Efficient Factories For Lightning Channels

John Law

December 21, 2022

Version 1.0

Abstract

Burchert, Decker and Wattenhofer introduced the idea of factories that allow multiple two-party channels to be created and re-sized with a single on-chain transaction and they created a protocol for a factory that can be closed unilaterally in $O(\log S)$ time using $O(\log S)$ on-chain transactions, assuming the factory supports S states. This paper presents protocols for factories that can be closed unilaterally in $O(1)$ time using $O(1)$ on-chain transactions, thus further improving the scalability of Bitcoin and the Lightning Network. No change to the underlying Bitcoin protocol is required.

1 Overview

Factories that allow multiple two-party Lightning **[AOP21][BOLT][PD16]** channels to be created, re-sized and closed with a small number of on-chain transactions are essential to the scalability of Bitcoin **[BDW18]**. Let P denote the number of parties, C denote the number of channels created, and S denote the number of factory states supported. The most efficient previously-known factory, created by Burchert, Decker and Wattenhofer **[BDW18],** can be closed unilaterally in $O(\log S)$ time using $O(\log S)$ on-chain transactions and $O(C + \log S)$ on-chain bytes[1]. For any given factory state, a single fixed transaction is used to instantiate the factory's channels when it is closed unilaterally, so the parties using the factory can maintain just one version of each off-chain channel state. Performing a unilateral close with $O(\log S)$ on-chain transactions requires that the party closing the factory interact with the blockchain at $O(\log S)$ different blockheights (so a unilateral close is an $O(\log S)$-shot procedure **[Law22a]**), which could be awkward for some users **[Zmn22]**.

This paper presents two protocols for factories that can be closed unilaterally in $O(1)$ time using $O(1)$ on-chain transactions. The first protocol, called the Tunable-Penalty Factory (TPF) protocol, requires only $O(C)$ on-chain bytes for a unilateral close, but it can use P different transactions to instantiate the factory's channels, thus forcing the parties using the factory to maintain P different versions of each

1 See Appendix A of **[Law22c]** for an analysis of how the number of on-chain transactions and the factory close time can be traded-off. The byte analysis presented here assumes that Schnorr signatures are used.

off-chain channel state. The second protocol, called the Single-Commitment (SC) protocol, requires $O(C + \log S)$ on-chain bytes for a unilateral close, but it uses only a single transaction to instantiate the factory's channels, so multiple versions of off-chain channel states are not required. The TPF protocol is particularly simple and allows a 2-shot unilateral close (that is, the party closing the channel only has to perform actions at 2 different blockheights).

Both protocols are based on the Tunable-Penalty Channel (TPC) protocol **[Law22b]**[2] and they share many of its properties, including:

- tunable penalties for putting old transactions on-chain, and
- watchtowers with storage that is logarithmic in the number of factory states supported (but linear in the number of parties in the factory).

No change to the underlying Bitcoin protocol is required.

The rest of this paper is organized as follows. The TPF protocol is presented in Section 2. Section 3 describes how that protocol can be modified to obtain the SC protocol. Sections 4 and 5 give related work and conclusions. A proof of correctness for the SC protocol is given in Appendix A.

2 The Tunable-Penalty Factory (TPF) Protocol

2.1 Overview

The Tunable-Penalty Factory (TPF) protocol is a slight modification of the Tunable-Penalty Channel (TPC) protocol **[Law22b]**.

Each party using the TPC protocol has their own on-chain Individual transaction, the output of which they spend with their State transaction. This State transaction is a control transaction and its first output's value is equal to the desired penalty for putting an old State transaction on-chain. This first output can be spent by the same party's Commitment transaction for the same state, but only after a relative delay equal to the maximum of the parties' *to_self_delay* parameters[3]. The relative delay gives the other party time to revoke an old State transaction by spending its first output and thus claiming the penalty. The TPC protocol's State transaction also has an HTLC control output for each HTLC that is active in that state. The TPC protocol revokes old State transactions with per-commitment keys that can be known to all parties, rather than with revocation keys that cannot be known by the party putting the revocable transaction on-chain **[Law22b]**.

The TPC protocol is modified to create the TPF protocol by:

2 The Tunable-Penalty Channel (TPC) protocol was called the Tunable-Penalty (TP) protocol in that paper. It is referred to here as the TPC protocol in order to distinguish it from the Tunable-Penalty Factory (TPF) protocol.

3 A party's *to_self_delay* parameter is a safety parameter that is the minimum relative delay before another party can claim the output of a revocable transaction that they put on-chain **[AOP21][BOLT][PD16]**. The delay is set to be large enough to guarantee that a revoked transaction can be detected and its output can be spent (thus revoking it) before being claimed by the party that put it on-chain.

- eliminating the HTLC control outputs from the State transactions,
- modifying the Commitment transactions to have channel outputs, each of which is owned by two parties in the factory (rather than having two individually-owned outputs plus HTLC outputs), and
- supporting $P > 2$ parties by allowing each party to have their own Individual, State and Commitment transactions.

2.2 Protocol Specification

Protocol Transactions

The resulting TPF protocol is shown in Figure 1.

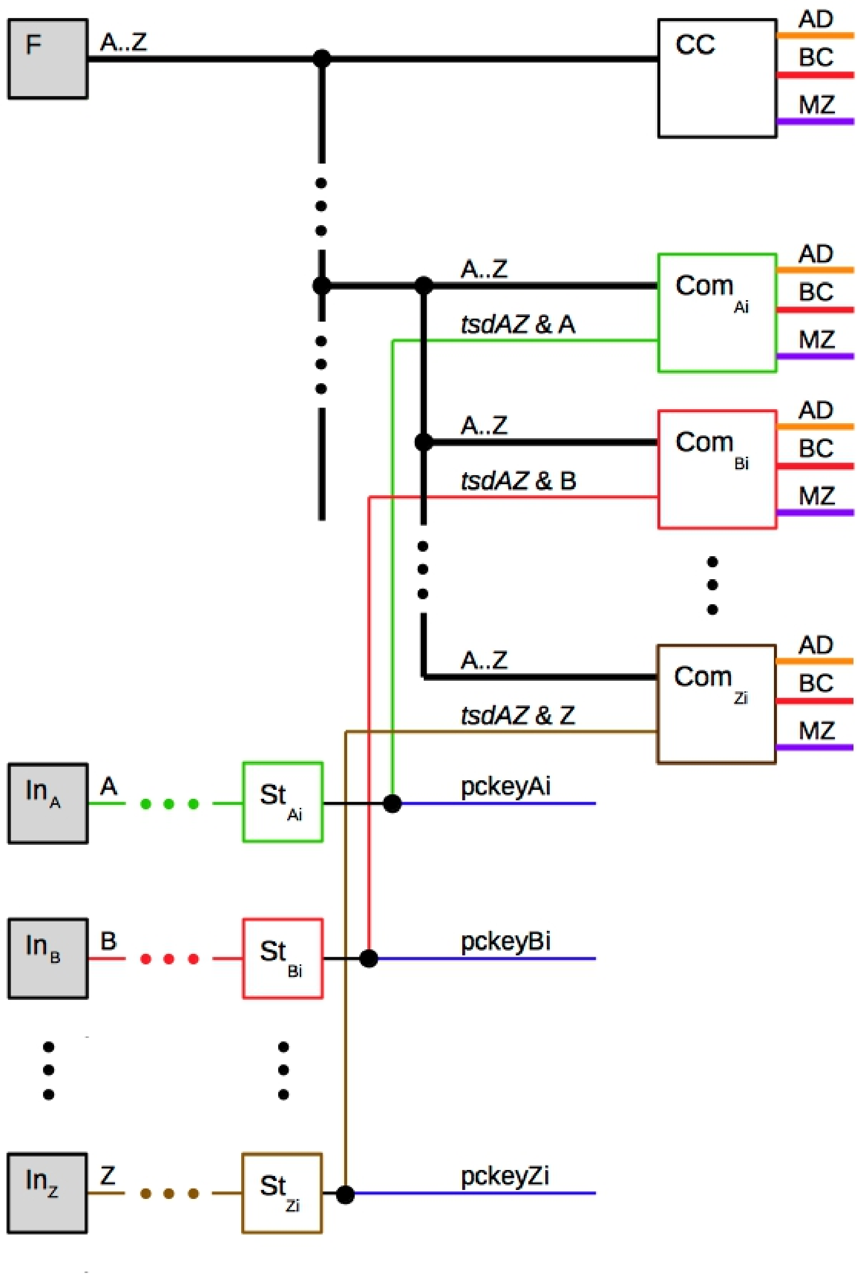

Figure 1. The Tunable-Penalty Factory (TPF) Protocol. Each of the *P* parties has their own Individual (In), State (St) and Commitment (Com) transactions. Old State transactions are revoked using per-commitment keys that are known to all parties.

In Figure 1 (and throughout the paper):

- **{A..Z}** denotes {A..Z}'s signature (a single party's signature),
- **A..Z** denotes A..Z's signature (every party's signature),

- pairs of capital letters indicate signatures from those two parties,
- **pckey{A..Z}i** denotes a signature using a per-commitment key for revoking {A..Z}'s state *i* transaction, and
- ***tsdAZ*** denotes the maximum of the *to_self_delay* parameters set by the *P* parties.

Shaded boxes represent transactions that are on-chain, while unshaded boxes represent off-chain transactions. Each box includes a label showing the transaction type, namely:

- **F** for the Funding transaction,
- **CC** for a Cooperative Close transaction,
- **In** for an Individual transaction,
- **St** for a State transaction, and
- **Com** for a Commitment transaction.

Subscripts denote which party can put the transaction on-chain (if only one party can do so) and which channel state the transaction is associated with (namely state *i* in the figure).

Bold lines carry factory funds and thin solid lines have value equal to the tunable penalty amount. When a single output can be spent by multiple off-chain transactions, those transactions are said to *conflict*, and only one of them can be put on-chain. A party will be said to *submit* a transaction when they attempt to put it on-chain.

Protocol Operation

The operation of the TPF protocol is based on that of the TPC protocol **[Law22b]**.

In order to establish a new factory state, all parties:

1. calculate the State and Commitment transactions for the new state[4],
2. exchange partial signatures for the new state's Commitment transactions, and
3. exchange per-commitment keys for the old state, thus revoking it.

All parties constantly look for old (revoked) State transactions put on-chain by other parties, and if they find such a transaction they use the corresponding per-commitment key to spend its first output, thus obtaining the penalty funds and revoking the old state. Once a State transaction has been on-chain for *tsdAZ* without its first output being spent, the party that put the State transaction on-chain can attempt to put their corresponding Commitment transaction on-chain at any time[5]. Because old State

4 This step requires that each party shares their per-commitment pubkey for the new state with each of the other parties.

5 Unlike the TPC protocol, the TPF protocol allows parties to arbitrarily delay putting their Commitment transaction on-chain. This is because TPF protocol does not support HTLCs at the factory-level, so any party with an unrevoked State transaction can put their corresponding Commitment transaction on-chain.

transactions are revoked before their corresponding Commitment transaction can be put on-chain, and because Commitment transactions require signatures from all parties, only current Commitment transactions can be put on-chain.

Analysis

Any party can close the factory unilaterally by putting their current State transaction on-chain, waiting until it has been on-chain for *tsdAZ*, and then submitting their corresponding Commitment transaction. Either that party's Commitment transaction, or a conflicting Commitment transaction from another party, will then appear on-chain. As a result, unilateral factory closes are 2-shot and require $O(1)$ on-chain transactions and time.

As was the case with the TPC protocol, per-commitment keys can use the Lightning protocol's compact storage technique for revocation keys to consume only $O(\log S)$ storage to revoke a maximum of S old transactions that could be put on-chain by a single other party **[Rus][Law22b]**. Therefore, each party requires $O(\mathrm{P}{*}\log S)$ storage to hold the per-commitment keys for all of the P parties.

Watchtowers

If desired, parties using the TPF protocol can choose to use an untrusted watchtower to monitor the blockchain and revoke old transactions put on-chain by other parties. Because the watchtower can know per-commitment keys for revoked transactions, each watchtower requires $O(\mathrm{P}{*}\log S)$ storage to hold the per-commitment keys for all of the P parties.

However, note that the parties using the TPF protocol can set their *to_self_delay* parameters to extremely large values (such as 1-3 months in the case of a casual user **[Law22a]**), thus allowing them to be unavailable for long periods without having to rely on a watchtower. As long as the channels created by the TPF factory use factory-optimized protocols, such as the PFO, FFO or FFO-WF protocol **[Law22c]**, the large *to_self_delay* parameters will have no effect on the time to resolve HTLCs within the channels.

Off-Chain Control Outputs

As was the case for the TPC protocol **[Law22b]**, a party that operates multiple factories using the TPF protocol can use a single UTXO to fund the inputs to the State transactions in all of their factories.

3 The Single-Commitment (SC) Factory Protocol

3.1 Overview

Note that the TPF protocol uses P conflicting Commitment transactions to establish the factory state and to instantiate the factory's channels. Therefore, a separate channel state must be maintained and updated for each of the P versions of the channel that could be instantiated, thus increasing the computation, storage, and communication for channels by a factor of P. The result could be quite expensive if there are a large number of parties in the factory.

The SC protocol eliminates this factor of P blow-up by having all of the parties use a single shared Commitment transaction. Like the TPF protocol, the SC protocol uses revocable State transactions, each of which spends a single party's Individual transaction output, to ensure that only the current Commitment transaction can be put on-chain. The challenge is how to use a single shared Commitment transaction that depends on the value of an unrevoked State transaction without actually spending any of the outputs of that State transaction (as spending a State transaction output would make the Commitment transaction's input dependent on which party's State transaction it spends, thus preventing the use of a single shared Commitment transaction). This challenge is solved by introducing a shared Trigger transaction and a per-user Mold transaction, where the Commitment and Mold transactions compete for the Trigger transaction's outputs. The Mold transaction is put on-chain prior to the Commitment transaction, and it constrains the Commitment transactions that can be put on-chain somewhat like how a mold shapes a liquid that is poured into it.

3.2 Protocol Specification

Protocol Transactions

The SC protocol is shown in Figure 2, which includes the Trigger (Tr) and Mold transactions.

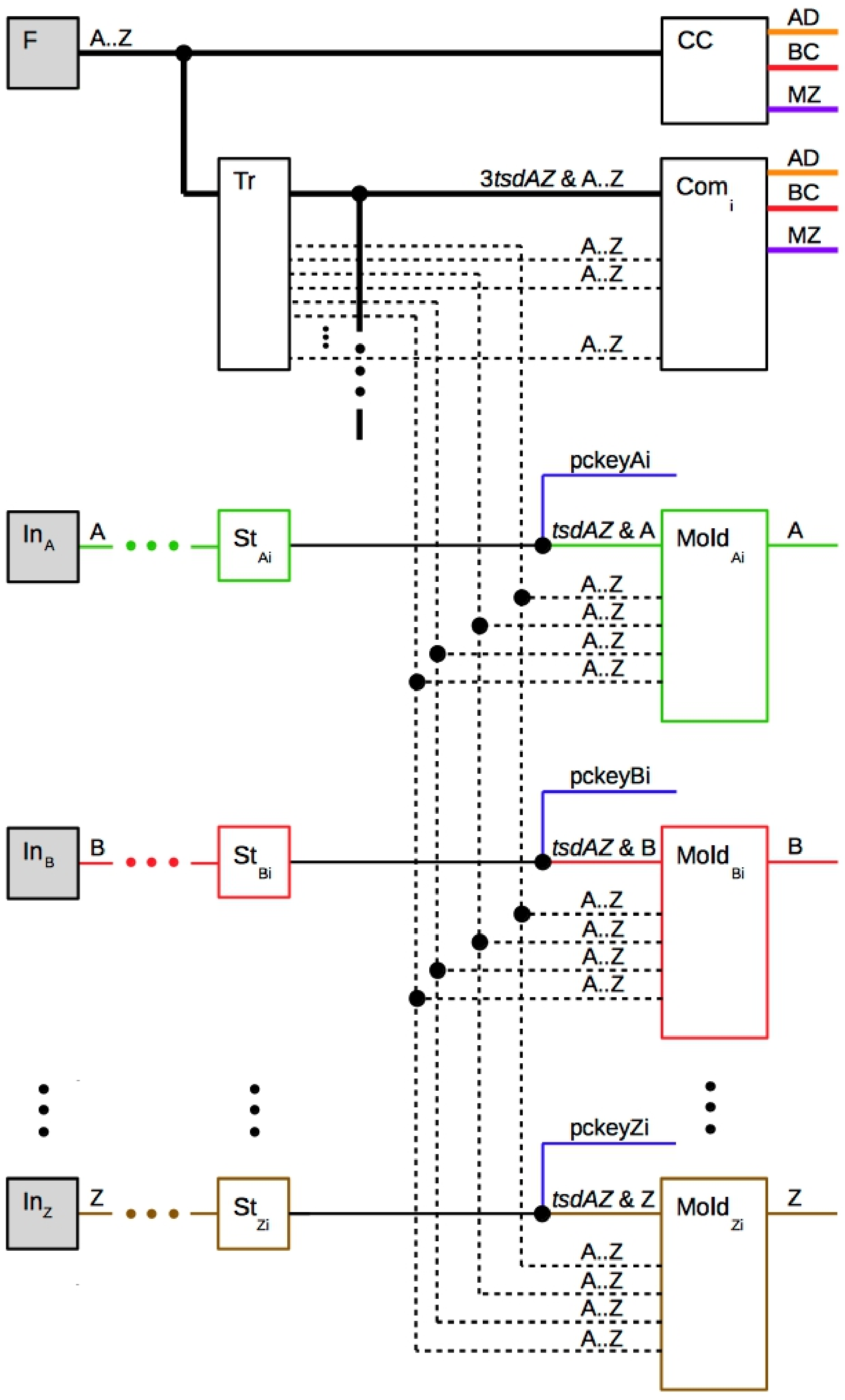

Figure 2. The SC Factory Protocol. The Trigger transaction has $\log_2 S$ control outputs that encode the current state number, with the Mold and Commitment transactions competing to spend those outputs.

The SC protocol takes a parameter *S* that determines the maximum number of factory states supported[6], where *S* is a power of 2. The Trigger transaction has one value output and $\log_2 S$ control outputs, numbered 0 .. $\log_2 S - 1$. Commitment transaction *i*, $0 \le i \le S - 1$, spends those Trigger control outputs *b*, $0 \le b \le \log_2 S - 1$, such that bit position *b* of the binary representation of *i* is a 0. Each Mold transaction *i*, $0 \le i \le S - 1$, spends the output of the same party's State *i* transaction, and Trigger control outputs *b*, $0 \le b \le \log_2 S - 1$, such that bit position *b* of the binary representation of *i* is a 1. States 0 through S - 1 are supported, with the exception of states of the form 2^b where $1 \le b \le \log_2 S - 1$[7].

Protocol Operation

The operation of the SC protocol is similar that of the TPF protocol.

In order to establish a new factory state, all parties:

4. calculate the State, Mold and Commitment transactions for the new state[8],
5. exchange partial signatures for the new state's Commitment and Mold transactions (in that order), and
6. exchange per-commitment keys for the old state, thus revoking it.

All parties constantly look for old (revoked) State transactions put on-chain by any party, and if they find such a transaction they use the corresponding per-commitment key to spend its first output, thus obtaining the penalty funds and revoking the old state.

In order to put the factory on-chain, a party submits the Trigger transaction and their current (unrevoked) State transaction to the blockchain. As soon as their State transaction has been on-chain for *tsdAZ*, that party submits their Mold transaction for the same state to the blockchain. Once the Trigger transaction has been on-chain for at least 3*tsdAZ*, that party submits to the blockchain the Commitment transaction for the same state as the on-chain Mold transaction.

Also, all parties constantly monitor the blockchain for the Trigger transaction, and if they find it on-chain they use the above protocol for putting the factory on-chain[9] as soon as possible.

Finally, whenever a party detects either the Trigger transaction or an unrevoked State transaction on-chain, that party stops updating the factory state.

Correctness

The rules for how the Commitment and Mold transactions spend the Trigger transaction's control outputs guarantee that once a Mold transaction for state *i* is fixed on-chain, only a Commitment

6 Specifically, $S - \log_2 S + 1$ states are supported.
7 These exceptions exist in order to prevent Mold transactions for two consecutive states from both being put on-chain where they would prevent any current Commitment transaction from being put on-chain. When state *i* is supported but state *i*+1 is not, the next state after *i* is *i*+2.
8 This step requires that each party shares their per-commitment pubkey for the new state with each of the other parties.
9 Minus submitting the Trigger transaction to the blockchain, as it is already on-chain.

transaction for the same state i, or for a later state, can be put on-chain. Because only Mold transactions for unrevoked State transactions can be put on-chain, and because only the latest State transactions can be unrevoked, only the latest (and thus current) Commitment transactions can be put on-chain.

This reasoning is formalized in the proof of correctness in Appendix A.

Analysis

While the SC protocol requires $O(1)$ time and $O(1)$ on-chain transactions for a unilateral factory close, the Trigger transactions have $O(\log S)$ outputs, Mold transactions have $O(\log S)$ inputs, and Commitment transactions have $O(\log S)$ inputs and $O(C)$ outputs and , so the total on-chain bytes required for a unilateral close is $O(C + \log S)$.

Also, note that if a Trigger transaction is detected on-chain, all parties put their State transactions on-chain, thus adding $O(P)$ on-chain transactions and on-chain bytes. In the case of a party that is closing the factory unilaterally according to the protocol, the race by all parties to put their State transactions on-chain is excessive.

One way to address this inefficiency is to increase the relative delay from the Trigger transaction to the Commitment transaction from $3tsdAZ$ to $6tsdAZ$. When a party follows the protocol and closes the factory unilaterally, their current Mold transaction will be on-chain within $3tsdAZ$ of the Trigger transaction. The remaining parties can therefore wait until $3tsdAZ$ after the Trigger transaction is on-chain before they put their State transactions on-chain. If no Mold transaction is on-chain within $3tsdAZ$ of the Trigger transaction, the remaining parties realize that the party putting the Trigger transaction on-chain is not following the protocol, so they need to put their State and Mold transactions on-chain as quickly as possible in order to guarantee that only a current Commitment transaction can be put on-chain.

4 Related Work

The concept of creating a Lightning channel factory, as well as the most efficient published protocol for such a factory, was presented by Burchert, Decker and Wattenhofer **[BDW18]**. The protocols given here differ in only requiring $O(1)$ time and $O(1)$ on-chain transactions for a unilateral close.

A number of researchers have proposed changes to Bitcoin in order to support simpler and more efficient factories. The eltoo factory protocol **[DRO18]** has a particularly simple structure and it allows the parties to maintain only one version of each off-chain channel state. It requires $O(1)$ time, $O(1)$ on-chain transactions and $O(C)$ on-chain bytes for a unilateral close with C channels. However, a malicious party could delay the closing of the factory until $O(S)$ transactions are put on-chain, if S states are supported and the malicious party is willing to pay the required fees. The eltoo protocol differs from the protocols presented here by requiring a change to Bitcoin, namely the support for BIP 118 **[BIP118]**.

The TPF and SC protocols are based on the TPC protocol presented by Law **[Law22b]**.

5 Conclusions

This paper presents new factory protocols that require $O(1)$ time and $O(1)$ on-chain transactions for a unilateral close. They are the first factory protocols known that achieve those bounds with the existing Bitcoin protocol. The ability to unilaterally close a factory with just two or three submissions to the blockchain, with a fixed delay between them, is quite a bit simpler than closing previously-published factories **[BDW18]**. As a result, it is hoped that TPF and SC factories will lead to the increased use of factories in practice, thus improving the scalability of Bitcoin and Lightning.

References

AOP21 Andreas Antonopoulos, Olaoluwa Osuntokun and Rene Pickhardt. Mastering the Lightning Network, 1st. ed. 2021.

BIP118 BIP 118: SIGHASH_ANYPREVOUT. See https://anyprevout.xyz/ and https://github.com/bitcoin/bips/pull/943.

BDW18 Conrad Burchert, Christian Decker and Roger Wattenhofer. Scalable Funding of Bitcoin Micropayment Channel Networks. In Royal Society Open Science, 20 July 2018. See http://dx.doi.org/10.1098/rsos.180089.

BOLT BOLT (Basis Of Lightning Technology) specifications. See https://github.com/lightningnetwork/lightning-rfc.

DRO18 Christian Decker, Rusty Russell and Olaoluwa Osuntokun. eltoo: A Simple Layer2 Protocol for Bitcoin. 2018. See https://blockstream.com/eltoo.pdf.

Law22a John Law. Watchtower-Free Lightning Channels For Casual Users. See https://github.com/JohnLaw2/ln-watchtower-free.

Law22b John Law. Lightning Channels With Tunable Penalties. See https://github.com/JohnLaw2/ln-tunable-penalties.

Law22c John Law. Factory-Optimized Channel Protocols For Lightning. See https://github.com/JohnLaw2/ln-factory-optimized.

PD16 Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments (Draft Version 0.5.9.2). January 14, 2016. See https://lightning.network/lightning-network-paper.pdf.

Rus Rusty Russell. Efficient Per-Commitment Secret Storage. See https://github.com/lightning/bolts/blob/master/03-transactions.md#efficient-per-commitment-secret-storage.

Zmn22 ZmnSCPxj. Channel Eviction From Channel Factories By New Covenant Operations. See https://lists.linuxfoundation.org/pipermail/lightning-dev/2022-February/003479.html.

Appendix A: Proof of Correctness of the SC Protocol

First, note that the Trigger and Commitment transactions in the SC protocol require signatures from all parties, so only valid Trigger and Commitment transactions can be put on-chain, provided at least one party follows the protocol.

The timing model from Appendix A of **[Law22a]**, including its parameters R, S, G, B, U and L, can be used to set the timing parameters for the SC protocol and to prove its correctness.

The following Theorems are helpful in proving the correctness of the SC protocol.

Theorem 1: If $0 \leq j < i \leq S - 1$ and $0 \leq p \leq P - 1$, and if Mold_{pi} and Com_j are valid transactions as defined by the protocol, then Mold_{pi} and Com_j both spend at least one of the Trigger transaction's control outputs, and thus conflict.

Proof: Let b denote the most-significant bit position in which the binary representations of i and j differ. Because $j < i$, it follows that j has a 0 in bit position b and i has a 1 in bit position b, which implies that Mold_{pi} and Com_j both spend Trigger control output b, and thus conflict. □

Theorem 2: If at least one party follows the SC protocol and if Com_j is fixed, where $0 \leq j \leq S - 1$, then there exists a p, $0 \leq p \leq P - 1$, and an i, $0 \leq i \leq j$, such that St_{pi} has not been revoked[10].

Proof: Assume for the sake of contradiction that Com_j is fixed, but St_{pi} has been revoked for each party p, $0 \leq p \leq P - 1$, and i, $0 \leq i \leq j$. Let q denote the party that follows the protocol and let T denote the block containing the Trigger transaction. Because party q follows the protocol, they detect the Trigger transaction and their current State transaction is fixed before $T + tsdAZ$. They then submit their current Mold transaction and either that Mold transaction, or a conflicting Mold transaction, is fixed before $T + 3tsdAZ$. There are two cases:

- **Case 1:** Party q's Mold transaction is fixed before $T + 3tsdAZ$. In this case, let Mold_{qi} denote this fixed Mold transaction, and note that Mold_{qi} is a valid transaction because q follows the protocol. Therefore, it follows from Theorem 1 that $j \geq i$. Furthermore, party q's State i transaction was unrevoked when they detected the Trigger transaction on-chain, they stopped updating the factory state when they detected the Trigger transaction on-chain, and State transactions are revoked in increasing order, so it follows that their St_{qi} transaction is unrevoked, which is a contradiction.

- **Case 2:** Party q's Mold transaction is not fixed before $T + 3tsdAZ$. In this case, let Mold_{ri} denote the Mold transaction that is fixed, and note that Mold_{ri} spends at least one of the Trigger

10 To make this statement precise, we need to define a State transaction as being revoked only when all parties know its per-commitment key and agree to revoke it.

transaction's control outputs (because party *q*'s Mold transaction is not fixed, so it must conflict with party *r*'s Mold transaction that is fixed). Spending that Trigger transaction's control output requires party *q*'s signature, which means that Mold_{ri} is a valid transaction that spent the output of State transaction St_{ri}. Furthermore, St_{ri} was unrevoked when it was first detected on-chain by party *q* (because otherwise party *q* would have revoked it by spending its output). Therefore, party *q* stopped updating the factory state when they detected St_{ri}, so St_{ri} remains unrevoked by party *q*, so St_{ri} has not been revoked, which is a contradiction.

Thus in either case there is a contradiction. □

Theorem 2 shows that the Mold transactions guarantee that only current Commitment transactions can be fixed, provided at least one party follows the SC protocol.

Finally, in addition to this safety property, a party that follows the protocol can also guarantee that a current Commitment transaction will be fixed. They accomplish this by using the protocol to attempt to put the Trigger transaction, their current State and Mold transactions, and the most recent Commitment transaction on-chain. They will either accomplish this themselves, or some other party will put a conflicting transaction on-chain first. Because the new state's Commitment transaction is fully-signed before any of the new state's Mold transactions is fully-signed, because at most two consecutive states can be current at one time, and because Mold transactions for consecutive current states either conflict or include a Mold transaction for state 0 (which does not spend any of the Trigger transaction's outputs), it is always possible to put the most recent Commitment transaction on-chain.