidirectional channels and were later established as bidirectional channels [7, 17]. Currently, there exist many different constructions of bidirectional payment channels, some applicable only on platforms that allow for arbitrary smart contracts such as Ethereum [3, 9, 11, 15], and some applicable also on blockchain systems with limited scripting languages like Bitcoin [4, 6, 7, 17]. This work falls in the second category.

The most famous and active payment network is the Bitcoin Lightning network [17] currently operating more than 35,900 channels by over 9,900 Bitcoin nodes that contain more than 830฿ in total [2]. However, Lighting as well as most of the other payment networks require channel parties that are frequently online, watching the blockchain. To address this issue, Dryja introduced Monitors [8], also known as Watchtowers [1], a third-party solution that acts as a proxy for a channel's party effectively allowing the party to go offline for a long period of time while maintaining the security of the channel operation (the other party cannot cheat). Watchtowers mainly focused on privacy preserving techniques to ensure the hired third-party does not learn any information about the off-chain transactions. Thereafter, Avarikioti et al. proposed DCWC [4], a distributed protocol for watchtower services, in an attempt to involve all full nodes to participate in the system and consequently enhance security. However, both these works, Watchtowers and DCWC, fail to provide the necessary incentives for participation in the system. In particular, the watchtowers are paid upon fraud. Thus, the watchtowers will not participate in this system because no rational party will commit fraud, hence they will never be paid. Therefore, there will be no watchtowers. On the contrary, Cerberus channels provide the necessary incentives mechanisms for watchtowers (rewards and punishment).

Towards the same direction, McCorry et al. presented Pisa [14], a protocol that outsources the dispute handling of (state[3]) channels to hired third-parties. However, Pisa has two shortcomings: First, the main protocol implementation is not secure against bribing since the watchtower's collateral is not linked to the party or the channel on-chain, hence the watchtower can double-spend it. Second, Pisa is not compatible with Bitcoin, because it requires a smart contract beyond the limitations of script. On the contrary, Cerberus channels are applicable on Bitcoin and furthermore they do not suffer from the security problems of Pisa since the collateral of the watchtower is linked to the hiring party with an on-chain transaction.

[3] State channels generalize payment channels to support smart contracts [15].

More recently, Avarikioti et al. introduced Brick [3], an asynchronous off-chain construction that employs a committee of watchtowers. Although Brick manages to remove the synchrony requirements on the network layer and the perfect blockchain assumption while maintaining the security of channels, it is not compatible with Bitcoin-like platforms, as opposed to Cerberus channels.

In a different line of work, Khabbazian et al. [13] proposed a lightweight watchtower design in which watchtowers do not need to store the singed justice (revocation) transactions, but instead can extract them directly from the commitment transactions that appear on the blockchain. This work is independent and complementary to ours, and can be applied also to Cerberus channels to improve the storage requirements for the watchtower service.

1.3 Our contribution

To summarize, the contribution of this paper is the following:

- We introduce Cerberus payment channels that enable participants on Bitcoin to employ watchtowers and thus go securely offline for an extended period of time.
- We define the desired properties for payment channel solutions and prove Cerberus channels are secure under our security model. Specifically, we show watchtowers are incentivized to both participate and act upon fraud. Thus, Cerberus channels are secure against collusion and bribing.
- We provide a proof-of-concept implementation of Cerberus channels on Bitcoin.

2 Background and Notation

2.1 Payment Channels

For the rest of the paper, when we refer to payment channels we imply Lightning channels, currently operating on the Bitcoin network. Next, we provide a brief overview of Lightning channels, on which Cerberus channels build upon.

To open a payment channel, the parties publish a funding transaction where they lock their funds into a common account, *i.e.*, both parties must sign to spend the output of the funding transaction. Every time the parties execute a transaction, they update the current state of the channel accordingly, meaning they distribute the funding transaction's output as agreed and sign the resulting "commitment" transaction. In addition, each party reveals to the counterparty a secret that allows the counterparty to sign a "revocation" transaction that spends the previous commitment transaction; the revocation transaction awards the cheating party's funds to the cheated party, effectively punishing the party that tried to cheat. The output of the party that published the commitment transaction is locked for a time period, known as the revocation period. The cheated party must publish the revocation transaction during the revocation period, otherwise the cheating party will be able to spend the balance of the revoked state. Thus the security of the channel construction crucially depends on all parties of the channel watching the blockchain and being online at least once during the revocation period.

2.2 Contracts

A contract is an agreement that can be enforced on the blockchain. Enforcing such a contract depends on the operations the programming language allows with respect to transaction outputs. Most recent cryptocurrencies, such as Ethereum, support a Turing-complete language and thus can enforce arbitrary rules with a *smart contract*. However, Bitcoin (as well as other cryptocurrencies) has strict limitations on the scripting language and allows only specific operations. As a result Bitcoin's contracts are simpler and with limited functionality. Next, we discuss the operations allowed in script with respect to transaction outputs, on which Cerberus channels build upon.

Signatures. A signature is the most basic form of contract and is essentially a proof of ownership of a transaction output. We denote by σ_A the signature that corresponds to the public key A. (We omit the signed message, as it always is the transaction which contains the signature). Further, an m-of-n *multisignature* is a contract that demands at least m signatures which correspond to any m of the n predefined public keys. If m valid signatures are provided then the output is immediately spendable. In this work, we will only use 2-of-2 multisignatures, so we introduce the following notation: $\sigma_{A,B}$ expresses that the output of the transaction can only be spent with both the signatures of A and B.

Timelocks. Timelocks are another type of contract. When a transaction or a transaction output is timelocked it cannot be included in the blockchain until the specified time has elapsed. There are two types of timelocks, *absolute* and *relative*. Transactions or transaction outputs with an absolute timelock become valid when the specified timestamp or block height is reached. On the other hand, a relative timelock allows a transaction output to be locked for a time relative to the block that included that output. Relative timelocks are used in the Lightning protocol as well as in our protocols. We denote by Δt a relative timelock that expires t blocks (confirmations) after the transaction is included in the blockchain. After this time the output of the transaction is spendable.

2.3 UTXO Notation

In this section we introduce the necessary notation for our protocol.

We assume the blockchain is UTXO-based (UTXO: Unspent Transaction Output), meaning that transactions consist of inputs and outputs. A transaction connects its inputs to UTXOs (removing the latter from the UTXO set) and creates new UTXOs, its outputs. Each UTXO can only be spent as a whole. A UTXO consists of a monetary value and the conditions under which it can be spent, *e.g.*, a signature corresponding to a public key, a timelock etc.

We denote by $o = (x \mid C)$ the UTXO that holds a monetary value of x coins that can be spent when conditions C are met. For example, $o = (10 \mid \sigma_A)$ means that the signature that matches the public key A can spend the output o which is equivalent to 10 coins.

A transaction in this model is a function mapping a set of UTXOs, called inputs, to a (new) set of UTXOs, called outputs. Thus, we define a transaction as follows:

$$TX_i = [o_j, o_k, \dots] \mapsto [o_i^1, o_i^2, \dots]$$

where o_j, o_k, etc. are the inputs of the transaction and $o_i^1, o_i^2, \dots$ are the first, second, etc. outputs, respectively. If transaction TX_i has a single output we simply write o_i. Moreover, if the specific UTXO that is input to a transaction is irrelevant to the protocol design, we refer to it as $\#$. If we demand the input to belong to a specific public key A and hold a specific value of x coins, but which of the UTXOs owned by A is spent by the input is irrelevant to the protocol design, we refer to it as $(x \mid \#_{\sigma_A})$. For instance,

$$TX_i = \left[(0.8 \mid \#_{\sigma_A}), o_k^2\right] \mapsto \left[(1 \mid h(s)), (0.5 \mid \sigma_B)\right]$$

denotes the transaction TX_i that spends a UTXO from the party A with value 0.8 coins and the second output of transaction TX_k and creates two new UTXOs. The first output holds the value of 1 coin and can be spent with the secret s, while the second holds 0.5 coins and can be spent with B's signature.

3 Protocol Overview

3.1 System Model

Cryptographic Assumptions. We make the typical cryptographic assumptions, *i.e.*, there are secure communication channels between participants, and cryptographically secure hash functions, signatures, and encryption schemes. Additionally, all parties of the protocol (watchtowers, channel parties, external adversaries) are computationally bounded.

Blockchain Assumptions. We assume a perfect blockchain, in the sense that both persistence and liveness hold [10]. In particular, we assume that if a valid transaction is propagated in the blockchain network it cannot be censored and will be included in the "permanent" part of the blockchain immediately[4].

Network Model. We assume that any participant of a channel can go offline (intentionally or due to a Denial-of-Service (DoS) attack) for a (long) period of time up to T. Furthermore, we consider watchtowers that are resilient against DoS attacks. We argue this assumption is realistic since watchtowers are also required to lock high collateral to participate in the system and thus operators will invest in anti-DoS protection.

Threat Model. We assume that the watchtowers as well as the channel participants are rational players. Thus, they will only deviate from the honest protocol execution if they can gain more profit. Moreover, channel participants can collude with the watchtower(s).

3.2 Cerberus Overview

Cerberus channels aim to alleviate the need for the channel parties to be frequently online watching the blockchain. We propose simple modifications on the Lightning protocol that allow the parties to employ watchtowers while incentives for active participation and thus security of the channels are guaranteed.

In particular, in Cerberus channels, the watchtower is rewarded for every update on the channel but also locks collateral as guarantee in case the watchtower does not act upon fraud. The party employing the watchtower can claim the collateral within the penalty period, if the watchtower misbehaves. The penalty period is however much larger than the revocation period, hence the party that employs the watchtower can go offline for an extended period of time. On the other hand, on a normal operation of the channel the watchtower can reclaim the collateral when its service is terminated. The watchtower has the option to terminate its service to the party at any point during the protocol execution. In such a case, the party can either update the channel and employ a new watchtower, close the channel, or be online more frequently to avoid fraud.

3.3 Payment Channel Properties

We define the desired properties of a payment channel construction below, summarizing the work of [3, 12, 14, 15].

1. **Security:** Any party of the channel can enforce the last agreed state (balance) on-chain at any time.
2. **Privacy:** No third-party, external to the payment channel, can gain information about the state (distribution of funds) of the channel.
3. **Scale-out:** The number of transactions on-chain is constant.

The first two, namely Safety and Privacy, are security properties, while the third is the performance property that is required in a channel to achieve its main purpose – higher transaction throughput.

We note that these properties are also met by the Lightning channel construction, but under a different network model. Specifically, Lighting channels require participants to be frequently online watching the blockchain to guarantee security. In this work, we aim to alleviate this requirement, thus providing security as specified in Section 3.1.

[4] Note that Cerberus channels can be made secure for any confirmation time k, but we choose $k = 1$ to simplify the protocol and security analysis.

4 Cerberus Design

In this section, we describe in detail the architecture of CERBERUS payment channels. We base our design on Lightning channels, and introduce the necessary modifications and extensions to guarantee the desired properties under the predefined model for our design and the applicability to Bitcoin.

First, we divide the channel lifetime in four phases: *Open, Update, Abort* and *Close*. Then, we describe the necessary transactions and present the protocol design for each phase. To simplify the description, we assume, wlog, that party B has hired watchtower W and the potentially cheating party is A.

We note that CERBERUS can accommodate the usecase where only one party wishes to hire a watchtower, as well as that in which both parties choose to do so. Different watchtowers can be used by the two parties. In the case neither party employs a watchtower, the protocol reverts to Lightning.

4.1 Phase: Open

Similarly to the Lightning protocol, phase *Open* includes a funding transaction and a commitment transaction. The funding transaction creates a common account between the parties and is eventually published on-chain to notarize the committed funds, hence the opening of the channel. The funding transaction of a CERBERUS channel spends the funds of the channel parties and creates a new 2-of-2 multisig output spendable only if the parties collaborate.

The commitment transaction (first of many to follow) distributes the funds between the parties and is signed and held in private by both parties. The commitment transaction, if published, does not allow the publishing party to spend its funds immediately, to ensure there is enough time for punishment in case of fraud (*i.e.*, the commitment transaction is not the last agreed state by the channel parties). This is known as the revocation period, denoted by t. The first commitment transaction should be signed by both parties before signing and publishing the funding transaction to avoid a hostage situation.

Furthermore, we introduce two new transactions in phase *Open*, the *collateral transaction* and *reclaim transaction*, that involve the watchtower service. The collateral transaction is funded by the watchtower, while its output is a joint account between the watchtower and the hiring party. The value of the collateral is slightly higher than the channel funds. The reclaim transaction, on the other hand, allows the watchtower to reclaim the collateral, effectively terminating its service. The output of the reclaim transaction can be spent as follows: either a long "penalty" period T has elapsed since the watchtower singed and published on-chain the reclaim transaction, or both the signatures of the watchtower and the party are present. Intuitively, the penalty period T allows the cheated party to penalize an inactive/malicious watchtower, during phase *Close*. Further, timelock T allows the party employing the watchtower to be notified either in case of fraud or in case the watchtower simply wants to withdraw its service. In the latter case, the party can either be online more frequently (revocation period), close the channel, or collaboratively update the channel to cancel out the previous commitment transactions and employ a new watchtower. Note that the reclaim transaction is signed by both the watchtower and the hiring party before the collateral transaction is put on-chain to avoid a hostage situation.

Further, $T >> t$, and security holds as long as any participant in a CERBERUS channel employing a watchtower is offline for at most T time.

Next, we present in detail the transactions involved in the first phase, *Open*.

- **Funding transaction:** Opens a channel between two parties, A and B. The inputs are the parties' funds and the output is a 2-of-2 multisig of A and B.
- **Commitment transaction:** Updates the state of the channel, *i.e.*, the distribution of the funds between the parties. The input of the commitment transaction spends the output of the funding transaction. The commitment transaction has two outputs, one for each channel party, and distributes the funds of the channel to the two parties as agreed. Each party has its own version of the commitment transaction, signed by the counterparty. Each version has two outputs, one awarded to the party holding the commitment transaction, wlog party A, and one awarded to the counterparty B. Both outputs are timelocked for the revocation period t. Further, each output can be spent before t time elapses in collaboration with the watchtower W, *i.e.*, if both the watchtower and the corresponding party sign a transaction.

– **Collateral transaction:** Commits the watchtower's collateral on-chain. Note that the value of the collateral should be slightly higher than the total funds of the channel. The input of the collateral transaction is funded by the watchtower, while the output is a 2-of-2 multisig of B and W.
– **Reclaim transaction:** Allows the watchtower to reclaim the collateral. The input of the reclaim transaction spends the output of the collateral transaction. The output, on the other hand, requires the signature of the watchtower relatively timelocked by T or both the signatures of W and B.

Protocol: Open
Preconditions: A, B and W own on-chain a, b and c coins respectively. It holds that $c > a + b$ and W is employed by B.
A and B prepare the necessary transactions.

1. A and B create the funding transaction: $TX_f = \left[(a \mid \#_{\sigma_A}), (b \mid \#_{\sigma_B})\right] \mapsto (a + b \mid \sigma_{A,B})$
2. A and B create the first commitment transaction (version held by A):
   $TX_{C1A} = o_f \mapsto \left[(a \mid (\sigma_A \wedge \Delta t) \vee \sigma_{A,W}), (b \mid (\sigma_B \wedge \Delta t) \vee \sigma_{B,W})\right]$

A and B open the channel.

3. B sends the signature of TX_{C1A} to A. Symmetrically, A sends the signature of TX_{C1A} to B.
4. Both A and B sign TX_f and publish it on-chain, as in Lightning.

W locks its collateral for the channel on-chain.

5. W creates the collateral and reclaim transactions and sends both to B:
   $TX_{Coll} = (c|\#_{\sigma_W}) \mapsto (c \mid \sigma_{W,B})$
   $TX_{RC} = o_{Coll} \mapsto (c \mid (\sigma_W \wedge \Delta T) \vee (\sigma_{B,W}))$
6. B verifies, signs and sends to W the signature for the input of TX_{RC}.
7. W signs and publishes on-chain the collateral transaction TX_{Coll}.

Postconditions: Opening of the channel between A and B with total value $a + b$ coins, and employment of W from B with c coins as collateral.

4.2 Phase: Update

The second phase, *Update*, materializes the main functionality of a channel. In this phase, the parties update the current state of the channel (distribution of funds) and consequently transactions are executed off-chain. The *Update* phase in the Lightning protocol consists of a (new) commitment transaction and a revocation transaction. The commitment transaction represents the last, agreed by both parties, state of the channel. On the other hand, the revocation transaction allows a party to claim all the funds of the channel in case the other party publishes the previous commitment transaction (attempts to cheat).

The *Update* phase in Cerberus produces the following transactions: a commitment transaction, a revocation transaction, and two penalty transactions. The revocation transaction spends both outputs of the previous valid commitment transaction and awards them to the cheated party B. The revocation transaction is signed by the potentially cheating party A and is sent to B. Then, both party B and watchtower W sign, exchange and store the revocation transaction. Note that B will not sign the new commitment transaction unless both A and W sign the revocation transaction. Further, A acts as in Lightning.

The penalty transactions allows party B to penalize watchtower W during the penalty period in case fraud occurred and W did not publish the revocation transaction in time. Specifically, the penalty transactions both depend on the previous commitment transaction and on the collateral and reclaim transactions, respectively. Hence, the penalty transaction is valid only if fraud occurs and is not revoked. Thus, the revocation transaction has a double functionality; it awards the money of the cheating party A to the cheated party B and additionally acts as an insurance transaction for the watchtower, since it invalidates the penalty transactions by spending the outputs of the commitment transaction. Note that B will only sign the new commitment transaction after receiving the singed penalty transactions from W.

Further, we assume that a watchtower is paid regularly on every channel update by the hiring party via a one-way payment channel [5].

To sum up, during the *Update* phase the following transactions are created:

- **Revocation transaction:** In case of fraud, *i.e.*, if party A publishes a revoked commitment transaction, the revocation transaction returns all channel funds to the cheated party B. The inputs of the revocation transaction spend the outputs of the commitment transaction. The output of the revocation transaction is awarded to the cheated party B.
- **Penalty transaction 1:** Allows a party to claim the watchtower's collateral during the penalty period T, in case fraud occurred and the watchtower did not act during the revocation period t. The inputs of penalty transaction 1 are (a) the output that reflects B's channel balance on the corresponding commitment transaction, and (b) the output of the collateral transaction. The output of penalty transaction 1 awards all funds to B.
- **Penalty transaction 2:** Allows a party to claim the watchtower's collateral during the penalty period T, in case fraud occurred and the watchtower did not revoke it during time t, but tried to reclaim the collateral. The inputs of penalty transaction 2 are (a) the output that reflects B's channel balance on the corresponding commitment transaction, and (b) the output of the reclaim transaction. The output awards all funds to B.

All described transactions and their dependencies are illustrated in Figure 1.

Protocol: Update
Preconditions: A and B own a coins and b coins respectively in a channel. W has a locked collateral of c coins. Note that $a' + b' = a + b$.

1. B creates the next commitment transaction:
   $TX_{C,i+1,A} = o_f \mapsto \big[(a' \mid (\sigma_A \wedge \Delta t) \vee \sigma_{A,W}), (b' \mid (\sigma_B \wedge \Delta t) \vee \sigma_{B,W})\big]$.
2. B creates and sends to A the revocation transaction for the previous commitment transaction:
   $TX_{RiA} = \big[o^1_{CiA}, o^2_{CiA}\big] \mapsto (a + b \mid \sigma_B)$.
3. A sends to B its signature for the revocation transaction TX_{RiA}.
4. B sends to W both parties' signatures for the revocation transaction TX_{RiA}, along with the commitment transaction TX_{CiA}.
5. W sends to B its signature for the revocation transaction TX_{RiA}.
6. W creates penalty transactions 1 and 2:
   $TX_{P1iA} = \big[o_{Coll}, o^2_{CiA}\big] \mapsto (b + c \mid \sigma_B)$, $TX_{P2iA} = \big[o_{RC}, o^2_{CiA}\big] \mapsto (b + c \mid \sigma_B)$
7. W sends to B its signatures for o_{Coll} and o_{RC}.
8. B sends to A its signature for $TX_{C,i+1,A}$.

Postconditions: A, B own a', b' coins, resp. W has c coins locked as collateral.

4.3 Phase: Close

The last phase, *Close*, handles the closing of a Cerberus channel. Similarly to the Lightning protocol, in this phase either the parties close the channel in collaboration or a commitment transaction is published on-chain unilaterally by one of the channel parties. As soon as the commitment transaction is included on-chain the revocation period t begins, allowing the counterparty or the watchtower to supervene to a potential dispute resolution.

[5] Ideally, this payment should be integrated with Cerberus for efficiency.

Collaborative closure. The normal closure of the channel (no cheating occurs) is described in Protocol Close (a). Note that in this case, both parties sign the agreed distribution of the channel's funds and the funds are immediately awarded to the parties as soon as the transaction is included in the blockchain, *i.e.*, no timelocks are required.

Non-collaborative non-cheating closure. This happens if one party wants to close the channel and the other is unresponsive. Then, the responsive party publishes on-chain the last commitment transaction (already signed by both parties on last update). After time t, the funds of the channel are distributed to the parties according to the published state. As soon as the last commitment is published on-chain, the watchtower can safely put on-chain the reclaim transaction, since no penalty transaction corresponding to the last commitment transaction exists. Consequently, the watchtower can spend the collateral after time T, or immediately if the hiring party agrees to collaborate and sign a transaction that spends the collateral and awards the funds to the watchtower.

Protocol: Close (a) (Non-cheating channel closure)
Preconditions: A owns a coins, B owns b coins, W has a locked collateral of c coins.

1. A and B sign and broadcast $o_f \mapsto \big[(a \mid \sigma_A), (b \mid \sigma_B)\big]$.
2. W publishes on-chain the reclaim transaction TX_{RC} (can spend it after T).

Postconditions: The channel is closed, A owns a' coins, B owns b' coins and the collateral is returned to the watchtower W.

Cheating closure & responsive watchtower. In case A cheats and publishes an old commitment transaction, the watchtower publishes the corresponding revocation transaction during the dispute period, awarding all the funds of the channel to the cheated party B. Then, the watchtower can publish the reclaim transaction and spend its output safely since the corresponding penalty transaction has been invalidated. As described in Protocol Close (b), publishing the revocation and reclaim transaction can be done simultaneously, since the timelock t on the hiring party's output of the published commitment transaction guarantees that the hiring party cannot claim the watchtower's collateral during the revocation period. On the other hand, in case B cheats, A acts exactly as in Lightning.

Protocol: Close (b) (Cheating party & responsive watchtower)
Preconditions: A owns a coins, B owns b coins, W has a locked collateral of c coins. The last commitment transaction is denoted TX_{CnA}.

1. Party A publishes on-chain an old commitment transaction $TX_{CiA},\ i < n$.
2. During the revocation period t, W publishes the corresponding revocation transaction TX_{RiA}.
3. W publishes on-chain the reclaim transaction TX_{RC} (can spend it after T).

Postconditions: The channel is closed, the channel funds are awarded to the cheated party B, and the collateral returned to the watchtower W.

Cheating closure & unresponsive watchtower. If the watchtower does not publish the revocation transaction in time when fraud occurs, the cheated party can publish a penalty transaction and claim the watchtower's collateral. Specifically, if the reclaim transaction is not published, the cheated party publishes penalty transaction 1. Otherwise, if the watchtower has published the reclaim transaction but not the revocation transaction, the cheated party publishes penalty transaction 2. Both penalty transactions are valid, as long as the cheated

Protocol: Close (c) (Cheating party & unresponsive/malicious watchtower)
Preconditions: A owns a coins, B owns b coins, W has a locked collateral of c coins. The last commitment transaction is denoted TX_{CnA}.

1. Party A publishes on-chain an old commitment transaction TX_{CiA}, $i < n$ waits for the timelock to expire and spends the $(a \mid \sigma_A \wedge \Delta t)$ output.
2. B checks the chain periodically every time T and notices TX_{CiA} is on-chain and spent. If W has published the reclaim transaction TX_{RC}, B publishes Penalty transaction 2 TX_{P2iA}. Else, B publishes Penalty transaction 1 TX_{P1iA}.

Postconditions: The channel is closed, the channel funds are awarded to the parties according to the published commitment transaction TX_{CiA}, and the collateral (c coins) is awarded to party B.

party does not spend its output of the commitment transaction. Further, penalty transaction 2 is valid only if less than time T has elapsed since the reclaim transaction was put on-chain. Hence, the cheated party must go online within at most T time to claim the watchtower's collateral. Close (c) describes this case.

4.4 Phase: Abort

In this phase, the watchtower W withdraws the collateral and thus terminates its employment by party B. *Abort* is an intermediate phase, that can occur at any time between phases *Open* and *Close*[6]. To that end, the watchtower publishes on-chain the reclaim transaction. Consequently, timelock T comes in effect, locking the watchtower's collateral for the penalty period. During this period, an honest hiring party will check the blockchain once. The party can then close the channel, hire a new watchtower or monitor the blockchain every t time.

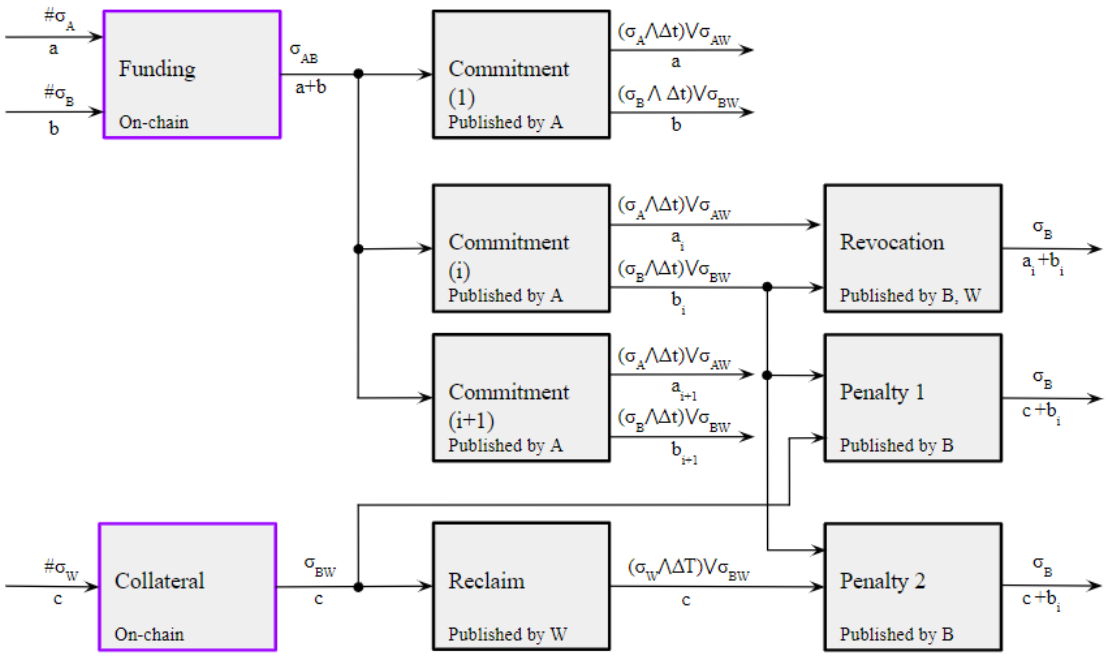

Fig. 1: The transactions of a Cerberus channel and their dependencies.

[6] We assume that a rational watchtower will publish the reclaim transaction at the latest when the channel is closed.

5 Security Analysis

5.1 Security

We show that Cerberus channels are secure within our system model under any collusion/bribing scheme involving the channel parties and the watchtower.

Lemma 1. *Phase Abort does not affect the security of a* Cerberus *channel,* i.e.*, no honest party or watchtower can be cheated out of its funds.*

Proof. To prove the lemma, we distinguish three cases:

(a) *Abort* terminates before *Close* initiates.
(b) *Abort* terminates during *Close*.
(c) *Abort* includes *Close* entirely, *i.e.*, *Close* initiates at most $T - t$ after the reclaim transaction is put on-chain.

In the first case, the watchtower withdraws the collateral, and is not liable anymore for the channel operation. Specifically, W publishes on-chain the reclaim transaction. Since no commitment transaction is published on-chain, all penalty transactions that can interfere with the ownership of the collateral are invalid. Hence, the watchtower claims the collateral after time T elapses – before *Close* initiates. From that point on, the security of the channel – the funds of the hiring party – is the same as in a Lightning channel, unless the channel is updated with a new watchtower service.

For the second case, suppose *Abort* initiates at time $t = 0$, and there is a time t', such that $T-t < t' < T$, in which *Close* initiates. If the closing of the channel is collaborative between the parties or the last commitment transaction was published by one of the parties, there are no valid penalty transactions that can interfere with the ownership of the collateral. Hence, the collateral will be owned by the watchtower at time T. Further, the funds of the channel parties will be distributed as last agreed. However, the channel can close with one of the parties publishing an old commitment transaction. In this case, both penalty transactions become valid at time $t' + t > T$. But the watchtower can spend the collateral at time T, before any penalty transaction becomes valid. Therefore, when Abort terminates during *Close*, the watchtower can safely reclaim the collateral, no matter how a Cerberus channel closes. Note, however, that the watchtower is not liable for the channel's correct operation when Abort terminates during *Close*. Thus, the watchtower is not obligated to publish the revocation transaction in time. Nevertheless, the hiring party comes online once during the *Abort* phase. From then on, it comes online every t time, since it realizes that the watchtower stops offering its service. Furthermore, it holds $t' + t > T$. Therefore, the hiring party will notice the fraud in time and will publish the revocation transaction. Hence, no honest hiring party or watchtower can be cheated out of its funds when Abort terminates during *Close*.

In contrast to the first two cases, when *Abort* includes *Close* entirely, the watchtower is still responsible for the correct operation of the channel. In such a case, the security of a Cerberus channel is the same with or without phase *Abort*. Thus, overall, *Abort* does not affect the security of a Cerberus channel. □

Next, we show that any honest party will maintain at least its funds in a Cerberus channel, even against malicious parties.

Lemma 2. *Any honest party involved in a* Cerberus *channel cannot be cheated out of its funds.*

Proof. Watchtowers are incentivized to participate in Cerberus channels due to the occasional rewards on every update. Thus, we need only show that under any collusion scheme Cerberus channels remain secure with respect to the system model of Section 3.1. This implies the scheme is also secure if no collusion occurs, *e.g.*, normal channel operation or the watchtower is offline. Note that whenever we assume collusion, it can also be the case that the same person handles both colluding identities. There can be the following collusion schemes:

(i) Both parties of the channel A and B collude. According to the cryptographic assumptions, the signature of the watchtower cannot be forged, hence the parties cannot create a penalty transaction without the collaboration of the watchtower. Moreover, the channel parties only hold the watchtower's signature for revocation and penalty transactions of previous commitment transactions. The reclaim transaction is only held by W and phase *Abort* does not affect the security of the protocol (Lemma 1). Therefore, the only available action for the colluding parties is to publish a previous commitment transaction on purpose. In such a case, an honest hence online watchtower publishes the corresponding revocation transaction. As soon as the revocation transaction is on-chain, one of the inputs of both penalty transactions – which is the same as the one in the revocation transaction – become invalid thus both penalty transaction become invalid. Therefore, the malicious parties cannot claim an honest watchtower's collateral.

(ii) Watchtower W colludes with party A. In this case, the malicious parties try to cheat B's balance out of the channel (equivalent to at most $a+b$ coins), while B is offline. We assume *Abort* has not initiated, since it does not affect the security of the channel[7] (Lemma 1). The colluding parties cannot forge B's signature to close the channel in a new state. Thus, the only available action is to publish a previous and more favorable to A commitment transaction. However, for each previous commitment transaction, party B holds two corresponding penalty transactions that award the collateral of the watchtower to B. Therefore, as soon as B goes online (within the time window T), B will publish the suitable penalty transaction on-chain (type 2 if the reclaim transaction is on-chain, type 1 otherwise) and claim the watchtower's collateral, $c > a + b$ coins. Note that this argument holds because the watchtower's collateral is locked on-chain involving the hiring party and thus cannot be used in parallel for other channels/parties.

(iii) In the last case, B colludes with watchtower W. This is the simplest case, since A is either online or employs its own watchtower. If A is online the security holds similarly to the Lightning protocol, *i.e.*, A publishes the revocation transaction on-chain and receives all the funds of the channel. If A employs a watchtower, then the previous analysis holds. □

Lemma 3. *A* Cerberus *channel will not close in a state that is not the last state agreed by all the participants of the channel.*

Proof. Any party of a Cerberus channel cannot gain more profit by deviating from the honest protocol execution, because all parties involved in a Cerberus channel always maintain (at least) their funds as shown in Lemma 2. Hence, a rational party that aims to maximize its profit will honestly follow the Cerberus protocol in every phase, and ultimately close the Cerberus channel in a non-cheating state, as described in Section 4. □

Lemma 4. *Any party of a* Cerberus *channel can close it at any time.*

Proof. Both parties of the Cerberus channel hold at least one valid commitment transaction (the one created at the last execution of the Update protocol, or if no update has taken place, the unique commitment transaction created during the Open protocol) which allows them to initiate phase *Close*, unilaterally, as described in Section 4.3. Hence, a Cerberus channel will be closed at the latest within time t from publishing a commitment transaction on-chain (given a perfect underlying blockchain protocol and network synchrony). □

Theorem 1. Cerberus *channels achieve Security (as defined in Section 3.3).*

Proof. From Lemma 4, it holds that any party can close a Cerberus channel at any time. Thus, to prove security it is enough to show that a Cerberus channel can only close on the last agreed by all parties state, which is shown in Lemma 3. Hence, any party of a Cerberus channel can enforce the last agreed state on-chain at any time. □

[7] To be precise, *Abort* could have initiated but less than $T - t$ time has elapsed between the reclaim transaction was published on-chain and the time *Close* initiated.

5.2 Performance

Next, we show that CERBERUS channels scale well, meaning that the number of on-chain transactions is constant and independent of the number of transactions executed in a channel, similarly to the Lightning protocol. The analysis below considers a single watchtower for each party of the channel. We discuss the scalability of the protocol if we enable multiple watchtowers in Section 7.

Theorem 2. CERBERUS *channels achieve Scale-out.*

Proof. In phase *Open*, a CERBERUS channel requires 3 transactions, one funding transaction to open the channel and two collateral transactions for each watchtower to lock the collateral to the corresponding party of the channel.

Phase *Update* is executed off-chain, hence no transactions are published on-chain.

During phase *Close*, the number of on-chain transactions can vary, however in the worst case 4 transactions are published. Specifically, in case of a non-cheating closure 3 transactions are published: (i) either a commitment transaction published unilaterally by a party of the channel, or a collaborative closing transaction published by both parties, (ii)-(iii) one reclaim transaction for each of the parties' watchtowers, published by the corresponding watchtowers respectively.

On the contrary, in case of fraud and a responsive watchtower, the following 4 transactions are necessary: (i) an old commitment transaction published by the cheating party (step 1, protocol CLOSE (B)), (ii) the reclaim transaction of the cheating party's watchtower, published by the watchtower, (iii) the revocation transaction from the cheated party's watchtower, published by the watchtower (step 2, protocol CLOSE (B)), and (iv) the reclaim transaction of the cheated party's watchtower, published by the watchtower (step 3, protocol CLOSE (B)).

Furthermore, in case of fraud and an unresponsive watchtower, up to 4 transactions are published on-chain, as illustrated in Protocol CLOSE (C): (i) an old commitment transaction published by the cheating party, (ii) the reclaim transaction of the cheating party's watchtower (published by the watchtower), (iii) the reclaim transaction of the cheated party's watchtower (published by the watchtower), and (iv) the corresponding penalty transaction published by the cheated party.

Phase *Abort* is an optional intermediate phase, that allows the watchtower to withdraw its service to the channel party. This phase includes one on-chain transaction – the reclaim transaction – but not additively to phase *Close*. After phase *Abort* the protocol for the party is similar to the Bitcoin Lightning protocol, which requires at most 3 on-chain transactions in total for a channel operation. Thus, the worst case performance analysis does not include phase *Abort*.

Overall, a CERBERUS channel requires at most 7 on-chain transactions. □

6 Cerberus transactions script

In the following specification we assume a channel between Alice and Bob, where only Bob is using a watchtower.

The funding and the collateral transactions both have a 2-of-2 multisig output. Its script is

`2 <pubkey1> <pubkey2> 2 OP_CHECKMULTISIG`,

where `pubkey1`, `pubkey2` are the funding public keys of Alice and Bob for the funding transaction and the collateral public keys of Bob and the watchtower, sorted by ascending order of their DER-encodings[8].

The reclaim transaction has a unique input that spends the collateral transaction multisig output with a witness script

`0 <collateral_pubkey1_sig> <collateral_pubkey2_sig>`.

It also has a single output, with script as shown in Fig. 2. The two penalty public keys are those of Bob and the watchtower, sorted by ascending order of their DER-encodings.

[8] `https://github.com/bitcoin/bips/blob/master/bip-0066.mediawiki`

```
OP_IF
    # Penalty
    2
    <penalty_pubkey1>
    <penalty_pubkey2>
    2
    OP_CHECKMULTISIG
OP_ELSE
    # Normal
    <long_delay>
    OP_CHECKSEQUENCEVERIFY
    OP_DROP
    <watchtower_penalty_pubkey>
    OP_CHECKSIG
OP_ENDIF
```

Fig. 2: Reclaim transaction output script.

The commitment transaction has a unique input that spends the funding output with witness script `0 <pubkey1_sig> <pubkey2_sig>`. It also has two outputs, the scripts of which are slight variations of each other. The exact scripts can be found in Fig. 3 and 4.

For the first output, the two revocation public keys are those of Alice and the watchtower, sorted by ascending order of their DER-encodings. Similarly sorted are the revocation public keys of Bob and the watchtower in the second output.

```
OP_IF
    # Revocation
    2
    <revocation_pubkey1>
    <revocation_pubkey2>
    2
    OP_CHECKMULTISIG
OP_ELSE
    # Normal
    <bob_delay>
    OP_CHECKSEQUENCEVERIFY
    OP_DROP
    <alice_delayed_pubkey>
    OP_CHECKSIG
OP_ENDIF
```

Fig. 3: Commitment transaction 1st output script.

The revocation transaction spends the two outputs of the commitment transaction following the revocation path. The witness script of both inputs is `0 <revocation_pubkey1_sig> <revocation_pubkey2_sig> 1` with the appropriate signatures for each output. It has a single P2WPKH[9] output to Bob's public key.

The penalty transaction 1 spends the second output of the commitment transaction and the output of the collateral transaction. It also has a single plain P2WPKH output to Bob's public key. The first input

[9] `https://wiki.trezor.io/P2WPKH`

```
OP_IF
    # Revocation
    2
    <revocation_pubkey1>
    <revocation_pubkey2>
    2
    OP_CHECKMULTISIG
OP_ELSE
    # Normal
    <alice_delay>
    OP_CHECKSEQUENCEVERIFY
    OP_DROP
    <bob_delayed_pubkey>
    OP_CHECKSIG
OP_ENDIF
```

Fig. 4: Commitment transaction 2nd output script.

follows the normal path and has a witness script `<bob_delayed_pubkey_sig> 0`, whereas the second input has a witness script `0 <collateral_pubkey1_sig> <collateral_pubkey2_sig>`.

Lastly, the penalty transaction 2 is very similar with penalty transaction 1, except that does not spend the collateral, but the reclaim transaction. Explicitly, penalty transaction 2 spends the second output of the commitment transaction and the output of the reclaim transaction following the penalty path. It also has a single plain P2WPKH output to Bob's public key. The first input follows the normal path and has a witness script `<bob_delayed_pubkey_sig> 0`, whereas the second input has a witness script `0 <penalty_pubkey1_sig> <penalty_pubkey2_sig> 1`.

A proof-of-concept implementation of the necessary transactions can be found in `https://github.com/OrfeasLitos/cerberus-script`.

7 Limitations and future work

Privacy. Cerberus channels maintain the privacy property, as defined in Section 3.3, assuming that the watchtowers are considered internal to the channel parties. This means that the transactions executed off-chain during the phase *Update* are known only to the parties of the channel and the hired watchtowers. Any other third-party, external to the channel, does not have any knowledge on the state of the channel. Nevertheless, Cerberus channels do not guarantee privacy from the watchtowers. Although Lightning watchtowers preserve the privacy of transactions from watchtowers, they also suffer from inadequate incentives for participation in the system. On the other hand, Cerberus channels provide the necessary incentive mechanisms to guarantee security, but watchtowers are aware of all transactions executed in the channel. Introducing stronger privacy mechanisms while maintaining the appropriate incentives is left for future work.

Extension to multiple watchtowers. To enhance security against possible crash failures or withdrawal of service of a watchtower, parties can employ multiple watchtowers. In such a case, the number of on-chain transactions grows linearly with the number of hired watchtowers. Every watchtower needs to lock its collateral on-chain, thus one collateral transaction per watchtower is needed. However, the sum of all watchtowers' collateral remains the same, to guarantee security; that is at least greater than the total funds locked in the channel by both parties. This way the counterparty of the channel cannot bribe all watchtowers since the sum of the required bribes will exceed the party's potential gain. Therefore, there will be at least one watchtower that will publish the revocation transaction in case of fraud.

Rewards and Collateral. Currently, in a CERBERUS channel, rewards are awarded to the watchtowers on every update of the channel by the hired party via a one-way channel. Ideally, these rewards should be returned to the hired party if the watchtower misbehaves. To this end, we can modify the collateral transaction to build a bidirectional payment channel in which both the watchtower locks collateral and the hiring party locks future rewards. Then, during an update of the CERBERUS channel and upon receiving the signed penalty transaction, the hiring party can update the distribution of funds in this channel, rewarding the watchtower for its active participation. Note that this process does not require a fair exchange protocol since the watchtower can simply withdraw its service in case the hiring party does not pay the reward, similarly to the current protocol. However, this modification implies that the watchtowers' rewards will be locked during the lifetime of the channel.

Assumptions. In every channel construction that the counterparty is allowed to publish on-chain a valid outdated state, timelocks are necessary to secure the construction. Assuming distrusting parties (including the watchtower), this implies that every party must be online once in a while to verify the correct operation of the construction. Hence, at best we can alleviate the availability assumption, but not abolish it completely. In turn, due to timelocks, the security of CERBERUS channels depends on synchrony assumptions and a perfect blockchain that cannot be censored. Although we enable shorter revocation periods and parties can go offline for an extended period of time, we cannot decouple the dependency of the security of payment channels on Bitcoin from the liveness and synchrony of the underlying blockchain.

References

1. Merge blocks. `https://www.coindesk.com/laolu-building-watchtower-fight-bitcoin-lightning-fraud`. Accessed: 2018-06-29.
2. Real-time lightning network statistics. `https://1ml.com/statistics`. Accessed: 2019-09-17.
3. G. Avarikioti, E. K. Kogias, and R. Wattenhofer. Brick: Asynchronous state channels. arXiv preprint: 1905.11360, 2019.
4. G. Avarikioti, F. Laufenberg, J. Sliwinski, Y. Wang, and R. Wattenhofer. Towards secure and efficient payment channels.
5. K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.
6. C. Decker, R. Russell, and O. Osuntokun. eltoo: A simple layer2 protocol for bitcoin.
7. C. Decker and R. Wattenhofer. A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In *Stabilization, Safety, and Security of Distributed Systems*, pages 3–18. Springer, Aug. 2015.
8. T. Dryja. Unlinkable outsourced channel monitoring.
9. S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski. Perun: Virtual payment hubs over cryptographic currencies. Technical report, IACR Cryptology ePrint Archive 2017, 2017.
10. J. Garay, A. Kiayias, and N. Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT 2015*, pages 281–310. Springer, 2015.
11. M. Green and I. Miers. Bolt: Anonymous payment channels for decentralized currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 473–489. ACM, 2017.
12. L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais. Sok: Off the chain transactions. *IACR Cryptology ePrint Archive*, 2019:360, 2019.
13. M. Khabbazian, T. Nadahalli, and R. Wattenhofer. Outpost: A responsive lightweight watchtower. Cryptology ePrint Archive, Report 2019/986, 2019. `https://eprint.iacr.org/2019/986`.
14. P. McCorry, S. Bakshi, I. Bentov, A. Miller, and S. Meiklejohn. Pisa: Arbitration outsourcing for state channels. *IACR Cryptology ePrint Archive*, 2018:582, 2018.
15. A. Miller, I. Bentov, R. Kumaresan, and P. McCorry. Sprites: Payment channels that go faster than lightning. *CoRR*, abs/1702.05812, 2017.
16. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
17. J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments. 2015.
18. J. Spilman. Anti dos for tx replacement. `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html`. Accessed: 2019-04-17.