Daric: A Storage Efficient Payment Channel With Penalization Mechanism

Arash Mirzaei, Amin Sakzad, Jiangshan Yu, and Ron Steinfeld

Faculty of Information Technology, Monash University, Melbourne, Australia
{arash.mirzaei,amin.sakzad,jiangshan.yu,ron.steinfeld}@monash.edu

Abstract. Lightning Network (LN), the most widely deployed payment channel for Bitcoin, requires channel parties to generate and store distinct revocation keys for all n payments of a channel to resolve fraudulent channel closures. To reduce the required storage in a payment channel, eltoo introduces a new signature type for Bitcoin to enable payment versioning. This allows a channel party to revoke all old payments by using a payment with a higher version number, reducing the storage complexity from $\mathcal{O}(n)$ to $\mathcal{O}(1)$. However, eltoo fails to achieve bounded closure, enabling a dishonest channel party to significantly delay the channel closure process. Eltoo also lacks a punishment mechanism, which may incentivize profit-driven channel parties to close a payment channel with an old state, to their own advantage.

This paper introduces Daric, a payment channel with unlimited lifetime for Bitcoin that achieves optimal storage and bounded closure. Moreover, Daric implements a punishment mechanism and simultaneously avoids the methods other schemes commonly use to enable punishment: 1) state duplication which leads to exponential increase in the number of transactions with the number of applications on top of each other or 2) dedicated design of adaptor signatures which introduces compatibility issues with BLS or most post-quantum resistant digital signatures. We also formalise Daric and prove its security in the Universal Composability model.

Keywords: Bitcoin · scalability · Payment channel · Lightning network · Watchtower

1 Introduction

Due to its permissionless nature, Bitcoin suffers from poor transaction throughput [13,19,29]. Payment channel constitutes a promising solution, which allows two parties to perform several transactions without touching the blockchain except for creating and closing the channel. In more details, two parties create a payment channel by locking Bitcoins in a shared address. Then, they pay each other arbitrarily many times by exchanging authenticated off-chain transactions that spend the shared address and split the channel funds among parties. Each party can finally close the channel by publishing the last authenticated transactions on the blockchain. Payment channels can also be linked to form a payment channel network (PCN) where each payment can be routed via intermediaries.

Since each party's share of coins in a channel changes over time, one might attempt to close the channel with an old state to maximize her profit. Lightning Network [28]–the most popular payment channel network–adopts a punishment mechanism to prevent parties from acting dishonestly. In this network, upon authorizing a new state, channel parties exchange some revocation secrets to revoke the previous state. Then, if a party publishes a revoked state, her counter-party, who is supposed to be always online, uses the corresponding revocation secrets to take all the channel funds. Parties might also delegate the punishing job to a third party, called the *watchtower* [28].

Although elegantly designed, the Lightning Network has some shortcomings. Firstly, since channel parties must store all the revocation secrets, received from their counter-parties, their storage amount increases linearly with the number of channel updates. Moreover, to detect and punish the misbehaving party, the channel state is duplicated meaning each party has its own copy of the state. Then, when for adding an application (e.g. *Virtual channel* [9]) on top of the channel, parties have to split their channel into sub-channels, the state of each sub-channel is duplicated and it must propagate on both duplicates of the parent channel. Thus, state duplication causes the number of transactions to exponentially rise with the number of applications k built on top of each other [8].

Towards a different direction, the payment channel eltoo [18] introduces `ANYPREVOUT` [17] (also known as `NOINPUT`) as a new Bitcoin signature type to deploy the concept of versioning. This allows channel parties to override the current channel state by creating a state with a higher version number, which can be published upon fraud. So, channel parties in eltoo do not store any revocation secrets from old channel states. This simplifies the key management and offers more affordable watchtowers as the transaction with the highest version invalidates all previous states. Furthermore, if an honest party forgets about an update and publishes an outdated state, it does not result in the loss of funds.

However, eltoo is incentive incompatible because its lack of punishment might encourage a dishonest party to publish an old state; Either the other side corrects it or the dishonest party wins [27]. The only discouraging factor–the fee for publishing the old state–is also determined by the dishonest party. Thus, she can set it to the minimum possible value, i.e. few cents for some blockchains such as Bitcoin hard forks (e.g. Litecoin and Bitcoin Cash) and less than 1 USD for Bitcoin. Moreover, the transaction fee is independent of the channel *capacity* (i.e. the total funds in the channel). Therefore, even for payment channels with huge capacity of several BTCs (e.g., channels listed in [3]), the dishonest party's cost will be still below 1 USD (See Section 6.2 for detailed analysis). Additionally, enforcing a large transaction fee or restricting the channel capacity (proposed in [6]) might be unfavourable to the honest party.

Furthermore, a dishonest party in eltoo might publish multiple outdated states to delay the channel closure process [5]. Thus, eltoo fails to achieve *bounded closure*, i.e. honest party is not guaranteed that the channel closure completes within a bounded time. This compromises the security of time-based payments, e.g. *Hash-Time Lock Contract* (HTLC) (See Section 6.1 for further analysis).

Therefore, the main motivation of this paper is designing a Bitcoin payment channel that (1) provides optimal storage, (2) achieves bounded closure, (3) provides incentive compatibility, and (4) avoids state duplication.

1.1 Contributions

The contributions of the paper are as follows:

- We present a new Bitcoin payment channel, called Daric, which (i) is provably secure in the Universal Composability (UC) framework, (ii) achieves constant size storage for both channel parties and the watchtower, (iii) provides bounded closure, (iv) provides punishment mechanism and hence achieves incentive compatibility, (v) avoids state duplication without needing any particular property (e.g. adaptor signature properties) for the underlying digital signature, and (vi) attains unlimited lifetime, given that channel parties on average pay each other at most once per second. Table 1 compares Daric with other Bitcoin payment channels.

Table 1. Comparison of different payment channels with n channel updates and k recursive channel splitting.

Scheme	Party's	Watch.	Lifetime	Incent.	# of	Ada. Sig.	Bnd.
	St. Req.	St. Req.		Compat.	Txs	Avoid.	Cls.
Lightning† [28]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	Unlimited	Yes	$\mathcal{O}(2^k)$	Yes	Yes
Generalized† [8]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	Unlimited	Yes	$\mathcal{O}(1)$	No	Yes
Outpost [23]	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$	Limited	Yes	$\mathcal{O}(2^k)$	Yes	Yes
FPPW [26]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	Unlimited	Yes	$\mathcal{O}(1)$	No	Yes
Cerberus [11]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	Unlimited	Yes	$\mathcal{O}(2^k)$	Yes	Yes
Sleepy† [10]	$\mathcal{O}(n)$	N/A	Limited	Yes	$\mathcal{O}(2^k)$	Yes	Yes
eltoo [18]	$\mathcal{O}(1)$	$\mathcal{O}(1)$	Unlimited§	No	$\mathcal{O}(1)$	Yes	No
Daric (this work)	$\mathcal{O}(1)$	$\mathcal{O}(1)$	Unlimited§	Yes	$\mathcal{O}(1)$	Yes	Yes

†: If parties pre-generate n keys in a merkle tree, their storage requirements decrease to $\mathcal{O}(\log(n))$ but the channel lifetime becomes limited to n channel updates.

§: Given that the channel update rate is at most one update per second.

- We compare Daric and eltoo and show Daric is robust against an attack [5] to eltoo, that we also formalize in this paper. We further perform a cost benefit analysis to assess the attacker's revenue in practice. We also show (i) Daric provides a higher deterrent effect against profit-driven attackers than eltoo and (ii) unlike eltoo, Daric's deterrent effect is flexible.
- We compare Daric, eltoo, Lightning, Generalized, Sleepy, Cerberus, FPPW and Outpost channels with respect to the amount of data that is published on the blockchain in different channel closure scenarios (See Table 3). We show that Daric in the dishonest closure scenario outperforms Lightning with

at least 1 HTLC output as well as all other schemes. In the non-collaborative closure scenario, Daric outperforms Lightning with at least 7 HTLC outputs as well as Generalized, eltoo and FPPW. Moreover, we compute the number of operations required for each channel update and show that (i) Unlike Lightning, Daric values are independent of the number of HTLC outputs m and (ii) Daric is comparable with other schemes (see Table 3).

1.2 Related works

The first payment channel [30] is unidirectional and hence cannot continue working if the payer's balance is depleted. DMC [19] uses decrementing timelocks but suffers from limited channel lifetime. For Lightning channel [28], an existing state is replaced upon authorizing a new state and then revoking the previous one where each party has his own version of transactions. *Generalized channel* [8] uses *adaptor signatures* to distinguish the publisher of a revoked state from her counter-party. In this way, Generalized channel avoids state duplication.

Outpost [23], Cerberus [11] and FPPW [26] are other payment channels, which focus on improving their watchtower properties. Sleepy channel [10] is a bi-directional payment channel without watchtowers where parties can go offline for prolonged periods of time. Towards a different direction, Teechain [25] requires channel parties to possess Trusted Execution Environment (TEE).

2 Background and Notations

2.1 Outputs and Transactions

Throughout this work, we define different attribute tuples. Let U be a tuple of multiple attributes including the attribute attr. To refer to this attribute, we use $U.\mathsf{attr}$. Our focus in this work is on Bitcoin or any other blockchains with *Unspent Transaction Output* (UTXO) model. In this model, units of value–which we call coins–are held in *outputs*. Each output contains a condition that needs to be fulfilled to spend the output. Satisfying a condition might require one or multiple parties' signatures. Such a condition contains public keys of all the involved parties and we say those parties own the output. A condition might also have several subconditions, one of which must be satisfied to spend the output.

A transaction changes ownership of coins, meaning it takes a list of existing outputs and transfers their coins to a list of new outputs. To distinct between these two lists, we refer to the list of existing outputs as *inputs*. A transaction $\mathtt{TX}$ is formally defined as the tuple $(\mathsf{txid}, \mathsf{Input}, \mathsf{nLT}, \mathsf{Output}, \mathsf{Witness})$. The identifier $\mathtt{TX}.\mathsf{txid} \in \{0,1\}^*$ is computed as $\mathtt{TX}.\mathsf{txid} := \mathcal{H}([\mathtt{TX}])$, where $[\mathtt{TX}]$ is called the *body* of the transaction defined as $[\mathtt{TX}] := (\mathtt{TX}.\mathsf{Input}, \mathtt{TX}.\mathsf{nLT}, \mathtt{TX}.\mathsf{Output})$ and $\mathcal{H}$ is a hash function, which is modeled as a random oracle. The attribute $\mathtt{TX}.\mathsf{nLT}$ denotes the value of the parameter $nLockTime$, where $\mathtt{TX}$ is invalid unless its $nLockTime$ is in the past. The attribute $\mathtt{TX}.\mathsf{Input}$ is a list of identifiers for all inputs of $\mathtt{TX}$. The attribute $\mathtt{TX}.\mathsf{Output}$ is a list of new outputs. The attribute

$\mathtt{TX}.\mathsf{Witness}$ is a list where its i^{th} element authorizes spending the output that is taken as the i^{th} input of $\mathtt{TX}$. We also use $\overline{[\mathtt{TX}]}$ and $\overline{\mathtt{TX}}$ to denote ($\mathtt{TX}.\mathsf{nLT}$, $\mathtt{TX}.\mathsf{Output}$) and ($\mathtt{TX}.\mathsf{nLT}, \mathtt{TX}.\mathsf{Output}, \mathtt{TX}.\mathsf{Witness}$), respectively.

Floating Transactions Each signature in a Bitcoin transaction contains a flag, called SIGHASH, which specifies which part of the transaction has been signed. Typically, signatures are of type SIGHASH_ALL, meaning the signature authorizes all inputs (i.e. references to previous outputs) and outputs. The SIGHASH of type ANYPREVOUT indicates that the signature does not authorize the inputs. This allows the signer to refer to any arbitrary UTXO whose condition is met by the transaction witness data. Such a transaction is called a *floating* transaction.

Timelocks The relative timelock of T rounds (a round in our paper is considered the same as the round in [8,10]) in an output condition is denoted by T^{+} and means the output cannot be spent unless at least T rounds passed since the output was recorded on the blockchain. The absolute timelock of i in an output condition is shown by $i^{\geq}$ and means the output cannot be spent unless the $nLockTime$ parameter in the spending transaction is equal to or greater than i. Since a transaction may only be recorded on the blockchain if its $nLockTime$ is in the past, $i^{\geq}$ in an output condition ensures the output cannot be spent unless i is expired (i.e. i is in the past). Table 2 summarizes the notations.

Table 2. Summary of notations

Notation	Description
$\mathtt{TX}$	Transaction $\mathtt{TX} = (\mathsf{txid}, \mathsf{Input}, \mathsf{nLT}, \mathsf{Output}, \mathsf{Witness})$
$\overline{\mathtt{TX}}$	Tuple ($\mathtt{TX}.\mathsf{nLT}, \mathtt{TX}.\mathsf{Output}, \mathtt{TX}.\mathsf{Witness}$)
$[\mathtt{TX}]$	Tuple ($\mathtt{TX}.\mathsf{Input}, \mathtt{TX}.\mathsf{nLT}, \mathtt{TX}.\mathsf{Output}$)
$\overline{[\mathtt{TX}]}$	Tuple ($\mathtt{TX}.\mathsf{nLT}, \mathtt{TX}.\mathsf{Output}$)
T^{+}	The relative timelock of T rounds
$i^{\geq}$	The absolute timelock of i

Representation of Transaction Flows We use charts to illustrate transaction flows. As Fig. 1 shows, doubled edge and single edge rectangles represent published and unpublished transactions, respectively. Since the output of $\mathtt{TX}$ with the value of $a + b$ has two subconditions, it is denoted by a diamond shape with two arrows. One of the subconditions can be fulfilled by both A and B and is relatively timelocked by T rounds and another subcondition can be fulfilled by C and contains an absolute timelock of i. Dotted arrow to $\mathtt{TX}''$ shows it is a floating transaction whose signature matches the public key pk_C. This transaction is denoted by $\overline{\mathtt{TX}''}$ to emphasize that since it is a floating transaction, its input is unspecified and can be any output with matching condition. The $nLockTime$ parameter for $\texttt{TX}$ and $\texttt{TX}'$ is 0, so it is not shown inside these transactions.

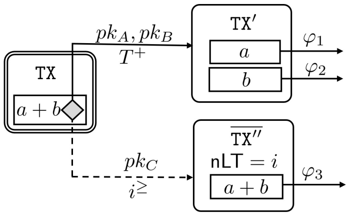

Fig. 1. A sample transaction flow.

2.2 Payment Channel

A payment channel between two parties Alice (or A) and Bob (or B) allows them to perform a number of transactions without publishing every single transaction on the blockchain. To create a channel with channel capacity of $a + b$ coins, A and B respectively deposit a and b coins into a joint address that is controlled by both parties. Parties can update their balance in the channel off-chain by agreeing on a new way to split the channel funds. Each party can close the channel at any time by enforcing the latest channel state on the blockchain where dispute between channel parties are resolved on the blockchain.

Lightning Channels [28] To create a Lightning channel, A and B publish a *funding* transaction to respectively deposit a and b coins into a joint address. Each party also has its own copy of an off-chain transaction, called the *commit* transaction, that spends the joint address and splits the channel funds between A and B accordingly, i.e. each commit transaction has two outputs: one output holding a coins owned by A and the other output holding b coins owned by B. Each party can publish the commit transaction to close the channel, but parties typically create new commit transactions to update their shares in the channel.

As one may submit an intermediate state (which is already replaced by a later state) to the blockchain, the channel parties will need to punish such misbehaviours. Thus, after each channel update, parties revoke their previous state by exchanging two *revocation* transactions (one version for each party) that take the output of the old commit transactions and give the balance of the dishonest party to the honest party. However, the honest party (e.g. A) must publish the revocation transaction before the dishonest party (e.g. B) can claim his balance. So, to give precedence to the revocation transaction, B has to wait for a relative timelock of T rounds (in practice, one day) before he can claim his output. This gives some time to A to publish the revocation transaction.

eltoo [18] An eltoo channel is created like a Lightning channel, but each state is represented by two transactions: (i) the *update* transaction and (ii) the *settlement* transaction, where both parties have the same version of these two transactions. Each update transaction is a floating transaction that transfers all the channel funds to a new joint address. The update transaction's output can be spent by its corresponding settlement transaction, which splits the channel funds among parties. If A submits an old update transaction, she has to wait for a relative timelock of T rounds before she can publish the corresponding settlement transaction. It gives some time to B to publish the latest update transaction (which is a floating transaction) and override the already published update transaction.

3 Solution Overview

To provide a high level overview of our solution, we start by reviewing the limitations of the Lightning channel and then gradually present our work.

Revocation Per State Parties' and their watchtower's storage in a Lightning channel increases over time as they should store some revocation-related data for each revoked state. Our main idea to reduce their storage is transforming the revocation transactions into floating transactions. Thereby, participants only need to store the latest revocation transaction with the largest version number and use it upon fraud. However, for a Lightning channel, (i) the monetary value of each revocation transaction typically differs from one state to another, and (ii) each commit transaction might have multiple HTLC outputs and hence the number of revocation transactions might also differ from one state to another. So, since revocation transactions of different states differ in value and number, it is infeasible to replace them all with the latest revocation transactions.

Therefore, our first modification is following the *punish-then-split* mechanism, introduced in [8]. According to this mechanism, the commit transaction sends the channel funds to a new joint output, which is controlled by both parties. Output of this commit transaction can be spent by its corresponding split transaction after T rounds where outputs of the split transaction split the channel funds between A and B. If A publishes a revoked commit transaction, B must spend its output within T rounds with the corresponding revocation transaction. This revocation transaction gives all the channel funds to B. Fig. 2 depicts the transaction flows for this channel where each party stores a single revocation transaction with fixed monetary value (i.e. $a + b$ coins) per state.

Revocation Per Channel In the scheme, depicted in Fig. 2, channel parties need to store a revocation transaction for each revoked state. Therefore, storage requirements of channel parties (or their watchtower) increase with each channel update. To solve this issue, we transform revocation transactions into floating transactions, i.e. the signatures in a revocation transaction, held by A, are of type `ANYPREVOUT` and meet the output condition of all commit transactions, held by B, and vice versa. It allows parties to only store the last revocation transaction.

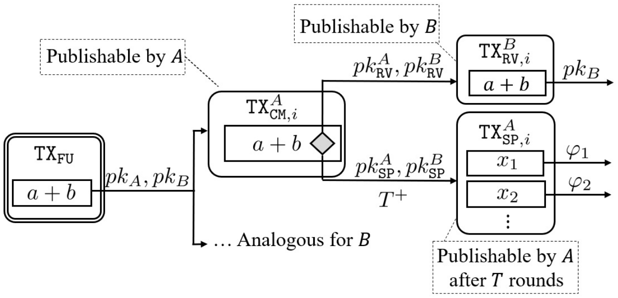
Publishable by B
Publishable by A
$\mathtt{TX}^B_{\mathtt{RV},i}$
$a+b$
pk_B
$pk^A_{\mathtt{RV}}, pk^B_{\mathtt{RV}}$
$\mathtt{TX}^A_{\mathtt{CM},i}$
$a+b$
$\mathtt{TX}^A_{\mathtt{SP},i}$
φ_1
x_1
φ_2
x_2
$\vdots$
$pk^A_{\mathtt{SP}}, pk^B_{\mathtt{SP}}$
T^+
$\mathtt{TX}_{\mathtt{FU}}$
$a+b$
pk_A, pk_B
… Analogous for B
Publishable by A
after T rounds

Fig. 2. Transaction flows for a Lightning channel with punish-then-split mechanism where $\mathtt{TX}_{\mathtt{FU}}$ denotes the funding transaction and $\mathtt{TX}^A_{\mathtt{CM},i}$, $\mathtt{TX}^A_{\mathtt{SP},i}$ and $\mathtt{TX}^A_{\mathtt{RV},i}$ (or respectively $\mathtt{TX}^B_{\mathtt{CM},i}$, $\mathtt{TX}^B_{\mathtt{SP},i}$ and $\mathtt{TX}^B_{\mathtt{RV},i}$) denote the commit, split and revocation transactions held by A (or respectively held by B) for state i.

Avoiding State Duplication Since each state in the introduced scheme contains two split transactions (one for each party), the scheme suffers from state duplication. To avoid this, we transform split transactions into floating transactions. Then, each state contains one split transaction (held by both parties), which spends any of two commit transactions of that state.

State Ordering Since split and revocation transactions are floating, it must be guaranteed that the latest commit transaction cannot be spent using any split or revocation transaction from previous states. Otherwise, the honest party, who has published the latest commit transaction, might lose some funds in the channel. To achieve this requirement, we repurpose [18] the *nLockTime* parameter of split and revocation transactions to store the *state number*: the number of times the channel has been updated to date. Furthermore, we add the state number to the output condition of each commit transaction as an absolute timelock. Then, since the absolute timelock in output condition of the last commit transaction would be larger than the *nLockTime* parameter in any split or revocation transaction from previous states, the mentioned requirement is met.

Putting Pieces Together The transaction flow for state i of Daric is depicted in Fig. 3. Let channel be in state n. To close the channel, each party (e.g. A) can publish the latest commit transaction (e.g. $\mathtt{TX}^A_{\mathtt{CM},n}$), wait for T rounds and finally publish the latest split transaction $\mathtt{TX}_{\mathtt{SP},n}$. There is no revocation transaction for the latest state. If party B publishes a revoked commit transaction (i.e. $\mathtt{TX}^B_{\mathtt{CM},i}$ with $i < n$), then party A instantly publishes the latest revocation transaction $\mathtt{TX}^A_{\mathtt{RV},n-1}$ to take all the channel funds.

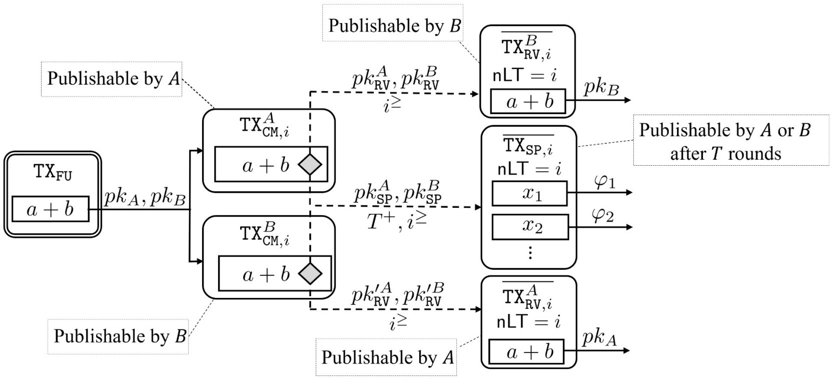
Publishable by B
$\overline{\mathtt{TX}^{B}_{\mathtt{RV},i}}$
nLT $= i$
$a+b$
pk_B
Publishable by A
$pk^{A}_{\mathtt{RV}}, pk^{B}_{\mathtt{RV}}$
$i^{\geq}$
$\mathtt{TX}^{A}_{\mathtt{CM},i}$
$a+b$
Publishable by A or B after T rounds
$\overline{\mathtt{TX}_{\mathtt{SP},i}}$
nLT $= i$
x_1
φ_1
x_2
φ_2
$\vdots$
$\mathtt{TX}_{\mathtt{FU}}$
$a+b$
pk_A, pk_B
$pk^{A}_{\mathtt{SP}}, pk^{B}_{\mathtt{SP}}$
$T^{+}, i^{\geq}$
$\mathtt{TX}^{B}_{\mathtt{CM},i}$
$a+b$
$pk'^{A}_{\mathtt{RV}}, pk'^{B}_{\mathtt{RV}}$
$i^{\geq}$
$\overline{\mathtt{TX}^{A}_{\mathtt{RV},i}}$
nLT $= i$
$a+b$
pk_A
Publishable by B
Publishable by A

Fig. 3. Transaction flows for state i of a Daric channel.

4 Protocol Description

This section presents our protocol using the transaction flows depicted in Fig. 3. The lifetime of a Daric channel can be divided into 4 phases including create, update, close, and punish. We introduce these phases through sections 4.1 to 4.4. Appendix D provides the formal description of the protocol.

4.1 Create

To create the channel, A and B sign and publish the funding transaction $\mathtt{TX}_{\mathtt{FU}}$ on the blockchain. By publishing this transaction, A and B fund the channel with a and b coins, respectively, but since output of the funding transaction can only be spent if both parties agree, one party might become unresponsive to raise a hostage situation. To avoid this, before signing the funding transaction, parties commit to the initial channel state, i.e. state 0, by exchanging signatures for the corresponding commit and split transactions. Let us explain different steps of the channel creation phase in more details.

Step 1: At the first step, A and B send their funding sources (i.e. $txid_A$ and $txid_B$) to each other. This enables them to create the body of the funding transaction $[\mathtt{TX}_{\mathtt{FU}}]$. **Step 2:** Having the transaction identifier of $\mathtt{TX}_{\mathtt{FU}}$, parties create the body of the commit transactions, i.e. $[\mathtt{TX}^{A}_{\mathtt{CM},0}]$ and $[\mathtt{TX}^{B}_{\mathtt{CM},0}]$. **Steps 3:** Parties exchange the required signatures (with `SIGHASH` of type `ANYPREVOUT`) to create the floating transaction $\overline{\mathtt{TX}_{\mathtt{SP},0}}$. This floating transaction could take output of $\mathtt{TX}^{A}_{\mathtt{CM},0}$ or $\mathtt{TX}^{B}_{\mathtt{CM},0}$ as its input. **Step 4:** Parties exchange the required signatures to create the commit transactions $\mathtt{TX}^{A}_{\mathtt{CM},0}$ and $\mathtt{TX}^{B}_{\mathtt{CM},0}$. **Step 5:** Parties exchange the required signatures to create the funding transactions $\mathtt{TX}_{\mathtt{FU}}$. **Step 6:** Parties publish the funding transaction on the blockchain.

The absolute timelock in output script of commit transactions and correspondingly the *nLockTime* parameter in the split transaction must be in the past. Otherwise, parties have to wait to publish such transactions. As explained in Section 3, the timelock is set to the state number and hence its value increases with each channel update. Absolute timelocks lower than 500,000,000 specify the block number after which the transaction can be included in a block. According to the value of the current block height, if we set the initial timelock to the first state number, i.e. 0, the channel can be updated around 700,000 times. However, absolute timelocks equal to or larger than 500,000,000 specify the UNIX timestamp after which the transaction will be valid. According to the value of the current timestamp, if we set the initial timelock (and correspondingly *nLockTime* parameter) to 500,000,000, the channel can be updated around 1 billion times [18]. Moreover, the current timestamp increases one unit per second, meaning if the average rate of the channel update is up to once per second, the channel can be updated an infinite number of times.

4.2 Update

Let the channel be in state $i \geq 0$ and channel parties decide to update it to state $i+1$. The update process is performed in two sub-phases. The first sub-phase is similar to steps 2 to 4 of channel creation phase where channel parties create two new commit transactions $\mathtt{TX}^A_{\mathtt{CM},i+1}$ and $\mathtt{TX}^B_{\mathtt{CM},i+1}$ as well as a new split transaction $\overline{\mathtt{TX}_{\mathtt{SP},i+1}}$ for the new state. In the second sub-phase, channel parties revoke the state i by signing two revocation transactions $\overline{\mathtt{TX}^A_{\mathtt{RV},i}}$ and $\overline{\mathtt{TX}^B_{\mathtt{RV},i}}$. The revocation transaction $\overline{\mathtt{TX}^A_{\mathtt{RV},i}}$ (or respectively $\overline{\mathtt{TX}^B_{\mathtt{RV},i}}$) contains no input yet and can spend output of any commit transaction $\overline{\mathtt{TX}^B_{\mathtt{CM},j}}$ (or respectively $\overline{\mathtt{TX}^A_{\mathtt{CM},j}}$) with $j \leq i$. With each channel update, the state number and hence the timelock value in the output condition of each commit transaction and *nLockTime* in split and revocation transactions increase by one unit. Let us explain different steps of the channel update phase in more details.

Step 1: Parties create the body of the commit transactions, i.e. $[\mathtt{TX}^A_{\mathtt{CM},i+1}]$ and $[\mathtt{TX}^B_{\mathtt{CM},i+1}]$. **Steps 2:** Parties exchange the required signatures (with `SIGHASH` of type `ANYPREVOUT`) to create the floating transaction $\overline{\mathtt{TX}_{\mathtt{SP},i+1}}$. This floating transaction takes output of $\mathtt{TX}^A_{\mathtt{CM},i+1}$ or $\mathtt{TX}^B_{\mathtt{CM},i+1}$ as its input. **Step 3:** Parties exchange the required signatures to create the commit transactions $\mathtt{TX}^A_{\mathtt{CM},i+1}$ and $\mathtt{TX}^B_{\mathtt{CM},i+1}$. **Step 4:** Parties exchange the required signatures (with `SIGHASH` of type `ANYPREVOUT`) to create the floating transactions $\overline{\mathtt{TX}^A_{\mathtt{RV},i}}$ and $\overline{\mathtt{TX}^B_{\mathtt{RV},i}}$.

One of the parties might misbehave by receiving the signature on the split or commit transactions in steps 2 or 3 (or respectively by receiving the signature on the revocation transaction in step 4) but avoiding signing the corresponding transaction for the other party. In such situations, the honest party non-collaboratively closes the channel with the latest valid channel state, i.e. state i (or respectively state $i+1$). More technical details can be found in Appendix D.

4.3 Close

Assume while the channel between A and B is in state n, they decide to collaboratively close it. To do so, A and B exchange signatures for a new transaction, called modified split transaction $\mathtt{TX}_{\overline{\mathtt{SP}}}$, and publish it on the blockchain. This transaction takes the funding transaction's output as its input and splits the channel funds among channel parties. If one of the channel parties, e.g. party B, becomes unresponsive, its counter-party A can still non-collaboratively close the channel by publishing $\mathtt{TX}^{A}_{\mathtt{CM},n}$, adding the output of $\mathtt{TX}^{A}_{\mathtt{CM},n}$ as an input to $\overline{\mathtt{TX}_{\mathtt{SP},n}}$ to transform it into $\mathtt{TX}_{\mathtt{SP},n}$, and finally publishing $\mathtt{TX}_{\mathtt{SP},n}$ after T rounds.

4.4 Punish

Let the channel be in state n. If a dishonest channel party, let's say A, publishes an old commit transaction $\mathtt{TX}^{A}_{\mathtt{CM},i}$ with $i < n$ on the ledger, party B adds the output of $\mathtt{TX}^{A}_{\mathtt{CM},i}$ as an input to $\overline{\mathtt{TX}^{B}_{\mathtt{RV},n-1}}$ in order to transform it into $\mathtt{TX}^{B}_{\mathtt{RV},n-1}$ and instantly publishes $\mathtt{TX}^{B}_{\mathtt{RV},n-1}$ on the blockchain.

5 Security Analysis

In this section, we firstly provide some payment channel notations as well as our security model, which follow previous works on *layer-2* solutions [8,22,20,21]. Then, we present desired properties of a payment channel and an ideal functionality $\mathcal{F}$ that attains those properties. Finally, we show Daric protocol is a realization of the ideal functionality $\mathcal{F}$ and hence achieves its desired properties.

5.1 Notation and Security Model

We use an extended version of the *universal composability* framework [15] to formally model the security of our construction. This extended version [16], called Global Universal Composability framework (GUC), supports a global setup. To simplify our model, we assume that the communication network is synchronous, meaning that the protocol is executed through multiple rounds and parties in the protocol are connected to each other via an authenticated communication channel which guarantees 1-round delivery. Transactions are recorded by a global ledger $\mathcal{L}(\Delta, \Sigma)$, where Σ is a signature scheme used by the blockchain and Δ is an upper bound on the blockchain delay: the number of rounds it takes a transaction to be accepted by the ledger. Appendix C provides more details on our security model.

A payment channel γ is defined as an attribute tuple $\gamma := (\mathsf{id}, \mathsf{users}, \mathsf{cash}, \mathsf{st}, \mathsf{sn}, \mathsf{flag}, \mathsf{st}')$, where $\gamma.\mathsf{id} \in \{0,1\}^*$ defines the channel identifier, $\gamma.\mathsf{users}$ represents the identities of the channel users, $\gamma.\mathsf{cash} \in \mathbb{R}^{\geq 0}$ is the total funds locked in the channel, $\gamma.\mathsf{st} := (\theta_1, \ldots, \theta_l)$ is a list of l outputs defining the channel state after the last complete channel update and $\gamma.\mathsf{sn}$ is the state number. The flag $\gamma.\mathsf{flag} \in \{1, 2\}$ and the state $\gamma.\mathsf{st}'$ will be explained below.

The initial value of $\gamma.\mathsf{flag}$ and $\gamma.\mathsf{st}'$ are 1 and $\perp$, respectively. Assume that the channel has been updated $n \geq 0$ times and the channel state after the n^{th} update is st and hence we have $\gamma.\mathsf{st} = st$. Now, assume that parties start the update process to update the state of the channel from state st to st'. From a particular point in the channel update process onward, at least one of the parties has sufficient data to enforce the new state st' on the blockchain when parties have not completely revoked the state $\gamma.\mathsf{st}$ yet. The flag $\gamma.\mathsf{flag}$ is set to 2 to identify such occasions and $\gamma.\mathsf{st}'$ is set to st' to maintain the new state. Thus, when $\gamma.\mathsf{flag} = 2$, the channel might be finalized with either $\gamma.\mathsf{st}$ or $\gamma.\mathsf{st}'$. At the end of the channel update process, once the state st was revoked by both parties, $\gamma.\mathsf{st}$ and $\gamma.\mathsf{st}'$ are set to st' and $\perp$, respectively, and $\gamma.\mathsf{flag}$ is set to 1.

5.2 Ideal Functionality

This section closely follows [8] to introduce desired security and efficiency properties of a payment channel as following:

- **Consensus on creation:** A channel γ is created only if both channel parties in the set $\gamma.\mathsf{users}$ agree to create it.
- **Consensus on update:** A channel γ is updated only if both channel parties in the set $\gamma.\mathsf{users}$ agree to update it. Also, parties reach agreement on update acceptance or rejection within a bounded number of rounds (the bound might depend on the ledger delay Δ).
- **Bounded closure with punish:** An honest user $P \in \gamma.\mathsf{users}$ has the assurance that within a bounded number of rounds (the bound might depend on the ledger delay Δ), she can finalize the channel state on the ledger either by enforcing a state that gives her $\gamma.\mathsf{cash}$ coins, or by enforcing $\gamma.\mathsf{st}$ if $\gamma.\mathsf{flag} = 1$ or by enforcing either $\gamma.\mathsf{st}$ $\gamma.\mathsf{st}'$ otherwise.
- **Optimistic update:** If both parties in $\gamma.\mathsf{users}$ are honest, the channel update completes with no ledger interaction.

Appendix A introduces an ideal functionality $\mathcal{F}$ that achieves these properties. Theorem 1 shows Daric protocol, denoted by π, is a realization of $\mathcal{F}$ and hence achieves its desired properties. It follows from 14 Lemmas. Due to space limits, we refer readers to Appendix G for the full security proof.

Theorem 1. *Let Σ be an $\mathsf{EUF-CMA}$ secure signature scheme. Then, for any ledger delay $\Delta \in \mathbb{N}$, the protocol π UC-realizes the ideal functionality $\mathcal{F}(T)$ with any $T > \Delta$.*

We formally define π in Appendix D and then provide a simulator $\mathcal{S}$ in Appendix G where $\mathcal{S}$ has interaction with the ideal functionality $\mathcal{F}$ and $\mathcal{L}$. The simulator simulates content and timing of all messages of the honest party to the adversary and also translates any message from the adversary into a message to the ideal functionality, such that an indistinguishable execution of the protocol in the ideal world is emulated. Thus, our protocol would be as secure as the ideal functionality $\mathcal{F}$. We also prove for any action that causes the ideal functionality to output Error with non-negligible probability, the simulator constructs a reduction against the existential unforgeability of the underlying signature scheme Σ with non-negligible success probability, which contradicts with our assumption regarding the security of Σ. This proves our protocol provides the desirable properties of $\mathcal{F}$.

6 Daric Versus Eltoo

In section 6.1, we present an attack to eltoo whose main purpose is to postpone the channel closure. We show this attack is practically profitable when applied to eltoo but it cannot be applied to Daric. In section 6.2, we analyze Daric and eltoo to compare their robustness against profit-driven attackers. We use the statistical data derived from the Lightning network to enable such an analysis.

6.1 HTLC Security

This section presents an attack against HTLC security in eltoo (previously informally discussed in [5]) and analyzes the attacker's revenue. Let the adversary represent two nodes on the PCN: node M_1 and node M_2. Assume that the adversary has established N channels from M_1 to victim nodes $V_1, \ldots, V_N$ and N channels from victim nodes to M_2. The channel between M_1 and V_i is denoted with γ_i. The adversary performs N simultaneous HTLC payments from M_1 to M_2 through $V_1, \ldots, V_N$. Let the payment value for all HTLCs be A coins and the timelock for all these payments for M_1's channels be T. Assume that M_2 accepts the payments and provides the required secrets for all HTLC payments and hence M_2 is paid $N \cdot A$ coins in total. Then, victims provide the secrets to the node M_1. However, M_1 does not update her channels with victims. Therefore, victims attempt to claim all HTLCs on-chain. To prevent victims from closing their channels in time, M_1 takes the following steps:

1. Submit a valid *Delay* transaction $\mathtt{TX}_{\mathtt{De}}$ with $N+1$ inputs and $N+1$ outputs where the i^{th} input-output pair corresponds with an outdated state of the channel γ_i and the last input-output pair adds further funds to be used as the transaction fee, which is set to any value larger than A.
2. If $\mathtt{TX}_{\mathtt{De}}$ is published and the timelock T is still unexpired, go to step 1.
3. Once the timelock T is expired, submit the latest channel state for all channels and claim their HTLC outputs.

In the explained scenario, to replace the already submitted transaction $\mathtt{TX}_{\mathtt{De}}$ with the latest state of the channel γ_i, V_i has to set a transaction fee that is larger than the total absolute transaction fee of $\mathtt{TX}_{\mathtt{De}}$ [31]. But since the transaction fee for $\mathtt{TX}_{\mathtt{De}}$ is larger than A, V_i will be unwilling to pay such a transaction fee.

Once the HTLC timelock is expired and the latest channel state is added to the ledger, there will be a race between M_1 and each victim to claim the HTLC output. The adversary will have a better chance to win the race if she has a better network connection with a higher number of nodes.

Now we perform a cost benefit analysis to determine if the attack is profitable to the attacker. For a fixed value of A, with setting N to the largest possible value, the adversary 1) reduces the fee per channel for each delay transaction and 2) reduces the pace at which outdated states are added to the blockchain. A Bitcoin transaction can contain up to 100,000 VBytes (where each VByte equals four weight units) and each input-output pair contains 222 bytes of witness data and 84 bytes of non-witness data (See Appendix H.4 for more details). Therefore, $\mathtt{TX}_{\mathtt{De}}$ can cover up to around $\frac{100{,}000}{0.25\times 222+84} \approx 715$ eltoo channels. The minimum possible fee rate is 1 Satoshi per VByte. Thus, if A is set to 100,000 Satoshi, the total fee for each delay transaction would be 100,000 Satoshi.

At the time of writing this paper (in April 2022), the average transaction fee is quite low and hence transactions with the minimum fee rate are added to the blockchain in 30 minutes. It means if HTLC timelocks are set to 3 days, 144 delay transactions are published before timelocks getting expired. In other words, the adversary pays $144A$ as transaction fee to earn up to $715A$. In more congested times, it might take several hours for a transaction with minimum fee rate to be added to the blockchain. Thus, the attack could be even more profitable to the attacker. This attack is inapplicable to Daric because once the attacker publishes an old commit transaction, the only valid transactions are the revocation transactions held by her counter-party.

6.2 Punishment Mechanism

Prior to providing a formal analysis, we provide intuitions as follows. The only cost for a dishonest party in eltoo is the fee for publishing the old state, which could be (i) less than 1 USD for Bitcoin and (ii) independent of the channel capacity. However, given that the balance of each party in a Daric channel cannot be less than 1% of the channel capacity (which is currently deployed in the Lightning network), the minimum amount that a dishonest party might lose would have the following properties: (i) It is proportional to the channel capacity, (ii) Its value (around 20 USD on average in the Lightning network in April 2022) is typically significantly larger than the transaction fee and (iii) It is easily raised by increasing the minimum possible balance of each channel party from 1% of the channel capacity to a higher proportion. Therefore, Daric's deterrent effect against profit-driven attackers is higher and more flexible than that of eltoo.

Now, we perform a more formal comparison between eltoo and Daric. We assume the channel party either stays online or employs a watchtower that is fair w.r.t the hiring party [26] (i.e. the watchtower guarantees its client's funds in the channel). For the former case, let p denote the probability that the honest channel party successfully reacts upon fraud, i.e. $1-p$ is the probability that the honest party, due to crash failures or DoS attacks, fails to react. We show that (i) to discourage attacks by profit-driven parties, p for eltoo must be more significant than that of Daric, and (ii) unlike Daric, increase in the channel capacity in eltoo channels raises the minimum value of p that is required to prevent fraud. However, achieving large values of p (e.g. 0.9999) could be difficult for ordinary users. This indicates eltoo needs a way to punish profit-driven attackers.

To monitor a channel, the watchtower's collateral equals the channel capacity [26,11]. Let C denote the total capacity of Bitcoin payment channel network and C_W denote the total capital that fair watchtowers have spent to watch their clients' channels. Then, the probability that a randomly selected payment channel is monitored by a fair watchtower is roughly computed as $\frac{C_W}{C}$.

Assume that a dishonest party $\mathcal{A}$ creates an eltoo channel with channel capacity of $C_{\mathcal{A}}$ coins, where the initial balance of $\mathcal{A}$ and her counter-party are $C_{\mathcal{A}}$ and 0, respectively. For now, we assume that parties know if their counter-parties are using a fair watchtower. We will relax this assumption later. If the channel is being monitored by a fair watchtower, $\mathcal{A}$ continues using the channel in an honest way. Otherwise, she sends all her balance to her counter-party in exchange for some products or services and then submits the initial channel state to the blockchain. In such a case, with probability of $1-p$ and p, $\mathcal{A}$'s revenue and her loss would be $C_{\mathcal{A}} - f$ and f, respectively, where f denotes the transaction fee. Thus, $\mathcal{A}$ is discouraged to attack iff:

$$(C_{\mathcal{A}} - f)(1-p) - f \cdot p < 0 \Leftrightarrow p > 1 - \frac{f}{C_{\mathcal{A}}}.$$

For a Daric channel, $\mathcal{A}$ is discouraged to attack iff:

$$0.99 \cdot C_{\mathcal{A}} \cdot (1-p) - 0.01 \cdot C_{\mathcal{A}} \cdot p < 0 \Leftrightarrow p > 0.99.$$

The threshold value for eltoo is typically more significant than that of Daric. At the time of writing this paper, the average values of f for a transaction and $C_{\mathcal{A}}$ for a Lightning channel are around 0.000055 BTC and 0.04 BTC, respectively, leading to $1 - \frac{f}{C_{\mathcal{A}}} \approx 0.999$. But the adversary can practically set f to the lowest possible value (i.e. 1 Satoshi per VByte) leading to $f \approx 0.0000021$[1] BTC and $1 - \frac{f}{C_{\mathcal{A}}} \approx 0.9999$ for eltoo. Therefore, (i) to discourage attacks, the honest party would require to meet a higher p in eltoo than in Daric, (ii) the threshold for eltoo depends on the channel capacity, and (iii) the threshold for Daric can simply decrease from 0.99 to lower values.

In the above analysis, we assumed that $\mathcal{A}$ knows whether her counter-party is hiring any fair watchtower. Considering the opposite case, the probability that the channel is not being monitored by any fair watchtower and the honest party fails to react upon fraud would be $p_0 := (1 - \frac{C_W}{C})(1-p)$. Thus, with probability of p_0 and $1 - p_0$, $\mathcal{A}$'s revenue and her loss in an eltoo channel would be $C_{\mathcal{A}} - f$ and f, respectively. Thus, $\mathcal{A}$ is discouraged to attack iff:

$$(C_{\mathcal{A}} - f) \cdot p_0 - f \cdot (1 - p_0) < 0 \Leftrightarrow p > 1 - \frac{\frac{f}{C_{\mathcal{A}}}}{1 - \frac{C_W}{C}}.$$

Similarly, for a Daric channel we have:

$$0.99 \cdot C_{\mathcal{A}} \cdot p_0 - 0.01 \cdot C_{\mathcal{A}} \cdot (1 - p_0) < 0 \Leftrightarrow p > 1 - \frac{0.01}{1 - \frac{C_W}{C}}.$$

As explained earlier, the threshold value for eltoo depends on $C_{\mathcal{A}}$ and is typically more significant than that of Daric.

[1] Each update transaction in eltoo contains 332 byte of witness data and 125 bytes of non-witness data leading to 208 VBytes. See Appendix H.4 for more details

7 Performance Analysis

Table 3 shows the total number of weight units of transactions, published on the blockchain for different payment channels in different channel closure scenarios. Since the weight units of a transaction directly impacts its fee, we use this parameter to compare different schemes. Payment channels perform similarly in the collaborative channel closure, so we do not consider this scenario in our analysis. Since the funding transaction is the same in all schemes, we do not involve it in our comparison results either. To do a consistent comparison, we assume that each transaction output is either P2WSH[2] or P2WPKH[3], each public key and signature are respectively 33 bytes and 73 bytes, shared outputs are implemented using the OP_CHECKMULTSIG opcode (rather than using multi-party signing), and each state contains m HTLC outputs with $0 \leq m \leq 966$ [2] where each party is the payer for $\frac{m}{2}$ HTLC outputs and the payee for the rest.

Once a dishonest party in a Lightning channel publishes a revoked commit transaction, $m+1$ revoked outputs are created. For simplicity, we assume that the victim claims all the revoked outputs through one transaction. Cerberus [11], Sleepy [10] and Outpost [23] have not explained ways HTLC is added to these schemes and discussing it is out of the scope of this paper, so Table 3 contains their figures with $m = 0$.

As Table 3 shows, in the dishonest closure scenario, (1) the weight units for Lightning and eltoo increase linearly with the number of HTLC outputs m compared to Daric, Generalized and FPPW and (2) Daric (with weight unit equal to 1239) is more cost effective than other schemes with $m \geq 1$. In the non-collaborative closure scenario with $m \neq 0$, Daric outperforms Generalized, eltoo and FPPW channels with any value of m and Lightning channel with $m > 6$.

Table 3 also compares the number of operations performed by each party for a channel update. To count the operations, we additionally assume that i) channel parties delegate the monitoring task to a watchtower and ii) they do not compute a signature unless it is supposed to be sent to their counter-party or their watchtower. Appendix H provides complete details regarding the way figures of Table 3 have been computed.

8 Discussions

Compatibility with P2WSH transactions: Let output of commit transactions be of type P2WSH, meaning that it can be spent based on the fulfilment of the script whose hash is included in the condition part of the output. Now, assume that while the channel is in state n, party A publishes $\mathtt{TX}^{A}_{\mathtt{CM},i}$ with $i < n$. According to the protocol, party B is supposed to instantly publish the latest revocation transaction $\mathtt{TX}^{B}_{\mathtt{RV},n-1}$. To do so, he must create the original script of the main output of $\mathtt{TX}^{A}_{\mathtt{CM},i}$ and add it to the witness data of $\mathtt{TX}^{B}_{\mathtt{RV},n-1}$. Since the

[2] Pay-to-Witness-Script-Hash: Used to lock bitcoin to a SegWit script hash.

[3] Pay-to-Witness-Public-Key-Hash: Used to lock bitcoin to a SegWit public key hash.

Table 3. On-chain cost of different closure scenarios and number of operations performed by each party for a channel update for different payment channels with m HTLC outputs ($0 \leq m \leq 966$). Cerberus [11], Sleepy [10] and Outpost [23] have not explained ways that HTLC outputs can be added to their schemes, so their figures in this table are for $m = 0$ only.

	dishonest closure		non-coll. closure		num. of operations		
Scheme	#Tx	weight units	#Tx	weight units	Sign	Verify	Exp.
Lightning [28]	≥ 2	$\geq$1209+582.5m	1+m	724+793m	2+2m	$1+\frac{m}{2}$	2
Generalized [8]	2	1342	2+m	1432+696m	3	2	1
FPPW [26]	2	2045	2+m	1562+696m	6	10	1
Cerberus [11]	2	1798	1	772	3	6	0
Outpost [23]	3	2632	3	3018	4	4	0
Sleepy [10]	3	2172	3	2558	5	5	0
eltoo [18]	3	2268+696m	2+m	1588+696m	2	2	1
Daric (this work)	2	1239	2+m	1363+696m	4	3	0

parameter i is a part of the script, B must extract the value of i from the published commit transaction. However, i varies in different commit transactions and its value cannot be directly derived from the hash of the script in the commit transaction output. Therefore, the value of i must be encoded in $\mathtt{TX}^{A}_{\mathtt{CM},i}$.nLT or in the parameter *sequence* of $\mathtt{TX}^{A}_{\mathtt{CM},i}$.Input.

Fee handling: Once a dishonest channel party publishes a revoked commit transaction, her counter-party has T rounds time to publish the revocation transaction on the ledger. However, the time it takes for a transaction to be published depends on two factors: i) network congestion at the time when the transaction is submitted to the blockchain network, and ii) the transaction fee. Revocation transactions in Daric have a single input and a single output. Since based on BIP 143 [24], the `ANYPREVOUT` flag may be combined with `SINGLE` flag, it is possible for a channel party to add a new input and a new output to the latest revocation transaction before submitting it to the blockchain. The difference between value of the new output and the new input can be collected by miners. Similar approach can also be used for commit transactions.

Compatibility with any digital signature scheme: Generalized and FPPW payment channels leverage adaptor signatures and hence may not work if the current Bitcoin digital signature scheme changes to BLS [14] or a post-quantum resistant digital signature. However, Daric is compatible with any digital signature and can benefit from their properties.

Extending Daric to multi-hop payments: Payment channels typically use HTLC to establish a PCN, where parties who do not have a shared channel can still exchange coins by using other nodes as relays. HTLC outputs can simply be added to outputs of split transactions and since state duplication is avoided there is not any complications in their deployment.

Other applications: To have a new application on top of a Daric channel, parties must update the channel state such that the new split transaction has one or multiple outputs for the new application. For example, assume that channel parties want to create multiple channels on top of their existing channel. To do so, they update their channel such that the new split transaction of the channel consists of multiple outputs where each output is a 2-of-2 multisignature address shared between channel parties and acts like the output of a funding transaction for a new Daric channel. The only difference between this new channel and the original one is that since the split transaction for the original channel is floating, its transaction identifier depends on its input and so cannot be determined in advance. Hence, the commit transactions of new established channels must be also floating. The only important criteria for new channels is that each channel must have its own set of public keys. Otherwise, for example, a commit transaction from one channel can spend funding transaction output of another channel.

Channel reset: If the lifetime of a Daric channel is close to its end (which occurs if the channel update rate is more than once per second), channel parties can reset the channel off-chain. To do so, they update the channel such that output of the split transaction in the latest state acts like the output of the funding transaction for a new channel. All the state numbers also reset and the new established channel can be updated at least for about 1 billion times again. Along with required data from the new established channel, each party must also maintain the last commit, split and revocation transactions from the original channel.

9 Conclusion and Future Work

In this work, we presented an efficient payment channel with unlimited lifetime for Bitcoin, called Daric, that achieves optimal storage. Moreover, the new scheme allows the honest channel party to penalize her dishonest counter-party by taking all the channel funds. Daric also guarantees that channel parties can close the channel within a bounded time. Furthermore, the new scheme is compatible with any digital signature algorithm and simultaneously avoids state duplication. We proved Daric is secure in Universal Compsability model.

An interesting open topic to study is extending Daric to an efficient m-party scheme with $m > 2$. Moreover, one of the main advantages of Daric is that storage requirements of the watchtower for each channel could be constant over time. However, there are also other factors for a watchtower (e.g. privacy, fairness, and coverage [26]) which must be carefully taken into account. Designing a watchtower for Daric which can achieve the mentioned properties could be another subject of future research.

References

1. Bolt 3: Bitcoin transaction and script formats, available at https://github.com/lightning/bolts/blob/master/03-transactions.md#appendix-a-expected-weights

2. Bolt 5: Recommendations for on-chain transaction handling, available at availableat:https://github.com/lightningnetwork/lightning-rfc/blob/master/05-onchain.md
3. Lightning channels - top capacity, available at, https://1ml.com/channel?order=capacity
4. A proof-of-concept implementation to evaluate a utxo-based generalized channel construction in bitcoin. available at: https://github.com/generalized-channels/gc
5. eltoo: A simplified update mechanism for lightning and off-chain contracts, available at (2018), https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-June/001313.html
6. Using per-update credential to enable eltoo-penalty. URL: https://lists.linuxfoundation.org/pipermail/lightning-dev/2019-July/002068.html (2019)
7. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostakova, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Generalized bitcoin-compatible channels. IACR Cryptol. ePrint Arch. **2020**, 476 (2020)
8. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostáková, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Generalized channels from limited blockchain scripts and adaptor signatures. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 635–664. Springer (2021)
9. Aumayr, L., Maffei, M., Ersoy, O., Erwig, A., Faust, S., Riahi, S., Hostáková, K., Moreno-Sanchez, P.: Bitcoin-compatible virtual channels. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 901–918. IEEE (2021)
10. Aumayr, L., Thyagarajan, S.A., Malavolta, G., Moreno-Sanchez, P., Maffei, M.: Sleepy channels: Bitcoin-compatible bi-directional payment channels without watchtowers. Cryptology ePrint Archive (2021)
11. Avarikioti, G., Litos, O.S.T., Wattenhofer, R.: Cerberus channels: Incentivizing watchtowers for bitcoin. Financial Cryptography and Data Security (FC) (2020)
12. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: A composable treatment. In: Annual international cryptology conference. pp. 324–356. Springer (2017)
13. Bamert, T., Decker, C., Elsen, L., Wattenhofer, R., Welten, S.: Have a snack, pay with bitcoins. In: IEEE P2P 2013 Proceedings. pp. 1–5. IEEE (2013)
14. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 514–532. Springer (2001)
15. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145. IEEE (2001)
16. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Theory of Cryptography Conference. pp. 61–85. Springer (2007)
17. Decker, C.: Bip 118 - sighashnoinput. URl: https://github.com/bitcoin/bips/blob/master/bip-0118.mediawiki (2017)
18. Decker, C., Russell, R., Osuntokun, O.: eltoo: A simple layer2 protocol for bitcoin. White paper: https://blockstream. com/eltoo. pdf (2018)
19. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Symposium on Self-Stabilizing Systems. pp. 3–18. Springer (2015)
20. Dziembowski, S., Eckey, L., Faust, S., Hesse, J., Hostáková, K.: Multi-party virtual state channels. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 625–656. Springer (2019)

21. Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: Perun: Virtual payment hubs over cryptocurrencies. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 106–123. IEEE (2019)
22. Dziembowski, S., Faust, S., Hostáková, K.: General state channel networks. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 949–966 (2018)
23. Khabbazian, M., Nadahalli, T., Wattenhofer, R.: Outpost: A responsive lightweight watchtower. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. pp. 31–40 (2019)
24. Lau, J., Wuille, P.: BIP 143: Transaction Signature Verification for Version 0 Witness Program (2016), https://github.com/bitcoin/bips/blob/master/bip-0143.mediawiki
25. Lind, J., Naor, O., Eyal, I., Kelbert, F., Sirer, E.G., Pietzuch, P.: Teechain: a secure payment network with asynchronous blockchain access. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles. pp. 63–79 (2019)
26. Mirzaei, A., Sakzad, A., Yu, J., Steinfeld, R.: Fppw: A fair and privacy preserving watchtower for bitcoin. Cryptology ePrint Archive, Report 2021/117 (2021), https://eprint.iacr.org/2021/117
27. Pickhardt, R.: Does eltoo eliminate the need to watch the blockchain/implement watchtowers. URL: https://bitcoin.stackexchange.com/questions/84846/does-eltoo-eliminate-the-need-to-watch-the-blockchain-implement-watchtowers (2019)
28. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)
29. Sompolinsky, Y., Zohar, A.: Accelerating bitcoin's transaction processing. Fast money grows on trees, not chains (2013)
30. Spilman, J.: [bitcoin-development] anti dos for tx replacement. available at: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/ 2013-April/002433.html (2013)
31. Todd, P., Harding, D.A.: Bip 125: Opt-in full replace-by-fee signaling. URl: https://github.com/bitcoin/bips/blob/master/bip-0125.mediawiki (2015)
32. Tsabary, I., Yechieli, M., Manuskin, A., Eyal, I.: Mad-htlc: because htlc is crazy-cheap to attack. arXiv preprint arXiv:2006.12031 (2020)

A Ideal Functionality

This section defines an ideal functionality $\mathcal{F}(T)$ with $T > \Delta$ that achieves the desired properties stated in Section 5.2. To simplify the notations, we abbreviate $\mathcal{F} := \mathcal{F}(T)$. The ideal functionality $\mathcal{F}$ stores a set Γ of all the created channels and their corresponding funding transactions. The set Γ can also be treated as a function s.t. $\Gamma(id) = (\gamma, \texttt{TX})$ with $\gamma.\mathsf{id} = id$ if γ exists and $\Gamma(id) = \perp$ otherwise. Before presenting the ideal functionality $\mathcal{F}$ in details, we briefly introduce its different phases and explain the way $\mathcal{F}$ achieves the desired properties.

a) Create: In this phase, $\mathcal{F}$ receives messages $(\texttt{INTRO}, \gamma, tid_P)$ and $(\texttt{CREATE}, \gamma.\mathsf{id})$ from both parties in rounds τ_0 and $\tau_0 + 1$, respectively, where tid_P specifies the funding source of the user P. Then, if the corresponding funding transaction appears on the ledger $\mathcal{L}$ within $2+\Delta$ rounds, $\mathcal{F}$ sends the message $(\texttt{CREATED}, \gamma.\mathsf{id})$ to both parties and stores γ and the funding transaction in $\Gamma(\gamma.\mathsf{id})$. If the $\texttt{CREATE}$ message is not received from both parties but the funding transaction appears on $\mathcal{L}$ within $2 + \Delta$ rounds, $\mathcal{F}$ outputs `Error`. Since the message `CREATED` might be sent to the parties only if they both have sent the message `CREATE` to $\mathcal{F}$, the ideal functionality achieves consensus on creation.

b) Update: One of the parties, denoted by P, initiates this phase by sending the message $(\texttt{UPDATE}, id, \vec{\theta}, t_{stp})$ to $\mathcal{F}$, where id is the channel identifier, $\vec{\theta}$ is the new channel state and t_{stp} is the number of rounds needed to prepare prerequisites of the channel update (e.g. preparing the needed HTLCs). Due to disagreeing with the new state or failure in preparing its prerequisites, party Q can stop it by not sending the message $(\texttt{UPDATE} - \texttt{OK}, id)$ in step 2. Abort by P or Q in next steps causes the procedure $\texttt{ForceClose}(id)$ to be executed. The property optimistic update is satisfied because if both parties act honestly, the channel can be updated without any blockchain interaction. Furthermore, if P or Q disagree to update the channel, they can stop sending the `UPDATE` or $\texttt{UPDATE} - \texttt{OK}$ messages, respectively. This stops the channel update process without changing the latest channel state. Also, in cases where either P or Q stop cooperating, the procedure $\texttt{ForceClose}(id)$ is executed. This procedure takes at most Δ rounds to complete. This also guarantees consensus on update.

c) Close: If $\mathcal{F}$ receives the message $(\texttt{CLOSE}, id)$ from both parties, a transaction `TX` is expected to appear on $\mathcal{L}$ within $\Delta + 1$ rounds. This transaction spends the output of the funding transaction and its outputs equal the latest channel state $\gamma.\mathsf{st}$. If the `CLOSE` message is received only from one of the parties, $\mathcal{F}$ executes the procedure $\texttt{ForceClose}(id)$. In both cases, output of the funding transaction must be spent within $\Delta + 1$ rounds. Otherwise, $\mathcal{F}$ outputs `Error`.

d) Punish: If a transaction `TX` spends the funding transaction's output of a channel γ, one of the following events is expected to occur: 1) another transaction appears on $\mathcal{L}$ within Δ rounds where this transaction spends output of `TX` and sends $\gamma.\mathsf{cash}$ coins to the honest party P; or 2) another transaction whose outputs correspond to the channel state $\gamma.\mathsf{st}$ or $\gamma.\mathsf{st}'$ appears on $\mathcal{L}$ within $T + \Delta$ rounds. Otherwise, $\mathcal{F}$ outputs `Error`. According to its definition, bounded closure with punish is achieved, if $\mathcal{F}$ returns no `Error` in the close and punish phases.

We describe the ideal functionality below. Normally, once $\mathcal{F}$ receives a message, it performs several validations on the message. But to simplify the description, we assume that messages are well-formed. Data exchange between $\mathcal{F}$ and other parties is represented by directed arrows. If $\mathcal{F}$ sends the message m to party P in round τ_0, we denote it with $m \overset{\tau_0}{\hookrightarrow} P$. Similarly, if $\mathcal{F}$ is supposed to receive the message m from party P in round τ_0, we denote it with $m \overset{\tau_0}{\hookleftarrow} P$.

Ideal Functionality $\mathcal{F}(\mathcal{T})$

Create

upon $(\texttt{INTRO}, \gamma, tid_P) \overset{\tau_0}{\hookleftarrow} P$:

- If $(\texttt{INTRO}, \gamma, tid_Q) \overset{\tau_0}{\hookleftarrow} Q$, then continue. Else stop.
- If $(\texttt{CREATE}, id) \overset{\tau_0+1}{\hookleftarrow} \gamma.users$:
  - Wait if in round $\tau_1 \leq \tau_0 + 3 + \Delta$ a transaction $\texttt{TX}_{\texttt{FU}}$ with $\texttt{TX}_{\texttt{FU}}.\mathsf{Input} = (tid_P, tid_Q)$ and $\texttt{TX}_{\texttt{FU}}.\mathsf{Output} = \{(\gamma.\mathsf{cash}, \varphi)\}$ appears on the ledger $\mathcal{L}$. If yes, set $\Gamma(\gamma.\mathsf{id}) := (\gamma, \texttt{TX}_{\texttt{FU}})$ and $(\texttt{CREATED}, \gamma.\mathsf{id}) \overset{\tau_1}{\hookrightarrow} \gamma.users$. Else stop.

Otherwise:

- Wait if in round $\tau_1 \leq \tau_0 + 3 + \Delta$ a transaction $\texttt{TX}_{\texttt{FU}}$ with $\texttt{TX}_{\texttt{FU}}.\mathsf{Input} = (tid_P, tid_Q)$ and $\texttt{TX}_{\texttt{FU}}.\mathsf{Output} = \{(\gamma.\mathsf{cash}, \varphi)\}$ appears on the ledger $\mathcal{L}$. If yes, Output $\texttt{Error} \overset{\tau_1}{\hookrightarrow} \gamma.\mathsf{users}$. Else, stop.

Update

Upon $(\texttt{UPDATE}, id, \vec{\theta}, t_{stp}) \overset{\tau_0}{\hookleftarrow} P$, parse $(\gamma, \texttt{TX}) := \Gamma(id)$ and proceed as follows:

1. Send $(\texttt{UPDATE}-\texttt{REQ}, id, \vec{\theta}, t_{stp}) \xhookrightarrow{\tau_0+1} Q$.
2. If $(\texttt{UPDATE}-\texttt{OK}, id) \xhookleftarrow{\tau_1 \leq \tau_0+1+t_{stp}} Q$, then set $\gamma.\mathsf{flag} := 2$ and $\gamma.\mathsf{st}' := \vec{\theta}$ and send $(\texttt{SETUP}, id) \xhookrightarrow{\tau_1+1} P$. Else stop.
3. If $(\texttt{SETUP}-\texttt{OK}, id) \xhookleftarrow{\tau_1+1} P$, then $(\texttt{SETUP}', id) \xhookrightarrow{\tau_1+2} Q$. Else $\texttt{ForceClose}(id)$ and stop.
4. If $(\texttt{SETUP}'-\texttt{OK}, id) \xhookleftarrow{\tau_1+2} Q$, then $(\texttt{UPDATE}-\texttt{OK}, id) \xhookrightarrow{\tau_1+3} P$. Else execute $\texttt{ForceClose}(id)$ and stop.
5. If $(\texttt{REVOKE}, id) \xhookleftarrow{\tau_1+3} P$, then $(\texttt{REVOKE}-\texttt{REQ}, id) \xhookrightarrow{\tau_1+4} Q$. Else execute $\texttt{ForceClose}(id)$ and stop.
6. If $(\texttt{REVOKE}', id) \xhookleftarrow{\tau_1+4} Q$, set $\gamma.st := \vec{\theta}$, $\gamma.\mathsf{flag} := 1$, $\gamma.\mathsf{st}' := \perp$, $\gamma.\mathsf{sn} := \gamma.\mathsf{sn} + 1$, $\Gamma(id) := (\gamma, \texttt{TX})$, $(\texttt{UPDATED}, id) \xhookrightarrow{\tau_1+5} \gamma.\mathsf{Users}$ and stop. Else execute $\texttt{ForceClose}(id)$ and stop.

Close

upon $(\texttt{CLOSE}, id) \overset{\tau_0}{\hookleftarrow} P$, distinguish:

Both agreed: If $(\texttt{CLOSE}, id) \overset{\tau_0}{\hookleftarrow} Q$, let $(\gamma, \texttt{TX}_{\texttt{FU}}) := \Gamma(id)$ and distinguish:

- If in round $\tau_1 \leq \tau_0 + 1 + \Delta$, $\texttt{TX}_{\overline{\texttt{SP}}}$, with $\texttt{TX}_{\overline{\texttt{SP}}}.\mathsf{Output} = \gamma.\mathsf{st}$ and $\texttt{TX}_{\overline{\texttt{SP}}}.\mathsf{Input} = \texttt{TX}_{\texttt{FU}}.\mathsf{txid}||1$ appears on $\mathcal{L}$, set $\Gamma(id) := (\perp, \texttt{TX}_{\texttt{FU}})$, $(\texttt{CLOSED}, id) \overset{\tau_1}{\hookrightarrow} \gamma.\mathsf{users}$ and stop.
- If in round $\tau_0+1+\Delta$, the $\texttt{TX}_{\texttt{FU}}$ is still unspent, output $\texttt{Error} \xhookrightarrow{\tau_0+1+\Delta} \gamma.\mathsf{users}$ and stop.

Q **disagreed:** Else, execute $\texttt{ForceClose}(id)$ in round $\tau_0 + 1$.

Punish (executed at the end of every round τ_0)

For each $(\gamma_i, \texttt{TX}_i) \in \Gamma$ check if there is a transaction $\texttt{TX}$ on the ledger $\mathcal{L}$ s.t. $\texttt{TX}.\mathsf{Input} = \texttt{TX}_i.\mathsf{txid}||1$ and $\gamma_i \neq \perp$. If yes, distinguish:

1. **Punish**: For the honest $P \in \gamma_i.\mathsf{users}$, in round $\tau_1 \leq \tau_0 + \Delta$, a transaction $\texttt{TX}_j$ with $\texttt{TX}_j.\mathsf{Input} = \texttt{TX}.\mathsf{txid}||1$ and $\texttt{TX}_j.\mathsf{Output} = (\gamma.\mathsf{cash}, pk_P)$ appears on $\mathcal{L}$. Then, $(\texttt{PUNISHED}, id) \overset{\tau_1}{\hookrightarrow} P$, set $\Gamma(id) := (\perp, \texttt{TX}_i)$ and stop.

2. **Close**: In round $\tau_1 \leq \tau_0 + T + \Delta$ a transaction $\mathtt{TX}_j$ appears on $\mathcal{L}$ where one of the following two sets of conditions hold: 1) $\gamma.\mathsf{flag} = 1$, $\mathtt{TX}_j.\mathsf{Input} = \mathtt{TX}.\mathsf{txid}\|1$ and $\mathtt{TX}_j.\mathsf{Output} = \gamma.\mathsf{st}$ or 2) $\gamma.\mathsf{flag} = 2$, $\mathtt{TX}_j.\mathsf{Input} = \mathtt{TX}.\mathsf{txid}\|1$ and either $\mathtt{TX}_j.\mathsf{Output} = \gamma.\mathsf{st}$ or $\mathtt{TX}_j.\mathsf{Output} = \gamma.\mathsf{st}'$. Then, set $\Gamma(id) := (\perp, \mathtt{TX}_i)$ and $(\mathtt{CLOSED}, id) \xhookrightarrow{\tau_1} \gamma.\mathsf{users}$.
3. **Error**: Otherwise, $\mathtt{Error} \xhookrightarrow{\tau_0+T+\Delta} \gamma.\mathsf{users}$.

Subprocedure $\mathtt{ForceClose}(id)$

Let τ_0 be the current round and $(\gamma, \mathtt{TX}_{\mathtt{FU}}) := \Gamma(id)$. If within Δ rounds, $\mathtt{TX}_{\mathtt{FU}}.\mathsf{Output}$ is still an unspent output on $\mathcal{L}$, then output $\mathtt{Error} \xhookrightarrow{\tau_0+\Delta} \gamma.\mathsf{users}$.

B Daric Transactions Scripts

Funding transaction has one output with the following script where `Com_pubkeyA` and `Com_pubkeyB` are public keys of A and B, respectively:

```
2 ⟨Com_pubkeyA⟩ ⟨Com_pubkeyB⟩ 2 OP_CHECKMULTISIG
```

Commit transactions have one input that takes the output of the funding transaction with the witness script of `0 ⟨Com_pubkeyA_sig⟩ ⟨Com_pubkeyB_sig⟩`. The commit transaction $\mathtt{TX}^{A}_{\mathtt{CM},i}$ has one output with the following script:

```
⟨absolute time S0 + i ⟩ OP_CHECKLOCKTIMEVERIFY OP_DROP
OP_IF
  # Revocation
  2 ⟨Rev_pubkeyA⟩ ⟨Rev_pubkeyB⟩ 2 OP_CHECKMULTISIG
OP_ELSE
  # Split
  ⟨delay T⟩ OP_CHECKSEQUENCEVERIFY OP_DROP
  2 ⟨Spl_pubkeyA⟩ ⟨Spl_pubkeyB⟩ 2 OP_CHECKMULTISIG
OP_ENDIF
```

where `⟨Rev_pubkeyA⟩` and `⟨Spl_pubkeyA⟩` are public keys of A and `⟨Rev_pubkeyB⟩` and `⟨Spl_pubkeyB⟩` are public keys of B. The script of the commit transaction $\mathtt{TX}^{B}_{\mathtt{CM},i}$ is similar to that of $\mathtt{TX}^{A}_{\mathtt{CM},i}$ but its revocation keys are `⟨Rev′_pubkeyA⟩` and `⟨Rev′_pubkeyB⟩`.

The split transaction $\mathtt{TX}_{\mathtt{SP},i}$ spends the output of $\mathtt{TX}^{A}_{\mathtt{CM},i}$ or $\mathtt{TX}^{B}_{\mathtt{CM},i}$ with the witness script:

```
0 ⟨Spl_pubkeyA_Sig⟩ ⟨Spl_pubkeyB_Sig⟩ 0
```

The revocation transactions $\mathtt{TX}^{A}_{\mathtt{RV},i}$ and $\mathtt{TX}^{B}_{\mathtt{RV},i}$ spend the output of a revoked commit transaction with the witness scripts

```
0 ⟨Rev′_pubkeyA_sig⟩ ⟨Rev′_pubkeyB_sig⟩ 1
```

and

```
0 ⟨Rev_pubkeyA_sig⟩ ⟨Rev_pubkeyB_sig⟩ 1,
```

respectively. The transactions $\mathtt{TX}^{A}_{\mathtt{RV},i}$ and $\mathtt{TX}^{B}_{\mathtt{RV},i}$ have one output with the scripts `⟨pubkeyA⟩ OP_CHECKSIG` and `⟨pubkeyB⟩ OP_CHECKSIG`, respectively.

C UC Framework

We model the security of our protocol in the synchronous version of global UC framework (GUC) [16] which is an extension of standard UC framework [15]. In the synchronous version, we can have a global setup which is used to model the ledger. The model in this work closely follows that of some works on layer-2 solutions to scalability of blockchains [7,22,20].

Let π be a protocol executed among parties of a set $\mathcal{P} = \{P_1, \cdots, P_n\}$. Assume that there exists an adversary $\mathcal{A}$ that takes as input a security parameter $\lambda \in \mathbb{N}$ and an auxiliary input $z \in \{0,1\}^*$. Before execution of the protocol π, the adversary $\mathcal{A}$ can select any party $P_i \in \mathcal{P}$ to learn their internal state and fully control them. Anything outside the protocol execution is modeled by the environment $\mathcal{E}$. Each protocol party as well as the adversary take their inputs from the environment. Outputs of all parties are also observed by the environment. There are also ideal functionalities $\mathcal{F}_1, \cdots, \mathcal{F}_m$ whose functions might be called by parties. Then, the protocol π is denoted by $\pi^{\mathcal{F}_1, \cdots, \mathcal{F}_m}$.

To simplify our model, we assume that the communication network is synchronous, meaning that we let the protocol be executed through several rounds. The ideal functionality $\mathcal{F}_{clock}$ represents a global clock that increases by one unit once all parties are prepared to proceed to the next round. All entities know the value of the current round.

We assume that parties of the protocol are connected to each other via an authenticated communication channel which guarantees that messages are delivered to recipient parties after exactly one round. In other words, if party P sends a message m to the party Q in round τ, the message reaches Q in the beginning of round $\tau + 1$ and Q can ensure that the sender is P. The adversary $\mathcal{A}$ observes the message m and can even change the order of messages that are sent in the same round. However, the adversary cannot drop, delay or change any transmitted message or insert new messages. The ideal functionality $\mathcal{F}_{GDC}$ models such a communication channel between the channel parties. Other communications in which some other entities, e.g. $\mathcal{A}$ or $\mathcal{E}$, are involved take zero rounds to complete. Moreover, to simplify the model, we assume that any required computation is also performed within zero rounds.

In this work we focus on UTXO-based cryptocurrencies such as Bitcoin. The *global* ideal functionality $\mathcal{L}$ models such ctyptocurrencies where it is parameterized by two parameters: 1) a digital signature scheme Σ, and 2) a delay parameter Δ which is an upper bound on the number of rounds it takes for a valid transaction that has been posted to the blockchain network to be published on the blockchain. To simplify the model, we assume that the set of parties $\mathcal{P}$ is fixed and transactions are published one by one rather than being published on the blockchain in blocks. Badertscher et. al [12] provided a more accurate model of Bitcoin blockchain.

The environment $\mathcal{E}$ initiates the ledger functionality $\mathcal{L}$ by 1) instructing $\mathcal{L}$ to generate the public parameters of the signature scheme Σ, 2) instructing each party $P \in \mathcal{P}$ to create a key pair (pk_P, sk_P) and submit its public key to $\mathcal{L}$, and 3) creating an initial state TX that contains all the accepted transactions. The set TX is accessible to everyone including the protocol parties, the environment and the adversary. Once a party $P \in \mathcal{P}$ posts a transaction tx to the bockchain, $\mathcal{L}$ waits for $\tau \leq \Delta$ rounds where τ is selected by the adversary. Then, if the validity of tx is successfully verified, it is added to the set TX.

Ideal Functionality $\mathcal{L}(\Delta, \Sigma)$

The set $\mathcal{P}$ defines the set of all parties who can send messages to the functionality. The functionality maintains the set PKI for the parties in $\mathcal{P}$. The sets TX and UTXO respectively define all the transactions accepted to date and all the unspent transaction outputs. The set of valid output conditions is represented by $\mathcal{V}$.

Public key Registration: Upon $(\text{register}, pk_P) \overset{\tau_0}{\hookleftarrow} P$, check if it is the first registration message received from $P \in \mathcal{P}$. If not drop the message, else add (pk_P, P) to PKI.

Post transaction: Upon $(\text{post}, \mathsf{tx}) \overset{\tau_0}{\hookleftarrow} P$, check if $|\mathsf{PKI}| = |\mathcal{P}|$, If not drop the message, else wait for $\tau \leq \Delta$ rounds where τ is selected by the adversary. Then, check if:

1. id uniqueness: For all $(t, \mathsf{tx}') \in \mathsf{TX}$, $\mathsf{tx}'.\mathsf{txid} \neq \mathsf{tx}.\mathsf{txid}$ holds.
2. Input and witness validity: For each $(txid \| i) \in \mathsf{tx}.\mathsf{Input}$, there exists $(t, txid, i, \theta) \in \mathsf{UTXO}$ s.t. $\mathsf{tx}.\mathsf{Witness}$ with inputs $\mathsf{tx}.\mathsf{nLT}$, the current round $\tau_0 + \tau$ and t satisfies $\theta.\varphi$.
3. Output validity: For each $\theta \in \mathsf{tx}.\mathsf{Output}$, $\theta.\mathsf{Cash} > 0$ and $\theta.\varphi \in \mathcal{V}$ hold.
4. Value validity: Let $I := \{\mathsf{utxo} := (t, txid, i, \theta) \mid \mathsf{utxo} \in \mathsf{UTXO} \wedge (txid \| i) \in \mathsf{tx}.\mathsf{Input}\}$. Then, $\Sigma_{\theta' \in \mathsf{tx}.\mathsf{Output}} \theta'.\mathsf{cash} \leq \Sigma_{\mathsf{utxo} \in I} \mathsf{utxo}.\theta.\mathsf{cash}$ holds.
5. Absolute timelock validity: For the transaction tx, $\mathsf{tx}.\mathsf{nLT} \leq \tau_0 + \tau$ holds.

If any of the above checks fail, drop the message. Else, set $\mathsf{TX} := \mathsf{TX} \bigcup (\tau_0 + \tau, \mathsf{tx})$, $\mathsf{UTXO} := \mathsf{UTXO} \backslash I$ and $\mathsf{UTXO} := \mathsf{UTXO} \bigcup \{(\tau_0 + \tau, \mathsf{tx}.\mathsf{txid}, i, \theta_i)\}_{i \in [n]}$ for $(\theta_1, \cdots, \theta_n) := \mathsf{tx}.\mathsf{Output}$.

Let π be a protocol that has access to the global ledger $\mathcal{L}(\Delta, \Sigma)$ as well as the global clock $\mathcal{F}_{clock}$ and $\varphi_{\mathcal{F}}$ denote the ideal protocol for an ideal functionality $\mathcal{F}$ with access to the same global functionalities. Let $\mathrm{EXE}_{\pi, \mathcal{A}, \mathcal{E}}^{\mathcal{L}(\Delta, \Sigma), \mathcal{F}_{clock}}(\lambda, z)$ denote the output of the environment $\mathcal{E}$ which interacts with a protocol π and an adversary $\mathcal{A}$ on input a security parameter λ and an auxiliary input z and similarly $\mathrm{EXE}_{\varphi_{\mathcal{F}}, \mathcal{S}, \mathcal{E}}^{\mathcal{L}(\Delta, \Sigma), \mathcal{F}_{clock}}(n, z)$ denote the output of the environment $\mathcal{E}$ which interacts with a protocol $\varphi_{\mathcal{F}}$ and an adversary $\mathcal{S}$ (also called the simulator) on input a security parameter λ and an auxiliary input z.

The following definition is informally saying that should a protocol π UC-realize $\mathcal{F}$, any attack against the protocol π can be transformed into an attack against the ideal protocol $\varphi_{\mathcal{F}}$ and vice versa.

Definition 1. *A protocol π UC-realizes an ideal functionality $\mathcal{F}$ with respect to a global ledger $\mathcal{L}(\Delta, \Sigma)$ and a global clock $\mathcal{F}_{clock}$ if for every adversary $\mathcal{A}$ there* *exists an adversary $\mathcal{S}$ such that we have*

$$\begin{aligned}&\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\pi,\mathcal{A},\mathcal{E}}(\lambda,z)\}_{n\in\mathbb{N},z\in\{0,1\}^*}\approx\\&\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\varphi_{\mathcal{F}},\mathcal{S},\mathcal{E}}(\lambda,z)\}_{n\in\mathbb{N},z\in\{0,1\}^*}\end{aligned}\tag{1}$$

where $\approx$ denotes computational indistinguishability.

D Protocol

In this appendix, details of different phases of Daric will be presented. Before presenting the protocol, some notations are introduced. Formally, an output θ is a tuple of two attributes, $\theta = (\mathsf{cash}, \varphi)$, where $\theta.\mathsf{cash}$ denotes the number of coins held in this output and $\theta.\varphi$ denotes the condition that needs to be fulfilled to spend the output θ. The condition $\theta.\varphi$ is encoded using any script supported by the underlying blockchain. If the condition $\theta.\varphi$ contains a user P's public key, we say that P controls or owns the output θ. If satisfying a condition requires authorizations by multiple parties, such a condition contains public keys of all the involved parties separated by $\wedge$ operation(s). Different subconditions of an output are separated by $\vee$ operation(s).

The attribute $Tx.\mathsf{Witness} = (\mathsf{W}_1, \ldots, \mathsf{W}_m)$ is a list of tuples where its i^{th} tuple authorizes spending the output that is taken as the i^{th} input of Tx. The tuple $\mathsf{W}_i = (\eta, \zeta)$ of the witness $Tx.\mathsf{Witness}$ contains two attributes where $\mathsf{W}_i.\zeta$ denotes the data, e.g. the signature(s), that is (are) required to meet the $\mathsf{W}_i.\eta^{\text{th}}$ subcondition of the output that is taken as the i^{th} input of Tx. The signature of party P for $Tx.\mathsf{Witness}.\mathsf{W}_i.\zeta$ is denoted by $\sigma^{P,i}_{Tx}$, where i can be removed for transactions with single input. If the signature is created using the `ANYPREVOUT` flag, it is denoted by $\tilde{\sigma}^{P,i}_{Tx}$.

In different steps of the protocol, channel participants generate (or verify) some signatures on protocol transactions. When a signature with SIGHASH of type `SIGHASH_ALL` or `ANYPREVOUT` is going to be generated (or verified) for the transaction `TX`, the input message to the signing (or verification) algorithm is denoted by $f(\mathtt{TX})$ or $\tilde{f}(\overline{[\mathtt{TX}]})$, respectively [24].

The set Γ^P, maintained by each party $P \in \mathcal{P}$, stores information of latest channel state for all the open channels that P is involved in, where $\Gamma^P(id)$ corresponds with the channel with identifier of id. In the channel update phase, while $\gamma.\mathsf{flag} = 2$, the channel has two active states where the information about the new state is maintained by $P \in \mathcal{P}$ in $\Gamma'^P(id)$ with $id = \gamma.\mathsf{id}$. To refer to the i^{th} element of $\Gamma'^P(id)$ we use $\Gamma^P(id)[i]$ with $i \geq 1$. The party P also maintains a signature from his counter-party on the latest revocation transaction for each open channel that P is a party of. These signatures are maintained by P in the set Θ^P where $\Theta^P(id)$ corresponds with the channel with identifier of id. We use directional arrows to show exchange of messages. To simplify the protocol description, we remove some validations that must be normally done by channel parties. The protocol wrapper $\mathcal{W}_{\mathcal{P}}$ in Appendix F defines those validations.

Daric protocol:

Create

Party P upon $(\mathtt{INTRO}, \gamma, tid_P) \xhookleftarrow{\tau_0} \mathcal{E}$

1. Set $id := \gamma.\mathsf{id}$, generate $(pk^P_{\mathtt{SP}}, sk^P_{\mathtt{SP}}) \leftarrow \mathsf{Gen}$, $(pk^P_{\mathtt{RV}}, sk^P_{\mathtt{RV}}) \leftarrow \mathsf{Gen}$ and $(pk'^P_{\mathtt{RV}}, sk'^P_{\mathtt{RV}}) \leftarrow \mathsf{Gen}$ and send $(\text{createInfo}, id, tid_P, pk^P_{\mathtt{SP}}, pk^P_{\mathtt{RV}}, pk'^P_{\mathtt{RV}}) \xhookrightarrow{\tau_0} Q$.
2. If $(\text{createInfo}, id, tid_Q, pk^Q_{\mathtt{SP}}, pk^Q_{\mathtt{RV}}, pk'^Q_{\mathtt{RV}}) \xhookleftarrow{\tau_0+1} Q$, create:

$$\begin{aligned}
[\mathtt{TX}_{\mathtt{FU}}] &:= \mathtt{GenFund}((tid_P, tid_Q), \gamma) \\
([\mathtt{TX}^P_{\mathtt{CM},0}], [\mathtt{TX}^Q_{\mathtt{CM},0}]) &:= \\
&\mathtt{GenCommit}([\mathtt{TX}_{\mathtt{FU}}].\mathsf{txid}\|1, I_P, I_Q, 0) \\
\overline{[\mathtt{TX}_{\mathtt{SP},0}]} &:= \mathtt{GenSplit}(\gamma.\mathsf{st}, 0)
\end{aligned}$$

   for $I_P := (pk^P_{\mathtt{SP}}, pk^P_{\mathtt{RV}}, pk'^P_{\mathtt{RV}})$ and $I_Q := (pk^Q_{\mathtt{SP}}, pk^Q_{\mathtt{RV}}, pk'^Q_{\mathtt{RV}})$. Else stop.
3. Compute $\tilde{\sigma}^P_{\mathtt{TX}_{\mathtt{SP},0}} := \mathsf{Sign}_{sk^P_{\mathtt{SP}}}(\tilde{f}(\overline{[\mathtt{TX}_{\mathtt{SP},0}]}))$ and $\sigma^P_{\mathtt{TX}^Q_{\mathtt{CM},0}} := \mathsf{Sign}_{sk_P}(f([\mathtt{TX}^Q_{\mathtt{CM},0}]))$ and send $(\text{createCom}, id, \tilde{\sigma}^P_{\mathtt{TX}_{\mathtt{SP},0}}, \sigma^P_{\mathtt{TX}^Q_{\mathtt{CM},0}}) \xhookrightarrow{\tau_0+1} Q$.
4. If $(\text{createCom}, id, \tilde{\sigma}^Q_{\mathtt{TX}_{\mathtt{SP},0}}, \sigma^Q_{\mathtt{TX}^P_{\mathtt{CM},0}}) \xhookleftarrow{\tau_0+2} Q$, s.t. $\mathsf{Vrfy}_{pk^Q_{\mathtt{SP}}}(\tilde{f}(\overline{[\mathtt{TX}_{\mathtt{SP},0}]}); \tilde{\sigma}^Q_{\mathtt{TX}_{\mathtt{SP},0}}) = 1$ and also $\mathsf{Vrfy}_{pk_Q}(f([\mathtt{TX}^P_{\mathtt{CM},0}]); \sigma^Q_{\mathtt{TX}^P_{\mathtt{CM},0}}) = 1$, then $\sigma^P_{\mathtt{TX}_{\mathtt{FU}}} := \mathsf{Sign}_{sk_P}(f([\mathtt{TX}_{\mathtt{FU}}]))$ and send $(\text{createFund}, id, \sigma^P_{\mathtt{TX}_{\mathtt{FU}}}) \xhookrightarrow{\tau_0+2} Q$. Else stop.
5. If $(\text{createFund}, id, \sigma^Q_{\mathtt{TX}_{\mathtt{FU}}}) \xhookleftarrow{\tau_0+3} Q$, s.t. $\mathsf{Vrfy}_{pk_Q}(f([\mathtt{TX}_{\mathtt{FU}}]); \sigma^Q_{\mathtt{TX}_{\mathtt{FU}}}) = 1$, create the transaction $\mathtt{TX}_{\mathtt{FU}} := (\mathcal{H}([\mathtt{TX}_{\mathtt{FU}}]), [\mathtt{TX}_{\mathtt{FU}}], ((x, \sigma^P_{\mathtt{TX}_{\mathtt{FU}}}), (y, \sigma^Q_{\mathtt{TX}_{\mathtt{FU}}})))$ and $(\text{post}, \mathtt{TX}_{\mathtt{FU}}) \xhookrightarrow{\tau_0+3} \mathcal{L}$. Else, create a transaction $\mathtt{TX}$ with $\mathtt{TX}.\mathsf{Input} := tid_P$ and $\mathtt{TX}.\mathsf{Output}.\varphi := pk_P$ and $(\text{post}, \mathtt{TX}) \xhookrightarrow{\tau_0+3} \mathcal{L}$.
6. If $\mathtt{TX}_{\mathtt{FU}}$ is accepted by $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + 3 + \Delta$, compute $\sigma^P_{\mathtt{TX}^P_{\mathtt{CM},0}} = \mathsf{Sign}_{sk_P}(f([\mathtt{TX}^P_{\mathtt{CM},0}]))$, create $\mathtt{TX}^P_{\mathtt{CM},0} := (\mathcal{H}([\mathtt{TX}^P_{\mathtt{CM},0}]), [\mathtt{TX}^P_{\mathtt{CM},0}], (1, \{\sigma^P_{\mathtt{TX}^P_{\mathtt{CM},0}}, \sigma^Q_{\mathtt{TX}^P_{\mathtt{CM},0}}\})$, set $\overline{\mathtt{TX}_{\mathtt{SP},0}} := (\overline{[\mathtt{TX}_{\mathtt{SP},0}]}, (1, \{\tilde{\sigma}^P_{\mathtt{TX}_{\mathtt{SP},0}}, \tilde{\sigma}^Q_{\mathtt{TX}_{\mathtt{SP},0}}\}))$, store $\Gamma^P(\gamma.id) := (\gamma, \mathtt{TX}_{\mathtt{FU}}, \mathtt{TX}^P_{\mathtt{CM},0}, [\mathtt{TX}^Q_{\mathtt{CM},0}], \overline{\mathtt{TX}_{\mathtt{SP},0}})$ and $(\mathtt{CREATED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$. Else $\Gamma^P(\gamma.id) := (\perp, \mathtt{TX}_{\mathtt{FU}}, \perp, \perp, \perp)$ and stop.

Update

Party P upon $(\mathtt{UPDATE}, id, \vec{\theta}, t_{stp}) \xhookleftarrow{\tau_0} \mathcal{E}$

1. Send $(\text{updateReq}, id, \vec{\theta}, t_{stp}) \xhookrightarrow{\tau_0} Q$.

Party Q upon $(\text{updateReq}, id, \vec{\theta}, t_{stp}) \xhookleftarrow{t_0} P$

2. Send $(\mathtt{UPDATE-REQ}, id, \vec{\theta}, t_{stp}) \xhookrightarrow{t_0} \mathcal{E}$.
3. If $(\mathtt{UPDATE-OK}, id) \xhookleftarrow{t_1 \leq t_0 + t_{stp}} \mathcal{E}$, then extract $\mathtt{TX}_{\mathtt{FU}}$ and $i := \gamma.\mathsf{sn}$ from $\Gamma^Q(id)$, create $([\mathtt{TX}^P_{\mathtt{CM},i+1}], [\mathtt{TX}^Q_{\mathtt{CM},i+1}]) := \mathtt{GenCommit}([\mathtt{TX}_{\mathtt{FU}}].\mathsf{txid}\|1, I_P, I_Q, i+1)$, and $\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]} := \mathtt{GenSplit}(\vec{\theta}, i+1)$ for $I_P := (pk^P_{\mathtt{SP}}, pk^P_{\mathtt{RV}}, pk'^P_{\mathtt{RV}})$ and $I_Q := (pk^Q_{\mathtt{SP}}, pk^Q_{\mathtt{RV}}, pk'^Q_{\mathtt{RV}})$, compute $\tilde{\sigma}^Q_{\mathtt{TX}_{\mathtt{SP},i+1}} = \mathsf{Sign}_{sk^Q_{\mathtt{SP}}}(\tilde{f}(\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]}))$, and send $(\text{updateInfo}, id, \tilde{\sigma}^Q_{\mathtt{TX}_{\mathtt{SP},i+1}}) \xhookrightarrow{t_1} P$.

Party P upon $(\text{updateInfo}, id, \tilde{\sigma}^Q_{\mathtt{TX}_{\mathtt{SP},i+1}}) \xhookleftarrow{\tau_1 \leq \tau_0 + 2 + t_{stp}} Q$

4. Extract $\mathtt{TX}_{\mathtt{FU}}$ and $i := \gamma.sn$ from $\Gamma^P(id)$ and create the transactions $([\mathtt{TX}^P_{\mathtt{CM},i+1}], [\mathtt{TX}^Q_{\mathtt{CM},i+1}]) := \mathtt{GenCommit}([\mathtt{TX}_{\mathtt{FU}}].\mathsf{txid}\|1, I_P, I_Q, i{+}1)$ and $\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]} := \mathtt{GenSplit}(\vec{\theta}, i+1)$ for $I_P := (pk^P_{\mathtt{SP}}, pk^P_{\mathtt{RV}}, pk'^P_{\mathtt{RV}})$ and $I_Q := (pk^Q_{\mathtt{SP}}, pk^Q_{\mathtt{RV}}, pk'^Q_{\mathtt{RV}})$. If $\mathsf{Vrfy}_{pk^Q_{\mathtt{SP}}}(\tilde{f}(\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]}); \tilde{\sigma}^Q_{\mathtt{TX}_{\mathtt{SP},i+1}}) = 1$, compute $\tilde{\sigma}^P_{\mathtt{TX}_{\mathtt{SP},i+1}} := \mathsf{Sign}_{sk^P_{\mathtt{SP}}}(\tilde{f}(\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]}))$, set $\overline{\mathtt{TX}_{\mathtt{SP},i+1}} := (\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]}, (1, \{\tilde{\sigma}^P_{\mathtt{TX}_{\mathtt{SP},i+1}}, \tilde{\sigma}^Q_{\mathtt{TX}_{\mathtt{SP},i+1}}\}))$, store $\Gamma'^P(id) = (\perp, [\mathtt{TX}^Q_{\mathtt{CM},i+1}], \overline{\mathtt{TX}_{\mathtt{SP},i+1}})$, set $\gamma.\mathsf{flag} = 2$ and $\gamma.\mathsf{st}' = \vec{\theta}$ and send $(\mathtt{SETUP}, id) \xhookrightarrow{\tau_1} \mathcal{E}$. Else stop.
5. If $(\mathtt{SETUP-OK}, id) \xhookleftarrow{\tau_1} \mathcal{E}$, sign $\sigma^P_{\mathtt{TX}^Q_{\mathtt{CM},i+1}} := \mathsf{Sign}_{sk_P}(f([\mathtt{TX}^Q_{\mathtt{CM},i+1}]))$ and output $(\text{updateComP}, id, \tilde{\sigma}^P_{\mathtt{TX}_{\mathtt{SP},i+1}}, \sigma^P_{\mathtt{TX}^Q_{\mathtt{CM},i+1}}) \xhookrightarrow{\tau_1} Q$. Else, execute $\mathtt{ForceClose}^P(id)$ and stop.

Party Q

6. If $(\text{updateComP}, id, \tilde{\sigma}^P_{\mathtt{TX}_{\mathtt{SP},i+1}}, \sigma^P_{\mathtt{TX}^Q_{\mathtt{CM},i+1}}) \xhookleftarrow{t_1+2} P$, such that $\mathsf{Vrfy}_{pk^P_{\mathtt{SP}}}(\tilde{f}(\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]}); \tilde{\sigma}^P_{\mathtt{TX}_{\mathtt{SP},i+1}}) = 1$ and also $\mathsf{Vrfy}_{pk_P}(f([\mathtt{TX}^Q_{\mathtt{CM},i+1}]); \sigma^P_{\mathtt{TX}^Q_{\mathtt{CM},i+1}}) = 1$, compute $\sigma^Q_{\mathtt{TX}^Q_{\mathtt{CM},i+1}} = \mathsf{Sign}_{sk_Q}(f([\mathtt{TX}^Q_{\mathtt{CM},i+1}]))$, set $\overline{\mathtt{TX}_{\mathtt{SP},i+1}} := (\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]}, (1, \{\tilde{\sigma}^P_{\mathtt{TX}_{\mathtt{SP},i+1}}, \tilde{\sigma}^Q_{\mathtt{TX}_{\mathtt{SP},i+1}}\}))$, set $\mathtt{TX}^Q_{\mathtt{CM},i+1} = ([\mathtt{TX}^Q_{\mathtt{CM},i+1}], (1, \{\sigma^P_{\mathtt{TX}^Q_{\mathtt{CM},i+1}}, \sigma^Q_{\mathtt{TX}^Q_{\mathtt{CM},i+1}}\}))$, $\gamma.\mathsf{flag} = 2$ and $\gamma.\mathsf{st}' = \vec{\theta}$, store $\Gamma'^Q(id) = (\mathtt{TX}^Q_{\mathtt{CM},i+1}, [\mathtt{TX}^P_{\mathtt{CM},i+1}], \overline{\mathtt{TX}_{\mathtt{SP},i+1}})$, and output $(\mathtt{SETUP}', id) \xhookrightarrow{t_1+2} \mathcal{E}$. Else, execute $\mathtt{ForceClose}^Q(id)$ and stop.
7. If $(\mathtt{SETUP'-OK}, id) \xhookleftarrow{t_1+2} \mathcal{E}$, sign $\sigma^Q_{\mathtt{TX}^P_{\mathtt{CM},i+1}} = \mathsf{Sign}_{sk_Q}(f([\mathtt{TX}^P_{\mathtt{CM},i+1}]))$ and send $(\text{updateComQ}, id, \sigma^Q_{\mathtt{TX}^P_{\mathtt{CM},i+1}}) \xhookrightarrow{t_1+2} P$. Else, execute $\mathtt{ForceClose}^Q(id)$ and stop.

Party P

8. If $(\text{updateComQ}, id, \sigma^Q_{\mathtt{TX}^P_{\mathtt{CM},i+1}}) \xhookleftarrow{\tau_1+2} Q$, s.t. $\mathsf{Vrfy}_{pk_Q}(f([\mathtt{TX}^P_{\mathtt{CM},i+1}]); \sigma^Q_{\mathtt{TX}^P_{\mathtt{CM},i+1}}) = 1$, compute $\sigma^P_{\mathtt{TX}^P_{\mathtt{CM},i+1}} := \mathsf{Sign}_{sk_P}(f([\mathtt{TX}^P_{\mathtt{CM},i+1}], 1))$, set $\mathtt{TX}^P_{\mathtt{CM},i+1} := (\mathcal{H}([\mathtt{TX}^P_{\mathtt{CM},i+1}]), [\mathtt{TX}^P_{\mathtt{CM},i+1}], (1, \{\sigma^P_{\mathtt{TX}^P_{\mathtt{CM},i+1}}, \sigma^Q_{\mathtt{TX}^P_{\mathtt{CM},i+1}}\}))$, set $\Gamma'^P(id)[1] := \mathtt{TX}^P_{\mathtt{CM},i+1}$ and then output $(\mathtt{UPDATE-OK}, id) \xhookrightarrow{\tau_1+2} \mathcal{E}$. Else, execute the procedure $\mathtt{ForceClose}^P(id)$ and stop.

9. If $(\mathtt{REVOKE}, id) \xhookleftarrow{\tau_1+2} \mathcal{E}$, create $(\overline{[\mathtt{TX}^P_{\mathtt{RV},i}]}, \overline{[\mathtt{TX}^Q_{\mathtt{RV},i}]}) := \mathtt{GenRevoke}(pk_P, pk_Q, \gamma.\mathsf{cash}, i+1)$, compute $\tilde{\sigma}^P_{\mathtt{TX}^Q_{\mathtt{RV},i}} := \mathsf{Sign}_{sk^P_{\mathtt{RV}}}(\tilde{f}(\overline{[\mathtt{TX}^Q_{\mathtt{RV},i}]}))$ if $P = A$ or $\tilde{\sigma}^P_{\mathtt{TX}^Q_{\mathtt{RV},i}} = \mathsf{Sign}_{sk'^P_{\mathtt{RV}}}(\tilde{f}(\overline{[\mathtt{TX}^Q_{\mathtt{RV},i}]}))$ otherwise and send $(\text{revokeP}, id, \tilde{\sigma}^P_{\mathtt{TX}^Q_{\mathtt{RV},i}}) \xhookrightarrow{\tau_1+2} Q$. Else, execute the procedure $\mathtt{ForceClose}^P(id)$ and stop.

Party Q

10. Create $(\overline{[\mathtt{TX}^P_{\mathtt{RV},i}]}, \overline{[\mathtt{TX}^Q_{\mathtt{RV},i}]}) := \mathtt{GenRevoke}(pk_P, pk_Q, \gamma.\mathsf{cash}, i{+}1)$. If $(\text{revokeP}, id, \tilde{\sigma}^P_{\mathtt{TX}^Q_{\mathtt{RV},i}}) \xhookleftarrow{t_1+4} P$, such that $\mathsf{Vrfy}_{pk^P_{\mathtt{RV}}}(\tilde{f}([\mathtt{TX}^Q_{\mathtt{RV},i}]); \tilde{\sigma}^P_{\mathtt{TX}^Q_{\mathtt{RV},i}}) = 1$ if $Q = B$ or $\mathsf{Vrfy}_{pk'^P_{\mathtt{RV}}}(\tilde{f}([\mathtt{TX}^Q_{\mathtt{RV},i}]); \tilde{\sigma}^P_{\mathtt{TX}^Q_{\mathtt{RV},i}}) = 1$ otherwise, set $\Theta^Q(id) := (\tilde{\sigma}^P_{\mathtt{TX}^Q_{\mathtt{RV},i}})$, $\gamma.\mathsf{sn} := i + 1$, $\gamma.\mathsf{st} := \vec{\theta}$, $\Gamma^Q(id) := (\gamma, \mathtt{TX}_{\mathtt{FU}}, \mathtt{TX}^Q_{\mathtt{CM},i+1}, [\mathtt{TX}^P_{\mathtt{CM},i+1}], \mathtt{TX}_{\mathtt{SP},i+1})$, $\gamma.\mathsf{flag} = 1$, and $\Gamma'^Q(id) = (\bot, \bot, \bot)$ and send $(\mathtt{REVOKE-REQ}, id) \xhookrightarrow{t_1+4} \mathcal{E}$. Else, execute the procedure $\mathtt{ForceClose}^Q(id)$ and stop.
11. If $(\mathtt{REVOKE}', id) \xhookleftarrow{t_1+4} \mathcal{E}$, then compute $\tilde{\sigma}^Q_{\mathtt{TX}^P_{\mathtt{RV},i}} := \mathsf{Sign}_{sk^Q_{\mathtt{RV}}}(\tilde{f}([\mathtt{TX}^P_{\mathtt{RV},i}]))$ if $Q = A$ or compute $\tilde{\sigma}^Q_{\mathtt{TX}^P_{\mathtt{RV},i}} := \mathsf{Sign}_{sk'^Q_{\mathtt{RV}}}(\tilde{f}([\mathtt{TX}^P_{\mathtt{RV},i}]))$ otherwise, output $(\text{revokeQ}, id, \tilde{\sigma}^Q_{\mathtt{TX}^P_{\mathtt{RV},i}}) \xhookrightarrow{t_1+4} P$ and $(\mathtt{UPDATED}, id) \xhookrightarrow{t_1+5} \mathcal{E}$. Else, execute the procedure $\mathtt{ForceClose}^Q(id)$ and stop.

Party P

12. If $(\text{revokeQ}, id, \tilde{\sigma}^Q_{\mathtt{TX}^P_{\mathtt{RV},i}}) \xhookleftarrow{\tau_1+4} Q$, such that it holds that $\mathsf{Vrfy}_{pk^Q_{\mathtt{RV}}}(\tilde{f}([\mathtt{TX}^P_{\mathtt{RV},i}]); \tilde{\sigma}^Q_{\mathtt{TX}^P_{\mathtt{RV},i}}) = 1$ if $P = B$ or $\mathsf{Vrfy}_{pk'^Q_{\mathtt{RV}}}(\tilde{f}([\mathtt{TX}^P_{\mathtt{RV},i}]); \tilde{\sigma}^Q_{\mathtt{TX}^P_{\mathtt{RV},i}}) = 1$ otherwise, assign $\Theta^P(id) := (\tilde{\sigma}^Q_{\mathtt{TX}^P_{\mathtt{RV},i}})$, $\gamma.\mathsf{sn} := i + 1$, $\gamma.\mathsf{st} := \vec{\theta}$, $\Gamma^P(id) := (\gamma, \mathtt{TX}_{\mathtt{FU}}, \mathtt{TX}^P_{\mathtt{CM},i+1}, [\mathtt{TX}^Q_{\mathtt{CM},i+1}], \overline{\mathtt{TX}_{\mathtt{SP},i+1}})$, $\gamma.\mathsf{flag} := 1$, and $\Gamma'^P(id) := (\bot, \bot, \bot)$ and send $(\mathtt{UPDATED}, id) \xhookrightarrow{\tau_1+4} \mathcal{E}$. Else, execute the procedure $\mathtt{ForceClose}^P(id)$ and stop.

Close

Party P upon $(\mathtt{CLOSE}, id) \xhookleftarrow{\tau_0} \mathcal{E}$

1. Extract $\mathtt{TX}_{\mathtt{FU}}$, $i := \gamma.sn$, $\mathtt{TX}_{\mathtt{SP},i}$ from $\Gamma^P(id)$ and create $[\mathtt{TX}_{\overline{\mathtt{SP}}}] := \mathtt{GenFinSplit}(\mathtt{TX}_{\mathtt{FU}}.\mathsf{txid} \| 1, \gamma.\mathsf{st})$.
2. Compute $\sigma^P_{\mathtt{TX}_{\overline{\mathtt{SP}}}} := \mathsf{Sign}_{sk_P}(f([\mathtt{TX}_{\overline{\mathtt{SP}}}]))$ and send $(\text{CloseP}, id, \sigma^P_{\mathtt{TX}_{\overline{\mathtt{SP}}}}) \xhookrightarrow{\tau_0} Q$.
3. If $(\text{CloseQ}, id, \sigma^Q_{\mathtt{TX}_{\overline{\mathtt{SP}}}}) \xhookleftarrow{\tau_0+1} Q$ s.t. $\mathsf{Vrfy}_{pk_Q}(f([\mathtt{TX}_{\overline{\mathtt{SP}}}]); \sigma^Q_{\mathtt{TX}_{\overline{\mathtt{SP}}}}) = 1$, create the transaction $\mathtt{TX}_{\overline{\mathtt{SP}}} := (\mathcal{H}([\mathtt{TX}_{\overline{\mathtt{SP}}}]), [\mathtt{TX}_{\overline{\mathtt{SP}}}], (1, \{\sigma^P_{\mathtt{TX}_{\overline{\mathtt{SP}}}}, \sigma^Q_{\mathtt{TX}_{\overline{\mathtt{SP}}}}\}))$ and send $(\text{post}, \mathtt{TX}_{\overline{\mathtt{SP}}}) \xhookrightarrow{\tau_0+1} \mathcal{L}$. Else, execute the procedure $\mathtt{ForceClose}^P(id)$ and stop.

4. If in round $\tau_1 \leq \tau_0 + 1 + \Delta$, the transaction $\mathtt{TX}_{\overline{\mathtt{SP}}}$ is accepted by $\mathcal{L}$, set $\Gamma^P(id) :=\perp$, $\Theta^P(id) :=\perp$ and send $(\mathtt{CLOSED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$.

Punish (executed at the end of every round τ_0)

Party P

For each $id \in \{0,1\}^*$, extract $i := \gamma.\mathsf{sn}$ and $flag := \gamma.\mathsf{flag}$ from $\Gamma^P(id)$.
If $flag = 1$:

– Set $(\gamma, \mathtt{TX}_{\mathtt{FU}}, \mathtt{TX}^P_{\mathtt{CM},i}, [\mathtt{TX}^Q_{\mathtt{CM},i}], \overline{\mathtt{TX}_{\mathtt{SP},i}}) := \Gamma^P(id)$ and $I := \{[\mathtt{TX}^P_{\mathtt{CM},i}], [\mathtt{TX}^Q_{\mathtt{CM},i}]\}$. Check if $\mathtt{TX}_{\mathtt{FU}}.\mathsf{Output}$ is spent by a transaction $\mathtt{TX}$ s.t. $[\mathtt{TX}] \notin I$. If yes:
  - Create $(\overline{[\mathtt{TX}^P_{\mathtt{RV},i-1}]}, \overline{[\mathtt{TX}^Q_{\mathtt{RV},i-1}]}) := \mathtt{GenRevoke}(pk_P, pk_Q, \gamma.\mathsf{cash}, i)$ and then set $[\mathtt{TX}^P_{\mathtt{RV},i-1}] := (\mathtt{TX}.\mathsf{txid}\|1, \overline{[\mathtt{TX}^P_{\mathtt{RV},i-1}]})$.
  - compute $\tilde{\sigma}^P_{\mathtt{TX}^P_{\mathtt{RV},i-1}} = \mathsf{Sign}_{sk^P_{\mathtt{RV}}}(\tilde{f}(\overline{[\mathtt{TX}^P_{\mathtt{RV},i-1}]}))$ if $P = B$ or $\tilde{\sigma}^P_{\mathtt{TX}^P_{\mathtt{RV},i-1}} = \mathsf{Sign}_{sk'^P_{\mathtt{RV}}}$ $(\tilde{f}(\overline{[\mathtt{TX}^P_{\mathtt{RV},i-1}]}))$ otherwise.
  - Set $\tilde{\sigma}^Q_{\mathtt{TX}^P_{\mathtt{RV},i-1}} := \Theta^P(id)$, and create $\mathtt{TX}^P_{\mathtt{RV},i-1} := (\mathcal{H}([\mathtt{TX}^P_{\mathtt{RV},i-1}]), [\mathtt{TX}^P_{\mathtt{RV},i-1}],$ $(2, \{\tilde{\sigma}^P_{\mathtt{TX}^P_{\mathtt{RV},i-1}}, \tilde{\sigma}^Q_{\mathtt{TX}^P_{\mathtt{RV},i-1}}\}))$.
  - Post $(\text{post}, \mathtt{TX}^P_{\mathtt{RV},i-1}) \xhookrightarrow{\tau_0} \mathcal{L}$.
  - Let $\mathtt{TX}^P_{\mathtt{RV},i-1}$ be accepted by $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + \Delta$. Set $\Theta^P(id) :=\perp$, $\Gamma^P(id) :=\perp$ and output $(\mathtt{PUNISHED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$.

  If no, set $[\mathtt{TX}_{\mathtt{SP},i}] := (\mathtt{TX}.\mathsf{txid}\|1, \overline{\mathtt{TX}_{\mathtt{SP},i}}.\mathsf{nLT}, \overline{\mathtt{TX}_{\mathtt{SP},i}}.\mathsf{Output})$, set $\mathtt{TX}_{\mathtt{SP},i} := (\mathcal{H}$ $([\mathtt{TX}_{\mathtt{SP},i}]), \mathtt{TX}.\mathsf{txid}\|1, \overline{\mathtt{TX}_{\mathtt{SP},i}})$ and then post $(\text{post}, \mathtt{TX}_{\mathtt{SP},i}) \xhookrightarrow{\tau_0+T} \mathcal{L}$. Let $\mathtt{TX}$ be spent in round $\tau_1 \leq \tau_0 + T + \Delta$. Set $\Theta^P(id) :=\perp$, $\Gamma^P(id) :=\perp$ and output $(\mathtt{CLOSED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$.

If $flag = 2$:

– Set $(\gamma, \mathtt{TX}_{\mathtt{FU}}, \mathtt{TX}^P_{\mathtt{CM},i}, [\mathtt{TX}^Q_{\mathtt{CM},i}], \overline{\mathtt{TX}_{\mathtt{SP},i}}) := \Gamma^P(id)$, and $(\mathtt{TX}^P_{\mathtt{CM},i+1}, [\mathtt{TX}^Q_{\mathtt{CM},i+1}], \overline{\mathtt{TX}_{\mathtt{SP},i+1}})$ $:= \Gamma'^P(id)$ and also $I := \{[\mathtt{TX}^P_{\mathtt{CM},i}], [\mathtt{TX}^Q_{\mathtt{CM},i}], [\mathtt{TX}^P_{\mathtt{CM},i+1}], [\mathtt{TX}^Q_{\mathtt{CM},i+1}]\}$. Check if $\mathtt{TX}_{\mathtt{FU}}.\mathsf{Output}$ is spent by a transaction $\mathtt{TX}$ s.t. $[\mathtt{TX}] \notin I$. If yes:
  - Create $(\overline{[\mathtt{TX}^P_{\mathtt{RV},i-1}]}, \overline{[\mathtt{TX}^Q_{\mathtt{RV},i-1}]}) := \mathtt{GenRevoke}(pk_P, pk_Q, \gamma.\mathsf{cash}, i)$. and then set $[\mathtt{TX}^P_{\mathtt{RV},i-1}] := (\mathtt{TX}.\mathsf{txid}\|1, \overline{[\mathtt{TX}^P_{\mathtt{RV},i-1}]})$.
  - compute $\tilde{\sigma}^P_{\mathtt{TX}^P_{\mathtt{RV},i-1}} := \mathsf{Sign}_{sk^P_{\mathtt{RV}}}(\tilde{f}(\overline{[\mathtt{TX}^P_{\mathtt{RV},i-1}]}))$ if $P = B$ or $\tilde{\sigma}^P_{\mathtt{TX}^P_{\mathtt{RV},i-1}} :=$ $\mathsf{Sign}_{sk'^P_{\mathtt{RV}}}(\tilde{f}(\overline{[\mathtt{TX}^P_{\mathtt{RV},i-1}]}))$ otherwise.
  - Set $\tilde{\sigma}^Q_{\mathtt{TX}^P_{\mathtt{RV},i-1}} := \Theta^P(id)$, and create $\mathtt{TX}^P_{\mathtt{RV},i-1} := (\mathcal{H}([\mathtt{TX}^P_{\mathtt{RV},i-1}]), [\mathtt{TX}^P_{\mathtt{RV},i-1}],$ $(2, \{\tilde{\sigma}^P_{\mathtt{TX}^P_{\mathtt{RV},i-1}}, \tilde{\sigma}^Q_{\mathtt{TX}^P_{\mathtt{RV},i-1}}\}))$.
  - Post $(\text{post}, \mathtt{TX}^P_{\mathtt{RV},i-1}) \xhookrightarrow{\tau_0} \mathcal{L}$.
  - Let $\mathtt{TX}^P_{\mathtt{RV},i-1}$ be accepted by $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + \Delta$. Set $\Theta^P(id) :=\perp$, $\Gamma^P(id) :=\perp$ and output $(\mathtt{PUNISHED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$.

If no, set $[\mathtt{TX}_{\mathtt{SP},i+1}] := (\mathtt{TX}.\mathsf{txid}\|1, \overline{\mathtt{TX}_{\mathtt{SP},i+1}}.\mathsf{nLT}, \overline{\mathtt{TX}_{\mathtt{SP},i+1}}.\mathsf{Output})$, $\mathtt{TX}_{\mathtt{SP},i+1} := (\mathcal{H}([\mathtt{TX}_{\mathtt{SP},i+1}]), \mathtt{TX}.\mathsf{txid}\|1, \overline{\mathtt{TX}_{\mathtt{SP},i+1}})$, wait for T rounds and post $(\text{post}, \mathtt{TX}_{\mathtt{SP},i+1}) \xhookrightarrow{\tau_0+T} \mathcal{L}$. Let $\mathtt{TX}_{\mathtt{SP},i+1}$ be accepted by $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + T + \Delta$. Then, set $\Theta^P(id) := \perp$, $\Gamma^P(id) := \perp$ and output $(\mathtt{CLOSED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$.

Subprocedures

$\mathtt{ForceClose}^P(id)$:

Let τ_0 be the current round. Extract $i := \gamma.\mathsf{sn}$, $flag := \gamma.\mathsf{flag}$, $\mathtt{TX}^P_{\mathtt{CM},i}$ and $\overline{\mathtt{TX}_{\mathtt{SP},i}}$ from $\Gamma^P(id)$. Also, extract $\mathtt{TX}^P_{\mathtt{CM},i+1}$ and $\overline{\mathtt{TX}_{\mathtt{SP},i+1}}$ from $\Gamma'^P(id)$ if $flag = 2$.

If i) $flag = 1$ or ii) $flag = 2$ and $\mathtt{TX}^P_{\mathtt{CM},i+1} = \perp$, then post $(\text{post}, \mathtt{TX}^P_{\mathtt{CM},i}) \xhookrightarrow{\tau_0} \mathcal{L}$. Otherwise, post $(\text{post}, \mathtt{TX}^P_{\mathtt{CM},i+1}) \xhookrightarrow{\tau_0} \mathcal{L}$.

(Publishing the corresponding split transaction takes place in the Punish phase.)

$\mathtt{GenFund}((tid_P, tid_Q), \gamma)$:

Return $[\mathtt{TX}_{\mathtt{FU}}]$ where $[\mathtt{TX}_{\mathtt{FU}}].\mathsf{Input} := (tid_P, tid_Q)$, $[\mathtt{TX}_{\mathtt{FU}}].\mathsf{nLT} := 0$ and $[\mathtt{TX}_{\mathtt{FU}}].\mathsf{Output} := (\gamma.\mathsf{Cash}, pk_P \wedge pk_Q)$

$\mathtt{GenCommit}([\mathtt{TX}_{\mathtt{FU}}].\mathsf{txid}\|1, (pk^P_{\mathtt{SP}}, pk^P_{\mathtt{RV}}, pk'^P_{\mathtt{RV}}), (pk^Q_{\mathtt{SP}}, pk^Q_{\mathtt{RV}}, pk'^Q_{\mathtt{RV}}), i)$:

Return $[\mathtt{TX}^P_{\mathtt{CM},i}]$ and $[\mathtt{TX}^Q_{\mathtt{CM},i}]$ where $\mathtt{TX}^P_{\mathtt{CM},i}.\mathsf{nLT} := 0$, $\mathtt{TX}^P_{\mathtt{CM},i}.\mathsf{Input} := \mathtt{TX}_{\mathtt{FU}}.\mathsf{txid}\|1$, and $\mathtt{TX}^P_{\mathtt{CM},i}.\mathsf{Output} := \{(\mathtt{TX}_{\mathtt{FU}}.\mathsf{Output.cash}, (\varphi_1 \vee \varphi_2)\}$, $\mathtt{TX}^Q_{\mathtt{CM},i}.\mathsf{nLT} := 0$, $\mathtt{TX}^Q_{\mathtt{CM},i}.\mathsf{Input} := \mathtt{TX}_{\mathtt{FU}}.\mathsf{txid}\|1$, and $\mathtt{TX}^Q_{\mathtt{CM},i}.\mathsf{Output} := \{(\mathtt{TX}_{\mathtt{FU}}.\mathsf{Output.cash}, (\varphi_1 \vee \varphi'_2)\}$ with $\varphi_1 := (pk^P_{\mathtt{SP}} \wedge pk^Q_{\mathtt{SP}} \wedge CSV_T \wedge CLTV_{S_0+i})$, $\varphi_2 := (pk^P_{\mathtt{RV}} \wedge pk^Q_{\mathtt{RV}} \wedge CLTV_{S_0+i})$, $\varphi'_2 := (pk'^P_{\mathtt{RV}} \wedge pk'^Q_{\mathtt{RV}} \wedge CLTV_{S_0+i})$.

$\mathtt{GenSplit}(\vec{\theta}, i)$:

Return $[\overline{\mathtt{TX}_{\mathtt{SP},i}}]$ where $[\overline{\mathtt{TX}_{\mathtt{SP},i}}].\mathsf{nLT} := S_0 + i$ and $[\overline{\mathtt{TX}_{\mathtt{SP},i}}].\mathsf{Output} := \vec{\theta}$.

$\mathtt{GenRevoke}(pk_P, pk_Q, \gamma.\mathsf{cash}, i+1)$:

Return $\overline{[\mathtt{TX}^P_{\mathtt{RV},i}]}$ and $\overline{[\mathtt{TX}^Q_{\mathtt{RV},i}]}$ where $\overline{[\mathtt{TX}^P_{\mathtt{RV},i}]}.\mathsf{nLT} := S_0 + i$, $\overline{[\mathtt{TX}^P_{\mathtt{RV},i}]}.\mathsf{Output} := \{(\gamma.\mathsf{cash}, pk_P)\}$, $\overline{[\mathtt{TX}^Q_{\mathtt{RV},i}]}.\mathsf{nLT} := S_0 + i$ and $\overline{[\mathtt{TX}^Q_{\mathtt{RV},i}]}.\mathsf{Output} := \{(\gamma.\mathsf{cash}, pk_Q)\}$

E Functionality Wrapper

The functionality $\mathcal{F}$ is supposed to perform several checks once he receives a message from another party. The functionality $\mathcal{F}$ must perform those checks in order to ensure that the received messages are well-formed. The following wrapper summarizes those checks.

Functionality Wrapper:$\mathcal{W}_{\mathcal{F}}$

Create

Upon $(\mathtt{INTRO}, \gamma, tid_P) \overset{\tau_0}{\hookleftarrow} P$ check if: $P \in \gamma.\mathsf{users}$; $\Gamma(\gamma.\mathsf{id}) \neq \perp$; there is no channel γ' with $\gamma.\mathsf{id} = \gamma'.\mathsf{id}$, $\gamma.\mathsf{sn} = 0$; $\gamma.\mathsf{st} = \{(c_P, \mathtt{One}-\mathtt{Sig}_{pk_P}), (c_Q, \mathtt{One}-\mathtt{Sig}_{pk_Q})\}$ with $c_P, c_Q \in \mathbb{R}^{>0}$ and $c_P + c_Q = \gamma.\mathsf{cash}$; there exist $(t, id, i, \theta) \in \mathcal{L}.\mathtt{UTXO}$ such that $\theta = (c_P, \mathtt{One}-\mathtt{Sig}_P)$ with $id \| i = tid$; and none of the other channels that are being created at the moment, must use tid_P. Drop the message if any above checks fails. Else proceed as $\mathcal{F}$.

Upon $(\mathtt{CREATE}, id) \overset{\tau}{\hookleftarrow} P$ check if: you accepted messages $(\mathtt{INTRO}, \gamma, tid_P) \overset{\tau_0}{\hookleftarrow} P$ and $(\mathtt{INTRO}, \gamma, tid_Q) \overset{\tau_0}{\hookleftarrow} Q$ with $P, Q \in \gamma.\mathsf{users}$, $\tau_0 + 1 = \tau$ and $id = \gamma.\mathsf{id}$. Else proceed as $\mathcal{F}$.

Update

Upon $(\mathtt{UPDATE}, id, \overrightarrow{\theta}, t_{stp}) \overset{\tau_0}{\hookleftarrow} P$ set $(\gamma, \mathtt{TX}_{\mathtt{FU}}) := \Gamma(id)$ and check if: $\gamma \neq \perp$, there is no other update being preformed; let $\overrightarrow{\theta} = (\theta_1, \cdots, \theta_l) = ((c_1, \varphi_1), \cdots, (c_l, \varphi_l))$, then $\Sigma_{i \in [l] c_i = \gamma.\mathsf{cash}}$ and $\varphi_i \in \mathcal{L}.\mathcal{V}$. Drop the message if any above checks fails. Else proceed as $\mathcal{F}$.

Upon $(\mathtt{UPDATE}-\mathtt{OK}, id) \overset{\tau}{\hookleftarrow} P$ check if: the message is a reply to the message $(\mathtt{UPDATE}-\mathtt{REQ}, id, \overrightarrow{\theta}, t_{stp})$ sent to P in round τ. If not, drop the message. Else proceed as $\mathcal{F}$.

Upon $(\mathtt{SETUP}-\mathtt{OK}, id) \overset{\tau}{\hookleftarrow} P$ check if: the message is a reply to the message $(\mathtt{SETUP}, id)$ sent to P in round τ_0 where $\tau = \tau_0 + t_{stp}$. If not, drop the message. Else proceed as $\mathcal{F}$.

Upon $(\mathtt{SETUP}'-\mathtt{OK}, id) \overset{\tau}{\hookleftarrow} P$ check if: the message is a reply to the message $(\mathtt{SETUP}', id)$ sent to P in round τ. If not, drop the message. Else proceed as $\mathcal{F}$.

Upon $(\mathtt{REVOKE}, id) \overset{\tau}{\hookleftarrow} P$ check if: the message is a reply to the message $(\mathtt{UPDATE}-\mathtt{OK}, id)$ sent to P in round τ. If not, drop the message. Else proceed as $\mathcal{F}$.

Upon $(\mathtt{REVOKE}', id) \overset{\tau}{\hookleftarrow} P$ check if: the message is a reply to the message $(\mathtt{REVOKE}-\mathtt{REQ}, id)$ sent to P in round τ. If not, drop the message. Else proceed as $\mathcal{F}$.

Close

Upon $(\mathtt{CLOSE}, id) \overset{\tau}{\hookleftarrow} P$, set $(\gamma, \mathtt{TX}_{\mathtt{FU}}) := \Gamma(id)$ and check if: $P \in \gamma.\mathsf{users}$, $\Gamma(\gamma.\mathsf{id}) \neq \perp$ and $\gamma.\mathsf{flag} = 1$. Drop the message if any above checks fails. Else proceed as $\mathcal{F}$.

F Protocol Wrapper

Each party in Daric protocol $\mathcal{F}$ is supposed to perform several checks once he receives a message from another party. Parties perform those checks to ensure that the received messages are well-formed. The following wrapper summarizes those checks.

Protocol Wrapper: $\mathcal{W}_{\mathcal{P}}$

Create

Upon $(\texttt{INTRO}, \gamma, tid_P) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$ check if: $P \in \gamma.\mathsf{users}$; $\Gamma(\gamma.\mathsf{id}) \neq \perp$; there is no channel γ' with $\gamma.\mathsf{id} = \gamma'.\mathsf{id}$, $\gamma.\mathsf{sn} = 0$; $\gamma.\mathsf{st} = \{(c_P, \texttt{One} - \texttt{Sig}_{pk_P}), (c_Q, \texttt{One} - \texttt{Sig}_{pk_Q})\}$ with $c_P, c_Q \in \mathbb{R}^{>0}$ and $c_P + c_Q = \gamma.\mathsf{cash}$; there exist $(t, id, i, \theta) \in \mathcal{L}.\texttt{UTXO}$ such that $\theta = (c_P, \texttt{One} - \texttt{Sig}_P)$ with $id \| i = tid$; and none of the other channels that are being created at the moment, must use tid_P. Drop the message if any above checks fails. Else proceed as P in Daric protocol.

Upon $(\texttt{CREATE}, id) \overset{\tau}{\hookleftarrow} \mathcal{E}$ check if: you accepted messages $(\texttt{INTRO}, \gamma, tid_P) \overset{\tau_0}{\hookleftarrow} P$ and $(\texttt{INTRO}, \gamma, tid_Q) \overset{\tau_0}{\hookleftarrow} Q$ with $P, Q \in \gamma.\mathsf{users}$, $\tau_0 + 1 = \tau$ and $id = \gamma.\mathsf{id}$. Else proceed as P in Daric protocol.

Update

Upon $(\texttt{UPDATE}, id, \vec{\theta}, t_{stp}) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$ set $(\gamma, \texttt{TX}_{\texttt{FU}}) := \Gamma(id)$ and check if: $\gamma \neq \perp$, there is no other update being preformed; let $\vec{\theta} = (\theta_1, \cdots, \theta_l) = ((c_1, \varphi_1), \cdots, (c_l, \varphi_l))$, then $\Sigma_{i \in [l] c_i = \gamma.\mathsf{cash}}$ and $\varphi_i \in \mathcal{L}.\mathcal{V}$. Drop the message if any above checks fails. Else proceed as P in Daric protocol.

Upon $(\texttt{UPDATE} - \texttt{OK}, id) \overset{\tau}{\hookleftarrow} \mathcal{E}$ check if: the message is a reply to the message $(\texttt{UPDATE} - \texttt{REQ}, id, \vec{\theta}, t_{stp})$ sent to $\mathcal{E}$ in round τ. If not, drop the message. Else proceed as P in Daric protocol.

Upon $(\texttt{SETUP} - \texttt{OK}, id) \overset{\tau}{\hookleftarrow} \mathcal{E}$ check if: the message is a reply to the message $(\texttt{SETUP}, id)$ sent to $\mathcal{E}$ in round τ_0 where $\tau = \tau_0 + t_{stp}$. If not, drop the message. Else proceed as P in Daric protocol.

Upon $(\texttt{SETUP}' - \texttt{OK}, id) \overset{\tau}{\hookleftarrow} \mathcal{E}$ check if: the message is a reply to the message $(\texttt{SETUP}', id)$ sent to $\mathcal{E}$ in round τ. If not, drop the message. Else proceed as P in Daric protocol.

Upon $(\texttt{REVOKE}, id) \overset{\tau}{\hookleftarrow} \mathcal{E}$ check if: the message is a reply to the message $(\texttt{UPDATE} - \texttt{OK}, id)$ sent to $\mathcal{E}$ in round τ. If not, drop the message. Else proceed as P in Daric protocol.

Upon $(\texttt{REVOKE}', id) \overset{\tau}{\hookleftarrow} \mathcal{E}$ check if: the message is a reply to the message $(\texttt{REVOKE} - \texttt{REQ}, id)$ sent to $\mathcal{E}$ in round τ. If not, drop the message. Else proceed as P in Daric protocol.

Close

Upon $(\texttt{CLOSE}, id) \overset{\tau}{\hookleftarrow} \mathcal{E}$, set $(\gamma, \texttt{TX}_{\texttt{FU}}) := \Gamma(id)$ and check if: $P \in \gamma.\mathsf{users}$, $\Gamma(\gamma.\mathsf{id}) \neq \perp$ and and $\gamma.\mathsf{flag} = 1$. Drop the message if any above checks fails. Else proceed as P in Daric protocol.

G Security Analysis

In this section, we prove the Theorem 1. To do so, a simulator for the protocol π in the ideal world is provided and then we formally show that the Daric protocol (introduced in Appendix D) UC-realizes the ideal functionality $\mathcal{F}$ (introduced in Section 5.2).

Simulator:

Create

Case A is honest and B is corrupted.

Upon A sending $(\mathtt{INTRO}, \gamma, tid_A) \overset{\tau_0}{\hookrightarrow} \mathcal{F}$,

1. Set $id := \gamma.\mathsf{id}$, generate $(pk^A_{\mathtt{SP}}, sk^A_{\mathtt{SP}}) \leftarrow \mathsf{Gen}$, $(pk^A_{\mathtt{RV}}, sk^A_{\mathtt{RV}}) \leftarrow \mathsf{Gen}$ and $(pk'^A_{\mathtt{RV}}, sk'^A_{\mathtt{RV}}) \leftarrow \mathsf{Gen}$ and send $(\text{createInfo}, id, tid_A, pk^A_{\mathtt{SP}}, pk^A_{\mathtt{RV}}, pk'^A_{\mathtt{RV}}) \overset{\tau_0}{\hookrightarrow} B$.
2. If B sends $(\text{createInfo}, id, tid_B, pk^B_{\mathtt{SP}}, pk^B_{\mathtt{RV}}, pk'^B_{\mathtt{RV}}) \overset{\tau_0}{\hookrightarrow} A$, then $(\mathtt{INTRO}, \gamma, tid_B) \overset{\tau_0}{\hookrightarrow} \mathcal{F}$ on behalf of B. Else stop.
3. If A sends $(\mathtt{CREATE}, id) \xhookrightarrow{\tau_0+1} \mathcal{F}$, then create $[\mathtt{TX}_{\mathtt{FU}}] := \mathtt{GenFund}((tid_A, tid_B), \gamma)$, $([\mathtt{TX}^A_{\mathtt{CM},0}], [\mathtt{TX}^B_{\mathtt{CM},0}]) := \mathtt{GenCommit}([\mathtt{TX}_{\mathtt{FU}}].\mathsf{txid}\|1, I_A, I_B, 0)$, and $\overline{[\mathtt{TX}_{\mathtt{SP},0}]} := \mathtt{GenSplit}(\gamma.\mathsf{st}, 0)$ for $I_A := (pk^A_{\mathtt{SP}}, pk^A_{\mathtt{RV}}, pk'^A_{\mathtt{RV}})$ and $I_B := (pk^B_{\mathtt{SP}}, pk^B_{\mathtt{RV}}, pk'^B_{\mathtt{RV}})$. Else stop.
4. Compute $\tilde{\sigma}^A_{\mathtt{TX}_{\mathtt{SP},0}} = \mathsf{Sign}_{sk^A_{\mathtt{SP}}}(\tilde{f}(\overline{[\mathtt{TX}_{\mathtt{SP},0}]}))$ and $\sigma^A_{\mathtt{TX}^B_{\mathtt{CM},0}} = \mathsf{Sign}_{sk^A_{\mathtt{CM}}}(f([\mathtt{TX}^B_{\mathtt{CM},0}]))$ and send $(\text{createCom}, id, \tilde{\sigma}^A_{\mathtt{TX}_{\mathtt{SP},0}}, \sigma^A_{\mathtt{TX}^B_{\mathtt{CM},0}}) \xhookrightarrow{\tau_0+1} B$.
5. If B sends $(\text{createCom}, id, \tilde{\sigma}^B_{\mathtt{TX}_{\mathtt{SP},0}}, \sigma^B_{\mathtt{TX}^A_{\mathtt{CM},0}}) \xhookrightarrow{\tau_0+1} A$, s.t. $\mathsf{Vrfy}_{pk^B_{\mathtt{SP}}}(\tilde{f}(\overline{[\mathtt{TX}_{\mathtt{SP},0}]}); \tilde{\sigma}^B_{\mathtt{TX}_{\mathtt{SP},0}}) = 1$ and $\mathsf{Vrfy}_{pk^B_{\mathtt{CM}}}(f([\mathtt{TX}^A_{\mathtt{CM},0}]); \sigma^B_{\mathtt{TX}^A_{\mathtt{CM},0}}) = 1$, send $(\mathtt{CREATE}, id) \xhookrightarrow{\tau_0+1} \mathcal{F}$ on behalf of B. Else stop.
6. Compute $\sigma^A_{\mathtt{TX}_{\mathtt{FU}}} = \mathsf{Sign}_{sk_A}(f([\mathtt{TX}_{\mathtt{FU}}]))$ and send $(\text{createFund}, id, \sigma^A_{\mathtt{TX}_{\mathtt{FU}}}) \xhookrightarrow{\tau_0+2} B$.
7. If B sends $(\text{createFund}, id, \sigma^B_{\mathtt{TX}_{\mathtt{FU}}}) \xhookrightarrow{\tau_0+2} A$, s.t. $\mathsf{Vrfy}_{pk_B}(f([\mathtt{TX}_{\mathtt{FU}}]); \sigma^B_{\mathtt{TX}_{\mathtt{FU}}}) = 1$, create $\mathtt{TX}_{\mathtt{FU}} := (\mathcal{H}([\mathtt{TX}_{\mathtt{FU}}]), [\mathtt{TX}_{\mathtt{FU}}], ((x, \sigma^A_{\mathtt{TX}_{\mathtt{FU}}}), (y, \sigma^B_{\mathtt{TX}_{\mathtt{FU}}})))$ and $(\text{post}, \mathtt{TX}_{\mathtt{FU}}) \xhookrightarrow{\tau_0+3} \mathcal{L}$. Else create a transaction $\mathtt{TX}$ with $\mathtt{TX}.\mathsf{input} := tid_A$ and $\mathtt{TX}.\mathsf{Output}.\varphi := pk_A$ and $(\text{post}, \mathtt{TX}) \xhookrightarrow{\tau_0+3} \mathcal{L}$.
8. If $\mathtt{TX}_{\mathtt{FU}}$ is accepted by $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + 3 + \Delta$, compute $\sigma^A_{\mathtt{TX}^A_{\mathtt{CM},0}} = \mathsf{Sign}_{sk^A_{\mathtt{CM}}}(f([\mathtt{TX}^A_{\mathtt{CM},0}]))$, create $\mathtt{TX}^A_{\mathtt{CM},0} := (\mathcal{H}([\mathtt{TX}^A_{\mathtt{CM},0}]), [\mathtt{TX}^A_{\mathtt{CM},0}], (1, \{\sigma^A_{\mathtt{TX}^A_{\mathtt{CM},0}}, \sigma^B_{\mathtt{TX}^A_{\mathtt{CM},0}}\})$, set $\overline{\mathtt{TX}_{\mathtt{SP},0}} := (\overline{[\mathtt{TX}_{\mathtt{SP},0}]}, (1, \{\tilde{\sigma}^A_{\mathtt{TX}_{\mathtt{SP},0}}, \tilde{\sigma}^B_{\mathtt{TX}_{\mathtt{SP},0}}\}))$, store $\mathit{\Gamma}^A(\gamma.id) := (\gamma, \mathtt{TX}_{\mathtt{FU}}, \mathtt{TX}^A_{\mathtt{CM},0}, [\mathtt{TX}^B_{\mathtt{CM},0}], \overline{\mathtt{TX}_{\mathtt{SP},0}})$ and $(\mathtt{CREATED}, id) \overset{\tau_1}{\hookrightarrow} \mathcal{E}$. Else $\mathit{\Gamma}^A(\gamma.id) := (\perp, \mathtt{TX}_{\mathtt{FU}}, \perp, \perp, \perp)$ and stop.

Update

Case A is honest and B is corrupted.

Upon A sending $(\mathtt{UPDATE}, id, \overrightarrow{\theta}, t_{stp}) \overset{\tau_0}{\hookrightarrow} \mathcal{F}$, proceed as follows:

1. Send $(\text{updateReq}, id, \overrightarrow{\theta}, t_{stp}) \xhookrightarrow{\tau_0} B$.
2. If B sends $(\text{updateInfo}, id, \tilde{\sigma}^{B}_{\mathtt{SP},i+1}) \xhookrightarrow{\tau_1 \leq \tau_0+1+t_{stp}} A$, extract $\mathtt{TX}_{\mathtt{FU}}$ and $i := \gamma.sn$ from $\Gamma^{A}(id)$ and create $([\mathtt{TX}^{A}_{\mathtt{CM},i+1}], [\mathtt{TX}^{B}_{\mathtt{CM},i+1}]) := \mathtt{GenCommit}([\mathtt{TX}_{\mathtt{FU}}].\mathsf{txid}\|1, I_A, I_B, i{+}1)$, and $\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]} := \mathtt{GenSplit}(\gamma.\mathsf{st}, i{+}1)$ for $I_A := (pk^{A}_{\mathtt{SP}}, pk^{A}_{\mathtt{RV}}, pk'^{A}_{\mathtt{RV}})$ and $I_B := (pk^{B}_{\mathtt{SP}}, pk^{B}_{\mathtt{RV}}, pk'^{B}_{\mathtt{RV}})$. If $\mathsf{Vrfy}_{pk^{B}_{\mathtt{SP}}}(\tilde{f}(\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]}); \tilde{\sigma}^{B}_{\mathtt{TX}_{\mathtt{SP},i+1}}) = 1$, then compute $\tilde{\sigma}^{A}_{\mathtt{TX}_{\mathtt{SP},i+1}} := \mathsf{Sign}_{sk^{A}_{\mathtt{SP}}}(\tilde{f}(\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]}))$, set $\overline{\mathtt{TX}_{\mathtt{SP},i+1}} := (\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]}, (1, \{\tilde{\sigma}^{A}_{\mathtt{TX}_{\mathtt{SP},i+1}}, \tilde{\sigma}^{B}_{\mathtt{TX}_{\mathtt{SP},i+1}}\}))$, store $\Gamma'^{A}(id) := (\bot, [\mathtt{TX}^{B}_{\mathtt{CM},i+1}], \overline{\mathtt{TX}_{\mathtt{SP},i+1}})$, set $\gamma.\mathsf{flag} := 2$ and $\gamma.\mathsf{st}' := \overrightarrow{\theta}$ and send $(\mathtt{UPDATE-OK}, id) \xhookrightarrow{\tau_1} \mathcal{F}$ on behalf of B. Else stop.
3. If A sends $(\mathtt{SETUP-OK}, id) \xhookrightarrow{\tau_1+1} \mathcal{F}$, compute $\sigma^{A}_{\mathtt{TX}^{B}_{\mathtt{CM},i+1}} := \mathsf{Sign}_{sk^{A}_{\mathtt{SP}}}(f([\mathtt{TX}^{B}_{\mathtt{CM},i+1}]))$ and send $(\text{updateComA}, id, \tilde{\sigma}^{A}_{\mathtt{TX}_{\mathtt{SP},i+1}}, \sigma^{A}_{\mathtt{TX}^{B}_{\mathtt{CM},i+1}}) \xhookrightarrow{\tau_1+1} B$. Else, execute the procedure $\mathtt{ForceClose}^{A}(id)$ and stop.
4. If B sends $(\text{updateComB}, id, \sigma^{B}_{\mathtt{TX}^{A}_{\mathtt{CM},i+1}}) \xhookrightarrow{\tau_1+2} A$, s.t. $\mathsf{Vrfy}_{pk^{B}_{\mathtt{CM}}}(f([\mathtt{TX}^{A}_{\mathtt{CM},i+1}]); \sigma^{B}_{\mathtt{TX}^{A}_{\mathtt{CM},i+1}}) = 1$, compute $\sigma^{A}_{\mathtt{TX}^{A}_{\mathtt{CM},i+1}} = \mathsf{Sign}_{sk^{A}_{\mathtt{SP}}}(f([\mathtt{TX}^{A}_{\mathtt{CM},i+1}]))$, set $\mathtt{TX}^{A}_{\mathtt{CM},i+1} := (\mathcal{H}([\mathtt{TX}^{A}_{\mathtt{CM},i+1}]), [\mathtt{TX}^{A}_{\mathtt{CM},i+1}], (1, \{\sigma^{A}_{\mathtt{TX}^{A}_{\mathtt{CM},i+1}}, \sigma^{B}_{\mathtt{TX}^{A}_{\mathtt{CM},i+1}}\}))$, store $\Gamma'^{A}(id)[1] := \mathtt{TX}^{A}_{\mathtt{CM},i+1}$, and send $(\mathtt{SETUP'-OK}, id) \xhookrightarrow{\tau_1+2} \mathcal{F}$ on behalf of B. Else, execute the procedure $\mathtt{ForceClose}^{A}(id)$ and stop.
5. If A sends $(\mathtt{REVOKE}, id) \xhookrightarrow{\tau_1+3} \mathcal{F}$, then create $(\overline{[\mathtt{TX}^{A}_{\mathtt{RV},i}]}, \overline{[\mathtt{TX}^{B}_{\mathtt{RV},i}]}) := \mathtt{GenRevoke}$ $(pk_A, pk_B, \gamma.\mathsf{cash}, i+1)$, compute $\tilde{\sigma}^{A}_{\mathtt{TX}^{B}_{\mathtt{RV},i}} = \mathsf{Sign}_{sk^{A}_{\mathtt{RV}}}(\tilde{f}(\overline{[\mathtt{TX}^{B}_{\mathtt{RV},i}]}))$ and send $(\text{revokeA}, id, \tilde{\sigma}^{A}_{\mathtt{TX}^{B}_{\mathtt{RV},i}}) \xhookrightarrow{\tau_1+3} B$. Else, execute the procedure $\mathtt{ForceClose}^{A}(id)$ and stop.
6. If B sends $(\text{revokeB}, id, \tilde{\sigma}^{B}_{\mathtt{TX}^{A}_{\mathtt{RV},i}}) \xhookrightarrow{\tau_1+4} A$, s.t. $\mathsf{Vrfy}_{pk^{B}_{\mathtt{RV}}}(\tilde{f}([\mathtt{TX}^{A}_{\mathtt{RV},i+1}]); \tilde{\sigma}^{B}_{\mathtt{TX}^{A}_{\mathtt{RV},i}}) = 1$, then set $\Theta^{A}(id) := (\tilde{\sigma}^{B}_{\mathtt{TX}^{A}_{\mathtt{RV},i}})$, $\gamma.\mathsf{sn} := i+1$, $\gamma.\mathsf{st} := \overrightarrow{\theta}$, $\Gamma^{A}(id) := (\gamma, \mathtt{TX}_{\mathtt{FU}}, \mathtt{TX}^{A}_{\mathtt{CM},i+1}, [\mathtt{TX}^{B}_{\mathtt{CM},i+1}], \overline{\mathtt{TX}_{\mathtt{SP},i+1}})$, $\gamma.\mathsf{flag} = 1$, and $\Gamma'^{A}(id) = (\bot, \bot, \bot)$ and send $(\mathtt{REVOKE'}, id) \xhookrightarrow{\tau_1+4} \mathcal{F}$ on behalf of B. Else, execute $\mathtt{ForceClose}^{A}(id)$ and stop.

Case B is honest and A is corrupted.

Upon A sending $(\mathit{updateReq}, id, \overrightarrow{\theta}, t_{stp}) \xhookrightarrow{\tau_0} B$, proceed as follows:

1. Send $(\mathtt{UPDATE}, id, \overrightarrow{\theta}, t_{stp}) \xhookrightarrow{\tau_0} \mathcal{F}$ on behalf of A.
2. If B sends $(\mathtt{UPDATE-OK}, id) \xhookrightarrow{\tau_1 \leq \tau_0+1+t_{stp}} \mathcal{F}$, extract $\mathtt{TX}_{\mathtt{FU}}$ and $i := \gamma.sn$ from $\Gamma^{B}(id)$, create $([\mathtt{TX}^{A}_{\mathtt{CM},i+1}], [\mathtt{TX}^{B}_{\mathtt{CM},i+1}]) := \mathtt{GenCommit}([\mathtt{TX}_{\mathtt{FU}}].\mathsf{txid}\|1, I_A, I_B, i+1)$, and $\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]} := \mathtt{GenSplit}(\gamma.\mathsf{st}, i+1)$ for $I_A := (pk^{A}_{\mathtt{SP}}, pk^{A}_{\mathtt{RV}}, pk'^{A}_{\mathtt{RV}})$ and $I_B := (pk^{B}_{\mathtt{SP}}, pk^{B}_{\mathtt{RV}}, pk'^{B}_{\mathtt{RV}})$, compute $\tilde{\sigma}^{B}_{\mathtt{TX}_{\mathtt{SP},i+1}} = \mathsf{Sign}_{sk^{B}_{\mathtt{SP}}}(\tilde{f}(\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]}))$ and send $(\text{updateInfo}, id, \tilde{\sigma}^{B}_{\mathtt{TX}_{\mathtt{SP},i+1}}) \xhookrightarrow{\tau_1} A$.

3. If A sends $(\text{updateComA}, id, \tilde{\sigma}^A_{\mathtt{TX}_{\mathtt{SP},i+1}}, \sigma^A_{\mathtt{TX}^B_{\mathtt{CM},i+1}}) \xhookrightarrow{\tau_1+1} B$, such that $\mathsf{Vrfy}_{pk^A_{\mathtt{SP}}}(\tilde{f}(\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]}); \tilde{\sigma}^A_{\mathtt{TX}_{\mathtt{SP},i+1}}) = 1$ and $\mathsf{Vrfy}_{pk^A_{\mathtt{CM}}}(f([\mathtt{TX}^B_{\mathtt{CM},i+1}]); \sigma^A_{\mathtt{TX}^B_{\mathtt{CM},i+1}}) = 1$, compute $\sigma^B_{\mathtt{TX}^B_{\mathtt{CM},i+1}} = \mathsf{Sign}_{sk^B_{\mathtt{CM}}}(\tilde{f}([\mathtt{TX}^B_{\mathtt{CM},i+1}]))$, set $\overline{\mathtt{TX}_{\mathtt{SP},i+1}} = (\overline{[\mathtt{TX}_{\mathtt{SP},i+1}]}, (1, \{\tilde{\sigma}^A_{\mathtt{TX}_{\mathtt{SP},i+1}}, \tilde{\sigma}^B_{\mathtt{TX}_{\mathtt{SP},i+1}}\}))$, $\mathtt{TX}^B_{\mathtt{CM},i+1} = ([\mathtt{TX}^B_{\mathtt{CM},i+1}], (1, \{\sigma^A_{\mathtt{TX}^B_{\mathtt{CM},i+1}}, \sigma^B_{\mathtt{TX}^B_{\mathtt{CM},i+1}}\}))$, $\gamma.\mathsf{flag} = 2$ and $\gamma.\mathsf{st}' = \overrightarrow{\theta}$, store $\Gamma'^B(id) = (\mathtt{TX}^B_{\mathtt{CM},i+1}, [\mathtt{TX}^A_{\mathtt{CM},i+1}], \overline{\mathtt{TX}_{\mathtt{SP},i+1}})$, set $\gamma.\mathsf{flag} = 2$ and $\gamma.\mathsf{st}' = \overrightarrow{\theta}$ and send $(\mathtt{SETUP-OK}, id) \xhookrightarrow{\tau_1+1} \mathcal{F}$ on behalf of A. Else execute the procedure $\mathtt{ForceClose}^B(id)$ and stop.
4. If $(\mathtt{SETUP'-OK}, id) \xhookleftarrow{\tau_1+2} \mathcal{B}$, compute $\sigma^B_{\mathtt{TX}^A_{\mathtt{CM},i+1}} = \mathsf{Sign}_{sk^B_{\mathtt{SP}}}(\tilde{f}([\mathtt{TX}^A_{\mathtt{CM},i+1}]))$ and send $(\text{updateComB}, id, \sigma^B_{\mathtt{TX}^A_{\mathtt{CM},i+1}}) \xhookrightarrow{\tau_1+2} A$. Else, execute $\mathtt{ForceClose}^B(id)$ and stop.
5. Create $(\overline{[\mathtt{TX}^A_{\mathtt{RV},i}]}, \overline{[\mathtt{TX}^B_{\mathtt{RV},i}]}) := \mathtt{GenRevoke}(pk_A, pk_B, \gamma.\mathsf{cash}, i+1)$. If A sends $(\text{revokeA}, id, \tilde{\sigma}^A_{\mathtt{TX}^B_{\mathtt{RV},i}}) \xhookrightarrow{\tau_1+3} B$, s.t. $\mathsf{Vrfy}_{pk^A_{\mathtt{RV}}}(\tilde{f}([\mathtt{TX}^B_{\mathtt{RV},i+1}]); \tilde{\sigma}^A_{\mathtt{TX}^B_{\mathtt{RV},i+1}}) = 1$, set $\Theta^B(id) := (\tilde{\sigma}^A_{\mathtt{TX}^B_{\mathtt{RV},i}})$, $\gamma.\mathsf{sn} := i+1$, $\gamma.\mathsf{st} := \overrightarrow{\theta}$, $\Gamma^B(id) := (\gamma, \mathtt{TX}_{\mathtt{FU}}, \mathtt{TX}^B_{\mathtt{CM},i+1}, [\mathtt{TX}^A_{\mathtt{CM},i+1}], \mathtt{TX}_{\mathtt{SP},i+1})$, $\gamma.\mathsf{flag} = 1$, and $\Gamma'^B(id) = (\perp, \perp, \perp)$ and send $(\mathtt{REVOKE}, id) \xhookrightarrow{\tau_1+3} \mathcal{F}$ on behalf of A. Else, execute the procedure $\mathtt{ForceClose}^B(id)$ and stop.
6. If B sends $(\mathtt{REVOKE'}, id) \xhookrightarrow{\tau_1+4} \mathcal{F}$, compute $\tilde{\sigma}^B_{\mathtt{TX}^A_{\mathtt{RV},i}} = \mathsf{Sign}_{sk^B_{\mathtt{RV}}}(\tilde{f}([\mathtt{TX}^A_{\mathtt{RV},i}], 1))$, send $(\text{revokeB}, id, \tilde{\sigma}^B_{\mathtt{TX}^A_{\mathtt{RV},i}}) \xhookrightarrow{\tau_1+4} A$. Else, execute $\mathtt{ForceClose}^B(id)$ and stop.

Close

Case A is honest and B is corrupted.

1. Upon A sending $(\mathtt{CLOSE}, id) \xhookrightarrow{\tau_0} \mathcal{F}$, extract $\mathtt{TX}_{\mathtt{FU}}$, $i := \gamma.sn$, $\mathtt{TX}_{\mathtt{SP},i}$ from $\Gamma^A(id)$ and create $[\mathtt{TX}_{\overline{\mathtt{SP}}}] := \mathtt{GenFinSplit}(\mathtt{TX}_{\mathtt{FU}}.\mathsf{txid} \| 1, \gamma.\mathsf{st}))$.
2. Compute $\sigma^A_{\mathtt{TX}_{\overline{\mathtt{SP}}}} := \mathsf{Sign}_{sk^A_{\mathtt{CM}}}(f(\overline{[\mathtt{TX}_{\overline{\mathtt{SP}}}]}))$ and send $(\text{CloseA}, id, \sigma^A_{\mathtt{TX}_{\overline{\mathtt{SP}}}}) \xhookrightarrow{\tau_0} B$.
3. If B sends $(\text{CloseB}, id, \sigma^B_{\mathtt{TX}_{\overline{\mathtt{SP}}}}) \xhookrightarrow{\tau_0} A$ s.t. $\mathsf{Vrfy}_{pk^B_{\mathtt{CM}}}(f([\mathtt{TX}_{\overline{\mathtt{SP}}}]); \sigma^B_{\mathtt{TX}_{\overline{\mathtt{SP}}}}) = 1$, then send $(\mathtt{CLOSE}, id) \xhookrightarrow{\tau_0} \mathcal{F}$ on behalf of B. Otherwise, execute the simulator code of the procedure $\mathtt{ForceClose}^A(id)$ and stop.
4. Create $\mathtt{TX}_{\overline{\mathtt{SP}}} := ([\mathtt{TX}_{\overline{\mathtt{SP}}}], (1, \{\sigma^A_{\mathtt{TX}_{\overline{\mathtt{SP}}}}, \sigma^B_{\mathtt{TX}_{\overline{\mathtt{SP}}}}\}))$ and send $(\text{post}, \mathtt{TX}_{\overline{\mathtt{SP}}}) \xhookrightarrow{\tau_0+1} \mathcal{L}$.
5. If $\tau_1 \leq \tau_0 + 1 + \Delta$ is the round in which $\mathtt{TX}_{\overline{\mathtt{SP}}}$ is accepted by $\mathcal{L}$, set $\Gamma^A(id) = \perp$, $\Theta^A(id) = \perp$.

Punish

Case A is honest and B is corrupted.

For each $id \in \{0,1\}^*$, extract $i := \gamma.\mathsf{sn}$ and $flag := \gamma.\mathsf{flag}$ from $\Gamma^A(id)$.

If $flag = 1$:

- Parse $(\gamma, \mathtt{TX}_{\mathtt{FU}}, \mathtt{TX}^A_{\mathtt{CM},i}, [\mathtt{TX}^B_{\mathtt{CM},i}], \overline{\mathtt{TX}_{\mathtt{SP},i}}) := \Gamma^A(id)$ and set $I := \{[\mathtt{TX}^A_{\mathtt{CM},i}], [\mathtt{TX}^B_{\mathtt{CM},i}]\}$. Check if $\mathtt{TX}_{\mathtt{FU}}.\mathsf{Output}$ is spent by a transaction $\mathtt{TX}$ s.t. $[\mathtt{TX}] \notin I$ and $\mathtt{TX}.\mathsf{Output} \neq \gamma.\mathsf{st}$. If yes:
  1. Create $(\overline{[\mathtt{TX}^A_{\mathtt{RV},i-1}]}, \overline{[\mathtt{TX}^B_{\mathtt{RV},i-1}]}) := \mathtt{GenRevoke}(pk^A_{\mathtt{SP}}, pk^B_{\mathtt{SP}}, \gamma.\mathsf{cash}, i)$ and then set $[\mathtt{TX}^A_{\mathtt{RV},i-1}] := (\mathtt{TX}.\mathsf{txid}\|1, \overline{[\mathtt{TX}^A_{\mathtt{RV},i-1}]})$.
  2. compute $\tilde{\sigma}^A_{\mathtt{TX}^A_{\mathtt{RV},i-1}} = \mathsf{Sign}_{sk'^A_{\mathtt{RV}}}(\tilde{f}(\overline{[\mathtt{TX}^A_{\mathtt{RV},i-1}]}))$ (for the case where B is honest and A is corrupted, $sk^B_{\mathtt{RV}}$ is used to compute the signature).
  3. Set $\tilde{\sigma}^B_{\mathtt{TX}^A_{\mathtt{RV},i-1}} := \Theta^A(id)$, and create $\mathtt{TX}^A_{\mathtt{RV},i-1} := (\mathcal{H}([\mathtt{TX}^A_{\mathtt{RV},i-1}]), [\mathtt{TX}^A_{\mathtt{RV},i-1}], (1, \{\tilde{\sigma}^A_{\mathtt{TX}^A_{\mathtt{RV},i-1}}, \tilde{\sigma}^B_{\mathtt{TX}^A_{\mathtt{RV},i-1}}\}))$.
  4. Post $(\text{post}, \mathtt{TX}^A_{\mathtt{RV},i-1}) \xhookrightarrow{\tau_0} \mathcal{L}$.
  5. Let $\mathtt{TX}^A_{\mathtt{RV},i-1}$ be accepted by $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + \Delta$. Set $\Theta^A(id) :=\perp$, $\Gamma^A(id) :=\perp$ and output $(\mathtt{PUNISHED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$.

  Otherwise, if $\mathtt{TX}.\mathsf{Output} = \gamma.\mathsf{st}$, then set $\Theta^A(id) :=\perp$, $\Gamma^A(id) :=\perp$ and output $(\mathtt{CLOSED}, id) \xhookrightarrow{\tau_0} \mathcal{E}$. Else, set $[\mathtt{TX}_{\mathtt{SP},i}] := (\mathtt{TX}.\mathsf{txid}\|1, \overline{\mathtt{TX}_{\mathtt{SP},i}}.\mathsf{nLT}, \overline{\mathtt{TX}_{\mathtt{SP},i}}.\mathsf{Output})$, $\mathtt{TX}_{\mathtt{SP},i} := (\mathcal{H}([\mathtt{TX}_{\mathtt{SP},i}]), \mathtt{TX}.\mathsf{txid}\|1, \overline{\mathtt{TX}_{\mathtt{SP},i}})$ and post $(\text{post}, \mathtt{TX}_{\mathtt{SP},i}) \xhookrightarrow{\tau_0+T} \mathcal{L}$. Let $\mathtt{TX}_{\mathtt{SP},i}$ be accepted by $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + T + \Delta$. Set $\Theta^A(id) :=\perp$, $\Gamma^A(id) :=\perp$ and output $(\mathtt{CLOSED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$.

If $flag = 2$:

- Parse $(\gamma, \mathtt{TX}_{\mathtt{FU}}, \mathtt{TX}^A_{\mathtt{CM},i}, [\mathtt{TX}^B_{\mathtt{CM},i}], \overline{\mathtt{TX}_{\mathtt{SP},i}}) := \Gamma^A(id)$ as well as $(\mathtt{TX}^A_{\mathtt{CM},i+1}, [\mathtt{TX}^B_{\mathtt{CM},i+1}], \overline{\mathtt{TX}_{\mathtt{SP},i+1}}) := \Gamma'^A(id)$ and set $I = \{[\mathtt{TX}^A_{\mathtt{CM},i}], [\mathtt{TX}^B_{\mathtt{CM},i}], [\mathtt{TX}^A_{\mathtt{CM},i+1}], [\mathtt{TX}^B_{\mathtt{CM},i+1}]\}$. Check if $\mathtt{TX}_{\mathtt{FU}}.\mathsf{Output}$ is spent by a transaction $\mathtt{TX}$ s.t. $[\mathtt{TX}] \notin I$, $\mathtt{TX}.\mathsf{Output} \neq \gamma.\mathsf{st}$, and $\mathtt{TX}.\mathsf{Output} \neq \gamma.\mathsf{st}'$. If yes:
  1. Create $(\overline{[\mathtt{TX}^A_{\mathtt{RV},i-1}]}, \overline{[\mathtt{TX}^B_{\mathtt{RV},i-1}]}) := \mathtt{GenRevoke}(pk_A, pk_B, \gamma.\mathsf{cash}, i)$ and then set $[\mathtt{TX}^A_{\mathtt{RV},i-1}] := (\mathtt{TX}.\mathsf{txid}\|1, \overline{[\mathtt{TX}^A_{\mathtt{RV},i-1}]})$.
  2. compute $\tilde{\sigma}^A_{\mathtt{TX}^A_{\mathtt{RV},i-1}} := \mathsf{Sign}_{sk'^A_{\mathtt{RV}}}(\tilde{f}(\overline{[\mathtt{TX}^A_{\mathtt{RV},i-1}]}))$ (for the case where B is honest and A is corrupted, $sk^B_{\mathtt{RV}}$ is used to compute the signature).
  3. Set $\tilde{\sigma}^B_{\mathtt{TX}^A_{\mathtt{RV},i-1}} := \Theta^A(id)$, and create $\mathtt{TX}^A_{\mathtt{RV},i-1} := (\mathcal{H}([\mathtt{TX}^A_{\mathtt{RV},i-1}]), [\mathtt{TX}^A_{\mathtt{RV},i-1}], (2, \{\tilde{\sigma}^A_{\mathtt{TX}^A_{\mathtt{RV},i-1}}, \tilde{\sigma}^B_{\mathtt{TX}^A_{\mathtt{RV},i-1}}\}))$.
  4. Post $(\text{post}, \mathtt{TX}^A_{\mathtt{RV},i-1}) \xhookrightarrow{\tau_0} \mathcal{L}$.
  5. Let $\mathtt{TX}^A_{\mathtt{RV},i-1}$ be accepted by $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + \Delta$. Set $\Theta^A(id) :=\perp$, $\Gamma^A(id) :=\perp$ and output $(\mathtt{PUNISHED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$.

  Otherwise, if $\mathtt{TX}.\mathsf{Output} = \gamma.\mathsf{st}$ or $\mathtt{TX}.\mathsf{Output} = \gamma.\mathsf{st}'$ hold, then set $\Theta^A(id) :=\perp$, $\Gamma^A(id) :=\perp$ and output $(\mathtt{CLOSED}, id) \xhookrightarrow{\tau_0} \mathcal{E}$. Else, set $[\mathtt{TX}_{\mathtt{SP},i+1}] = (\mathtt{TX}.\mathsf{txid}\|1, \overline{\mathtt{TX}_{\mathtt{SP},i+1}}.\mathsf{nLT}, \overline{\mathtt{TX}_{\mathtt{SP},i+1}}.\mathsf{Output})$, $\mathtt{TX}_{\mathtt{SP},i+1} := (\mathcal{H}([\mathtt{TX}_{\mathtt{SP},i+1}]), \mathtt{TX}.\mathsf{txid}\|1, \overline{\mathtt{TX}_{\mathtt{SP},i+1}})$ and post $(\text{post}, \mathtt{TX}_{\mathtt{SP},i+1}) \xhookrightarrow{\tau_0+T} \mathcal{L}$. Let $\mathtt{TX}_{\mathtt{SP},i+1}$ be accepted by $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + T + \Delta$. Then, set $\Theta^P(id) :=\perp$, $\Gamma^A(id) :=\perp$ and $(\mathtt{CLOSED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$.

Subprocedure $\mathtt{ForceClose}^P(id)$

Let τ_0 be the current round. Extract $i := \gamma.\mathsf{sn}$, $flag := \gamma.\mathsf{flag}$, $\mathtt{TX}^{P}_{\mathsf{CM},i}$ and $\overline{\mathtt{TX}_{\mathsf{SP},i}}$ from $\Gamma^P(id)$ and $\mathtt{TX}^{P}_{\mathsf{CM},i+1}$ and $\overline{\mathtt{TX}_{\mathsf{SP},i+1}}$ from $\Gamma'^P(id)$.

If $flag = 1$:

1. Post $(\text{post}, \mathtt{TX}^{P}_{\mathsf{CM},i}) \xhookrightarrow{\tau_0} \mathcal{L}$.
2. Let $\tau_1 \leq \tau_0 + \Delta$ be the round in which $\mathtt{TX}^{P}_{\mathsf{CM},i}$ is accepted by the blockchain. Wait for T rounds, set $[\mathtt{TX}_{\mathsf{SP},i}] = (\mathtt{TX}^{P}_{\mathsf{CM},i}.\mathsf{txid}\|1, \overline{\mathtt{TX}_{\mathsf{SP},i}}.\mathsf{nLT}, \overline{\mathtt{TX}_{\mathsf{SP},i}}.\mathsf{Output})$, $\mathtt{TX}_{\mathsf{SP},i} := (\mathcal{H}([\mathtt{TX}_{\mathsf{SP},i}]), \mathtt{TX}^{P}_{\mathsf{CM},i}.\mathsf{txid}\|1, \overline{\mathtt{TX}_{\mathsf{SP},i}})$ and post $(\text{post}, \mathtt{TX}_{\mathsf{SP},i}) \xhookrightarrow{\tau_2 := \tau_1 + T} \mathcal{L}$.
3. Once $\mathtt{TX}_{\mathsf{SP},i}$ is accepted by $\mathcal{L}$ in round $\tau_3 \leq \tau_2 + \Delta$ set $\Theta^P(id) :=\perp$ and $\Gamma^P(id) :=\perp$ and output $(\mathtt{CLOSED}, id) \xhookrightarrow{\tau_3} \gamma.\mathsf{users}$.

Otherwise, extract $\mathtt{TX}^{P}_{\mathsf{CM},i+1}$ and $\overline{\mathtt{TX}_{\mathsf{SP},i+1}}$ from $\Gamma'^P(id)$:

1. If $\mathtt{TX}^{P}_{\mathsf{CM},i+1} =\perp$, Send $(\text{post}, \mathtt{TX}^{P}_{\mathsf{CM},i}) \xhookrightarrow{\tau_0} \mathcal{L}$. Else, Send $(\text{post}, \mathtt{TX}^{P}_{\mathsf{CM},i+1}) \xhookrightarrow{\tau_0} \mathcal{L}$.
2. Let $\tau_1 \leq \tau_0 + \Delta$ be the round in which either $\mathtt{TX}^{P}_{\mathsf{CM},i}$ or $\mathtt{TX}^{P}_{\mathsf{CM},i+1}$ is accepted by the blockchain. Wait for T rounds, set $[\mathtt{TX}_{\mathsf{SP},i+1}] = (\mathtt{TX}.\mathsf{txid}\|1, \overline{\mathtt{TX}_{\mathsf{SP},i+1}}.\mathsf{nLT}, \overline{\mathtt{TX}_{\mathsf{SP},i+1}}.\mathsf{Output})$, $\mathtt{TX}_{\mathsf{SP},i+1} := (\mathcal{H}([\mathtt{TX}_{\mathsf{SP},i+1}]), \mathtt{TX}.\mathsf{txid}\|1, \overline{\mathtt{TX}_{\mathsf{SP},i+1}})$ and then post $(\text{post}, \mathtt{TX}_{\mathsf{SP},i+1}) \xhookrightarrow{\tau_2 := \tau_1 + T} \mathcal{L}$.
3. Once $\mathtt{TX}_{\mathsf{SP},i+1}$ is accepted by $\mathcal{L}$ in round $\tau_3 \leq \tau_2 + \Delta$ set $\Theta^P(id) :=\perp$ and $\Gamma^P(id) :=\perp$ and output $(\mathtt{CLOSED}, id) \xhookrightarrow{\tau_3} \gamma.\mathsf{users}$.

Lemma 1. *Let Σ be a secure signature scheme. Then, the Create phase of protocol π GUC-emulates the Create phase of functionality $\mathcal{F}$.*

Proof. We define the following messages.

- $m_0 := (\text{createCom}, id, \tilde{\sigma}^B_{\mathtt{TX}_{\mathsf{SP},0}}, \sigma^B_{\mathtt{TX}^A_{\mathsf{CM},0}})$,
- $m_1 := (\text{createFund}, id, \sigma^A_{\mathtt{TX}_{\mathsf{FU}}})$,

The proof is composed of multiple hybrids, where we gradually modify the initial experiment.

Hybrid $\mathcal{H}_0$: This corresponds to the Create phase of protocol π.

Hybrid $\mathcal{H}_1$: For the honest party A in hybrid $\mathcal{H}_0$, if the corrupted party B publishes the funding transaction $\mathtt{TX}_{\mathtt{FU}}$ on the ledger $\mathcal{L}$ without sending the message m_0 to A in round $\tau_0 + 1$, then the experiment outputs `Error` and fails.

Simulator $\mathcal{S}$: This corresponds to the Create phase of the simulator, as defined in beginning of this Appendix.

Lemma 2. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_0,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \approx$$
$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_1,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$$

Proof. Two hybrids differ if the experiment outputs `Error`. Thus, we must bound the probability that this event occurs. The experiment outputs `Error`, if and only if the corrupted party B publishes the funding transaction $\mathtt{TX}_{\mathtt{FU}}$ on the ledger $\mathcal{L}$ without sending the message m_0 to A. Furthermore, according to the protocol π, if A does not receive the message m_0, he does not send the message m_1 including the signature $\sigma^{A}_{\mathtt{TX}_{\mathtt{FU}}}$ to B. However, this signature must be part of $\mathtt{TX}_{\mathtt{FU}}.\mathsf{Witness}$ if $\mathtt{TX}_{\mathtt{FU}}$ is published on $\mathcal{L}$ and hence the signature $\sigma^{A}_{\mathtt{TX}_{\mathtt{FU}}}$ must be created by the adversary. Thus, given that the experiment outputs `Error` with non-negligible probability, as will be shown in the next paragraph, we construct a reduction against the existential unforgeability of the underlying signature scheme Σ with non-negligible success probability which contradicts with our assumption regarding the security of Σ.

Assume that $\Pr[\mathtt{Error} \mid \mathcal{H}_0] \geq \frac{1}{\mathsf{poly}(\lambda)}$. The reduction receives as input a public key pk from the challenger and registers it by sending the message $(\text{register}, pk)$ to $\mathcal{L}$. Now assume that the honest party A, upon receiving the message $(\mathtt{INTRO}, \gamma, tid_A)$ from $\mathcal{E}$, initiates the Create phase of protocol π. If in this process, m_0 is received from the adversary, the hybrid $\mathcal{H}_1$ does not output `Error` and the reduction aborts. If the experiment outputs `Error`, meaning that for this channel the corresponding funding transaction $\mathtt{TX}_{\mathtt{FU}}$ is accepted by $\mathcal{L}$, then the reduction outputs (m^*, σ^*) with $m^* = [\mathtt{TX}_{\mathtt{FU}}]$ and $\sigma^* \in \mathtt{TX}_{\mathtt{FU}}.\mathsf{Witness}$. This reduction is clearly efficient, and whenever $\mathcal{H}_1$ outputs `Error`, the reduction succeeds in forging the signature. Moreover, the reduction has never called the signing oracle for any messages. Therefore, the reduction outputs a valid forgery with probability at least $\frac{1}{\mathsf{poly}(\lambda)}$, which contradicts with our assumption regarding the security of the signature scheme Σ. This proves that $\Pr[\mathtt{Error} \mid \mathcal{H}_0] < \frac{1}{\mathsf{poly}(\lambda)}$.

Lemma 3. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_1,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \approx$$
$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\varphi_{\mathcal{F}},\mathcal{S},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}.$$

Proof. The two experiments are identical, and hence, indistinguishability follows.

This concludes the proof of Lemma 1.

Lemma 4. *The Update phase of protocol π GUC-emulates the Update phase of functionality $\mathcal{F}$.*

Proof. The two experiments are identical, and hence, indistinguishability follows.

Lemma 5. *The Close phase of protocol π GUC-emulates the Close phase of functionality $\mathcal{F}$.*

Proof. The proof is composed of multiple hybrids, where we gradually modify the initial experiment.

Hybrid $\mathcal{H}_0$: This corresponds to the Close phase of protocol π.

Hybrid $\mathcal{H}_1$: If the honest party A initiates hybrid $\mathcal{H}_0$ in round τ_0 and in round $\tau_0 + 1 + \Delta$, the output $\texttt{TX}_{\textsf{FU}}.\textsf{Output}$ is still unspent, then the experiment outputs `Error` and fails.

Simulator $\mathcal{S}$: This corresponds to the Close phase of the simulator, as defined in beginning of this Appendix.

Lemma 6. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\{\text{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_0,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \approx$$
$$\{\text{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_1,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}.$$

Proof. Similar to the proof of Lemma 2, we must bound the probability that $\mathcal{H}_1$ outputs `Error`. According to $\mathcal{H}_1$, the honest party A sends either $(\text{post}, \texttt{TX}_{\overline{\texttt{SP}}})$ $\xhookrightarrow{\tau_0+1} \mathcal{L}$ or $(\text{post}, \texttt{TX}^{A}_{\texttt{CM},i}) \xhookrightarrow{\tau_0+1} \mathcal{L}$. Since both $\texttt{TX}_{\overline{\texttt{SP}}}$ and $\texttt{TX}^{A}_{\texttt{CM},i}$ are valid and both take output of $\texttt{TX}_{\texttt{FU}}$ as their input, based on the ledger functionality $\mathcal{L}$, $\texttt{TX}_{\texttt{FU}}.\textsf{Output}$ becomes spent within at most Δ rounds. Therefore, $\mathcal{H}_1$ will never output `Error`. This completes the proof.

Lemma 7. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\{\text{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_1,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \approx$$
$$\{\text{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\varphi_{\mathcal{F}},\mathcal{S},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}.$$

Proof. The two experiments are identical, and hence, indistinguishability follows.

This concludes the proof of Lemma 5.

Lemma 8. *Let Σ be a secure signature scheme. Then, the Punish phase of protocol π GUC-emulates the Punish phase of functionality $\mathcal{F}$.*

Proof. The proof is composed of multiple hybrids, where we gradually modify the initial experiment.

- Hybrid $\mathcal{H}_0$: This corresponds to the Punish phase of the protocol π.
- Hybrid $\mathcal{H}_1$: For the honest party A in hybrid $\mathcal{H}_0$, if a transaction `TX` is published s.t. $[\texttt{TX}] = [\texttt{TX}^{B}_{\texttt{CM},j}]$ with $j \in [0, i-1]$ but $\texttt{TX}^{A}_{\texttt{RV},i-1}$ is not accepted by $\mathcal{L}$ within Δ rounds, then the experiment outputs `Error` and fails.
- Hybrid $\mathcal{H}_2$: For the honest party A in hybrid $\mathcal{H}_1$, if a transaction `TX` is published s.t. $[\texttt{TX}] = [\texttt{TX}^{B}_{\texttt{CM},i}]$ given that $\gamma.\textsf{flag} = 1$ or $[\texttt{TX}] \in \{[\texttt{TX}^{B}_{\texttt{CM},i}], [\texttt{TX}^{B}_{\texttt{CM},i+1}]\}$ given that $\gamma.\textsf{flag} = 2$, and then a transaction $\texttt{TX}'$ is published s.t. $\texttt{TX}'.\textsf{Input} = \texttt{TX}.\textsf{txid}\|1$ and $\texttt{TX}'.\textsf{Witness}.\eta = 2$ (i.e. $\texttt{TX}'.\textsf{Witness}$ satisfies the second subcondition of $\texttt{TX}.\textsf{Output}$), then the experiment outputs `Error` and fails.
- Hybrid $\mathcal{H}_3$: For the honest party A in hybrid $\mathcal{H}_2$, if either of the following cases occur, the experiment outputs `Error` and fails.

  - While $\gamma.\mathsf{flag} = 1$, a transaction $\mathtt{TX}$ is published s.t. $[\mathtt{TX}] = [\mathtt{TX}^{A}_{\mathtt{CM},i}]$, and then a transaction $\mathtt{TX}'$ is published s.t. $\mathtt{TX}'.\mathsf{Input} = \mathtt{TX}.\mathsf{txid}\|1$ and $\mathtt{TX}'.\mathsf{Witness}.\eta = 2$ (i.e. $\mathtt{TX}'.\mathsf{Witness}$ satisfies the second subcondition of $\mathtt{TX}.\mathsf{Output}$).
  - While $\gamma.\mathsf{flag} = 2$, a transaction $\mathtt{TX}$ is published s.t. $[\mathtt{TX}] = [\mathtt{TX}^{A}_{\mathtt{CM},i}]$ given that $\Gamma'^{A}(id)[1] =\perp$ or $[\mathtt{TX}] = [\mathtt{TX}^{A}_{\mathtt{CM},i+1}]$ otherwise, and then a transaction $\mathtt{TX}'$ is published s.t. $\mathtt{TX}'.\mathsf{Input} = \mathtt{TX}.\mathsf{txid}\|1$ and $\mathtt{TX}'.\mathsf{Witness}.\eta = 2$ (i.e. $\mathtt{TX}'.\mathsf{Witness}$ satisfies the second subcondition of $\mathtt{TX}.\mathsf{Output}$).
- Hybrid $\mathcal{H}_4$: For the honest party A in hybrid $\mathcal{H}_3$, if either of the following cases occur, the experiment outputs `Error` and fails.
  - While $\gamma.\mathsf{flag} = 1$, a transaction $\mathtt{TX}$ with $[\mathtt{TX}] \in \{[\mathtt{TX}^{A}_{\mathtt{CM},i}], [\mathtt{TX}^{B}_{\mathtt{CM},i}]\}$ is published, but $\mathtt{TX}_{\mathtt{SP},i}$ is not accepted by $\mathcal{L}$ within $T + \Delta$ rounds.
  - While $\gamma.\mathsf{flag} = 2$, a transaction $\mathtt{TX}$ with $[\mathtt{TX}] \in \{[\mathtt{TX}^{A}_{\mathtt{CM},i}], [\mathtt{TX}^{B}_{\mathtt{CM},i}], [\mathtt{TX}^{B}_{\mathtt{CM},i+1}]\}$ is published, but $\mathtt{TX}_{\mathtt{SP},i}$ or $\mathtt{TX}_{\mathtt{SP},i+1}$ is not accepted by $\mathcal{L}$ within $T + \Delta$ rounds.
- Hybrid $\mathcal{H}_5$: For the honest party A in hybrid $\mathcal{H}_4$, if either of the following cases occur, the experiment outputs `Error` and fails.
  - While $\gamma.\mathsf{flag} = 1$, a transaction $\mathtt{TX}$ with $\mathtt{TX}.\mathsf{Input} = \mathtt{TX}_{\mathtt{FU}}.\mathsf{txid}\|1$ is published s.t. $[\mathtt{TX}] \notin \{[\mathtt{TX}^{A}_{\mathtt{CM},i}], [\mathtt{TX}^{B}_{\mathtt{CM},j}]\}$, $j = [0, i]$ and $\mathtt{TX}.\mathsf{Output} \neq \gamma.\mathsf{st}$.
  - While $\gamma.\mathsf{flag} = 2$, a transaction $\mathtt{TX}$ with $\mathtt{TX}.\mathsf{Input} = \mathtt{TX}_{\mathtt{FU}}.\mathsf{txid}\|1$ is published s.t. $[\mathtt{TX}] \notin \{[\mathtt{TX}^{A}_{\mathtt{CM},i}], [\mathtt{TX}^{A}_{\mathtt{CM},i+1}], [\mathtt{TX}^{B}_{\mathtt{CM},j}]\}$, $j = [0, i{+}1]$ and $\mathtt{TX}.\mathsf{Output} \neq \gamma.\mathsf{st}$.
- Simulator $\mathcal{S}$: This corresponds to the Punish phase of the simulator, as defined in beginning of this Appendix.

Lemma 9. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_0,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \approx$$
$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_1,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}.$$

Proof. Two hybrids differ if the experiment outputs `Error`. Thus, we must bound the probability that this event occurs. The hybrid $\mathcal{H}_1$ does not output `Error` unless $\mathtt{TX}_{\mathtt{FU}}.\mathsf{Output}$ is spent by a transaction $\mathtt{TX}$ s.t. $[\mathtt{TX}] = [\mathtt{TX}^{B}_{\mathtt{CM},j}]$ with $j \in [0, i-1]$. According to $\mathcal{H}_1$, once this transaction is observed by A at the end of round τ_0, A posts $(\text{post}, \mathtt{TX}^{A}_{\mathtt{RV},i-1}) \overset{\tau_0}{\hookrightarrow} \mathcal{L}$. Since $\mathtt{TX}^{A}_{\mathtt{RV},i-1}$ with $\mathtt{TX}^{A}_{\mathtt{RV},i-1}.\mathsf{Input} = \mathtt{TX}.\mathsf{txid}\|1$ is a valid transaction, it is accepted by $\mathcal{L}$ within Δ rounds unless another valid transaction $\mathtt{TX}'$ is published within this Δ-round interval with $\mathtt{TX}'.\mathsf{Input} = \mathtt{TX}.\mathsf{txid}\|1$. The output $\mathtt{TX}.\mathsf{Output}$ has two subconditions, one of which must be satisfied by $\mathtt{TX}'$. The first subcondition $\varphi_1 = pk^{A}_{\mathtt{SP}} \wedge pk^{B}_{\mathtt{SP}} \wedge CLTV_{S_0+j} \wedge CSV_T$ cannot be satisfied within T rounds and since we have $T > \Delta$, φ_1 cannot be met within Δ rounds. Satisfying the second subcondition $\varphi_2 = pk'^{A}_{\mathtt{RV}} \wedge pk'^{B}_{\mathtt{RV}} \wedge CLTV_{S_0+j}$ requires A's signature and according to the protocol π, A does not grant such an authorization to anyone. Thus, if the experiment outputs `Error` with non-negligible probability, we construct a reduction against the existential unforgeability of the underlying signature scheme Σ with non-negligible success probability which contradicts with our assumption regarding the security of Σ.

Assume that $\Pr[\texttt{Error} \mid \mathcal{H}_0] \geq \frac{1}{\mathsf{poly}(\lambda)}$. The reduction receives a public key pk from the challenger as input, and in the channel creation phase sets $pk'^{A}_{\texttt{RV}} := pk$. The channel might be updated any arbitrary number of times and might be closed at any time using any method (peacefully or forcefully) selected by the adversary. Assume that the channel has been updated i times. If the output of the funding transaction is spent by a transaction $\texttt{TX}$ s.t. $[\texttt{TX}] \neq [\texttt{TX}^{B}_{\texttt{CM},j}]$ with $j = [0, i-1]$, the hybrid $\mathcal{H}_1$ does not output $\texttt{Error}$ and hence the reduction aborts. If a transaction $\texttt{TX}$ with $[\texttt{TX}] \in [\texttt{TX}^{B}_{\texttt{CM},j}]$ with $j = [0, i-1]$ is published, the reduction calls the signing oracle for the message $[\texttt{TX}^{A}_{\texttt{RV},i-1}]$, creates $\texttt{TX}^{A}_{\texttt{RV},i-1}$ and posts it to the ledger $\mathcal{L}$. If $\texttt{TX}^{A}_{\texttt{RV},i-1}$ is published on $\mathcal{L}$ within Δ rounds, the hybrid $\mathcal{H}_1$ does not output $\texttt{Error}$ and hence the reduction aborts. Otherwise, since $\texttt{TX}^{A}_{\texttt{RV},i-1}$ is a valid transaction, based on our assumptions on $\mathcal{L}$, another transaction $\texttt{TX}'$ with $[\texttt{TX}'] \neq [\texttt{TX}^{A}_{\texttt{RV},i-1}]$ and $\texttt{TX}'.\mathsf{Input} = \texttt{TX}.\mathsf{txid}\|1$ appears on the ledger within this Δ round interval. This causes $\mathcal{H}_1$ to output $\texttt{Error}$. As mentioned earlier, $\texttt{TX}'.\mathsf{Witness}$ satisfies the condition $pk'^{A}_{\texttt{RV}} \wedge pk'^{B}_{\texttt{RV}} \wedge CLTV_{S_0+j}$ of the output $\texttt{TX}.\mathsf{Output}$. Now, the reduction outputs (m^*, σ^*) with $m^* = [\texttt{TX}']$ and $\sigma^* \in \texttt{TX}'.\mathsf{Witness}.\zeta$. Moreover, the reduction has never called the signing oracle for m^* before, because the signing oracle was called only once for $[\texttt{TX}^{A}_{\texttt{RV},i-1}] \neq m^*$. Therefore, the reduction outputs a valid forgery with probability at least $\frac{1}{\mathsf{poly}(\lambda)}$, which contradicts with our assumption regarding the security of Σ. This contradiction proves that $\Pr[\texttt{Error} \mid \mathcal{H}_0] < \frac{1}{\mathsf{poly}(\lambda)}$.

Lemma 10. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_1,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \approx$$
$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_2,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}.$$

Proof. Similar to the proof of Lemma 9, we show that if the experiment outputs $\texttt{Error}$ with non-negligible probability we construct a reduction against the existential unforgeability of the underlying signature scheme Σ with non-negligible success probability. Assume that $\Pr[\texttt{Error} \mid \mathcal{H}_1] \geq \frac{1}{\mathsf{poly}(\lambda)}$. The reduction receives as input a public key pk, and in the channel creation phase sets $pk'^{A}_{\texttt{RV}} := pk$. The channel is updated any arbitrary number of times and might be closed at any time using any method (peacefully or forcefully) selected by the adversary. Assume that the channel has been updates i times. If the output of the funding transaction is spent by a transaction $\texttt{TX}$ with $[\texttt{TX}] \neq [\texttt{TX}^{B}_{\texttt{CM},i}]$ given that $\gamma.\mathsf{flag} = 1$ or $[\texttt{TX}] \notin \{[\texttt{TX}^{B}_{\texttt{CM},i}], [\texttt{TX}^{B}_{\texttt{CM},i+1}]\}$ given that $\gamma.\mathsf{flag} = 2$, the experiment does not output $\texttt{Error}$ and the reduction aborts. Otherwise, the reduction waits for T rounds and then publishes the latest split transaction. If $\texttt{TX}.\mathsf{Output}$ is spent by a transaction $\texttt{TX}'$ s.t. $\texttt{TX}'.\mathsf{Witness}.\eta = 1$, the experiment does not output $\texttt{Error}$ and the reduction aborts. However, if $\texttt{TX}'.\mathsf{Witness}.\eta = 1$, the experiment outputs $\texttt{Error}$. Since $\texttt{TX}'.\mathsf{Witness}$ satisfies the condition $pk'^{A}_{\texttt{RV}} \wedge pk'^{B}_{\texttt{RV}} \wedge CLTV_{S_0+i}$ of the output $\texttt{TX}.\mathsf{Output}$, reduction outputs (m^*, σ^*) with $m^* = [\texttt{TX}']$ and $\sigma^* \in \texttt{TX}'.\mathsf{Witness}.\zeta$. Moreover, the reduction has never called the signing oracle. Therefore, the reduction outputs a valid forgery with probability at least $\frac{1}{\cdot\mathsf{poly}(\lambda)}$, which contradicts with our assumption regarding the security of Σ. This contradiction proves that $\Pr[\texttt{Error} \mid \mathcal{H}_1] < \frac{1}{\mathsf{poly}(\lambda)}$.

Lemma 11. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_2,\mathcal{A},\mathcal{E}}(\lambda,z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \approx$$
$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_3,\mathcal{A},\mathcal{E}}(\lambda,z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}.$$

Proof. We bound the probability that the experiment outputs $\texttt{Error}$. Assume that $\Pr[\texttt{Error} \mid \mathcal{H}_2] \geq \frac{1}{\mathsf{poly}(\lambda)}$. The reduction receives as input a public key pk from the challenger and in the channel creation phase sets $pk'^{A}_{\texttt{RV}} := pk$. The channel is updated any arbitrary number of times and might be closed at any time using any method (peacefully or forcefully) selected by the adversary. For the j^{th} channel update, to generate the signature on $\texttt{TX}^{B}_{\texttt{RV},j-1}$, a call to the signing oracle for the message $\overline{[\texttt{TX}^{B}_{\texttt{RV},j-1}]}$ is performed. We know that the channel has been updated i times. Assume that $\texttt{TX}_{\texttt{FU}}.\mathsf{Output}$ is spent by a transaction $\texttt{TX}$. If one of the following cases occur, the experiment does not output $\texttt{Error}$ and the reduction aborts: 1) $\gamma.\mathsf{flag} = 1$ and $[\texttt{TX}] \neq [\texttt{TX}^{A}_{\texttt{CM},i}]$, 2) $\gamma.\mathsf{flag} = 2$, and $[\texttt{TX}] \neq [\texttt{TX}^{A}_{\texttt{CM},i}]$ given that $\Gamma'^{A}(id)[1] =\perp$ or $[\texttt{TX}] \neq [\texttt{TX}^{A}_{\texttt{CM},i+1}]$ otherwise. Otherwise, the reduction waits for T rounds and then publishes the latest split transaction. If $\texttt{TX}.\mathsf{Output}$ is spent by a transaction $\texttt{TX}'$ s.t. $\texttt{TX}'.\mathsf{Witness}.\eta = 1$, the experiment does not output $\texttt{Error}$ and hence the reduction aborts. If $\texttt{TX}'.\mathsf{Witness}.\eta = 2$, the experiment outputs $\texttt{Error}$. Now either of the following cases might have occurred:

- We have $\gamma.\mathsf{flag} = 1$ or $\gamma.\mathsf{flag} = 2$ and $\Gamma'^{A}(id)[1] =\perp$ and hence $[\texttt{TX}] = [\texttt{TX}^{A}_{\texttt{CM},i}]$ holds. Also, since $\texttt{TX}'.\mathsf{Witness}.\eta = 2$, $\texttt{TX}'.\mathsf{Witness}.\zeta$ satisfies the condition $pk'^{A}_{\texttt{RV}} \wedge pk'^{B}_{\texttt{RV}} \wedge CLTV_{S_0+i}$. The reduction outputs (m^*, σ^*) with $m^* = [\texttt{TX}']$ and $\sigma^* \in \texttt{TX}'.\mathsf{Witness}.\zeta$. Moreover, the signing oracle was only called for messages $\overline{[\texttt{TX}^{B}_{\texttt{RV},j}]}$ with $j < i$ and since for these transactions we have $\overline{[\texttt{TX}^{B}_{\texttt{RV},j}]}.\mathsf{nLT} = S_0 + j < S_0 + i$, according to $\mathcal{L}$, $\texttt{TX}^{B}_{\texttt{RV},j}$ can not spend $\texttt{TX}.\mathsf{Output}$ and hence $m^* \neq \overline{[\texttt{TX}^{B}_{\texttt{RV},j}]}$ with $j < i$.
- We have $\gamma.\mathsf{flag} = 2$ and $\Gamma'^{A}(id)[1] \neq\perp$ and hence $[\texttt{TX}] = [\texttt{TX}^{A}_{\texttt{CM},i+1}]$ holds. Also, since $\texttt{TX}'.\mathsf{Witness}.\eta = 2$, $\texttt{TX}'.\mathsf{Witness}.\zeta$ satisfies the condition $pk'^{A}_{\texttt{RV}} \wedge pk'^{B}_{\texttt{RV}} \wedge CLTV_{S_0+i+1}$. The reduction outputs (m^*, σ^*) with $m^* = [\texttt{TX}']$ and $\sigma^* \in \texttt{TX}'.\mathsf{Witness}.\zeta$. Moreover, the signing oracle was only called for messages $\overline{[\texttt{TX}^{B}_{\texttt{RV},j}]}$ with $j \leq i$ and since for these transactions we have $\overline{[\texttt{TX}^{B}_{\texttt{RV},j}]}.\mathsf{nLT} = S_0 + j < S_0 + i + 1$, according to $\mathcal{L}$, $\texttt{TX}^{B}_{\texttt{RV},j}$ cannot spend $\texttt{TX}.\mathsf{Output}$ and hence $m^* \neq \overline{[\texttt{TX}^{B}_{\texttt{RV},j}]}$ with $j \leq i$.

Therefore, the reduction has never called the signing oracle for the message m^* and hence the reduction outputs a valid forgery with probability at least $\frac{1}{\mathsf{poly}(\lambda)}$, which contradicts with our assumption regarding the security of Σ. This contradiction proves that $\Pr[\texttt{Error} \mid \mathcal{H}_2] < \frac{1}{\mathsf{poly}(\lambda)}$.

Lemma 12. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_3,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \approx$$
$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_4,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}.$$

Proof. Similar to previous proofs, assume that $\Pr[\mathtt{Error} \mid \mathcal{H}_3] \geq \frac{1}{\mathsf{poly}(\lambda)}$. The reduction receives as input a public key pk, and in the channel creation phase, sets $pk^A_{\mathsf{SP}} = pk$. The channel is updated any arbitrary number of times and might be closed at any time using any method (peacefully or forcefully) selected by the adversary. Once the channel is created or once it is updated for the j^{th} time, a call to the signing oracle is performed to receive the signature on $\mathtt{TX}_{\mathsf{SP},0}$ and $\mathtt{TX}_{\mathsf{SP},j}$, respectively. Assume that the channel has been updated i times. If the output of the funding transaction is spent by a transaction $\mathtt{TX}$ with $[\mathtt{TX}] \in \{[\mathtt{TX}^A_{\mathtt{CM},i}], [\mathtt{TX}^B_{\mathtt{CM},i}]$ given that $\gamma.\mathsf{flag} = 1$ or $[\mathtt{TX}] \in \{[\mathtt{TX}^A_{\mathtt{CM},i}], [\mathtt{TX}^B_{\mathtt{CM},i}], [\mathtt{TX}^A_{\mathtt{CM},i+1}], [\mathtt{TX}^B_{\mathtt{CM},i+1}]$ given that $\gamma.\mathsf{flag} = 2$, the experiment does not output $\mathtt{Error}$ and the reduction aborts. Otherwise, the reduction waits for T rounds and then posts the transaction $\mathtt{TX}'$ on the ledger $\mathcal{L}$ with $\mathtt{TX}' = \mathtt{TX}_{\mathsf{SP},i}$ given that $\gamma.\mathsf{flag} = 1$ or $\mathtt{TX}' = \mathtt{TX}_{\mathsf{SP},i+1}$ given that $\gamma.\mathsf{flag} = 2$.

If $\mathtt{TX}'$ is accepted by $\mathcal{L}$, the experiment does not output $\mathtt{Error}$ and the reduction aborts. If $\mathtt{TX}.\mathsf{Output}$ is spent by a transaction $\mathtt{TX}''$ with $[\mathtt{TX}''] \neq [\mathtt{TX}']$ we know that either the first or the second subcondition of $\mathtt{TX}.\mathsf{Output}$ is satisfied by $\mathtt{TX}''.\mathsf{Witness}$. The second subcondition of $\mathtt{TX}.\mathsf{Output}$ is not satisfied by $\mathtt{TX}''.\mathsf{Witness}$ otherwise $\mathcal{H}_2$ or $\mathcal{H}_3$ would output $\mathtt{Error}$ which contradicts with our assumptions. Thus, $\mathtt{TX}''.\mathsf{Witness}$ satisfies the first subcondition of $\mathtt{TX}.\mathsf{Output}$. Therefore, the reduction outputs (m^*, σ^*) with $m^* = \overline{[\mathtt{TX}'']}$ and $\sigma^* \in \mathtt{TX}''.\mathsf{Witness}.\zeta$. Moreover, the signing oracle was only called for messages $\overline{[\mathtt{TX}_{\mathsf{SP},j}]}$ with $j = [0, i]$ given that $\gamma.\mathsf{flag} = 1$ or $j = [0, i+1]$ given that $\gamma.\mathsf{flag} = 2$. However, $m^* \notin \{\overline{[\mathtt{TX}_{\mathsf{SP},i}]}, \overline{[\mathtt{TX}_{\mathsf{SP},i+1}]}\}$. Otherwise, $\mathcal{H}_4$ would not output $\mathtt{Error}$. Also, $m^* \notin \{\overline{[\mathtt{TX}_{\mathsf{SP},j}]}$ with $j = [0, i-1]$ because for these transactions we have $\overline{[\mathtt{TX}_{\mathsf{SP},j}]}.\mathsf{nLT} = S_0 + j < S_0 + i$, and hence according to $\mathcal{L}$, $\mathtt{TX}_{\mathsf{SP},j}$ can not spend $\mathtt{TX}.\mathsf{Output}$. Therefore, the reduction has never called the signing oracle for the message m^* and hence the reduction outputs a valid forgery with probability at least $\frac{1}{\mathsf{poly}(\lambda)}$, which contradicts with our assumption regarding the security of Σ. This contradiction proves that $\Pr[\mathtt{Error} \mid \mathcal{H}_3] < \frac{1}{\mathsf{poly}(\lambda)}$.

Lemma 13. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_4,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \approx$$
$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_5,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}.$$

Proof. Similar to previous proofs, we construct a reduction against the existential unforgeability of the underlying signature scheme Σ. Assume that $\Pr[\mathtt{Error} \mid \mathcal{H}_4] \geq \frac{1}{\mathsf{poly}(\lambda)}$. The reduction receives as input a public key pk from the challenger and in the channel creation phase sets $pk_A := pk$. The channel is updated any arbitrary number of times and might be closed at any time using any method (peacefully or forcefully) selected by the adversary. All calls to the signing algorithm are redirected to the signing oracle. Assume that the channel has been updated i times. Now assume that the output of the funding transaction is spent by a transaction $\mathtt{TX}$ s.t. either of the following two sets of conditions hold:

- $\gamma.\mathsf{flag} = 1$, $[\mathtt{TX}] \notin \{[\mathtt{TX}^A_{\mathtt{CM},i}], [\mathtt{TX}^B_{\mathtt{CM},j}]\}$, $j = [0, i]$ and $\mathtt{TX}.\mathsf{Output} \neq \gamma.\mathsf{st}$.
- $\gamma.\mathsf{flag} = 2$, $[\mathtt{TX}] \notin \{[\mathtt{TX}^A_{\mathtt{CM},i}], [\mathtt{TX}^A_{\mathtt{CM},i+1}], [\mathtt{TX}^B_{\mathtt{CM},j}]\}$, $j = [0, i+1]$ and $\mathtt{TX}.\mathsf{Output} \neq \gamma.\mathsf{st}$.

Then, the hybrid outputs $\mathtt{Error}$. The reduction also outputs (m^*, σ^*) with $m^* = [\mathtt{TX}]$ and $\sigma^* \in \mathtt{TX}.\mathsf{Witness}.\zeta$. The signature σ^* is a valid signature on m^* because $\mathtt{TX}.\mathsf{Witness}$ satisfies the condition of $\mathtt{TX}_{\mathtt{FU}}.\mathsf{Output}$ which is $pk_A \wedge pk_B$. Moreover, as we will show in the next paragraph, the signing oracle was never called for the message m^*.

Once the channel is created or each time it is updated, a call to the signing oracle is performed to receive the signature on the funding transaction as well as the new commit transaction held by B, i.e. $[\mathtt{TX}^B_{\mathtt{CM},j}]$ with $j = [0, i]$ if $\gamma.\mathsf{flag} = 1$ or $j = [0, i+1]$ otherwise. Also, if the channel is closed forcefully, A calls the signing oracle to receive the signature on $[\mathtt{TX}^A_{\mathtt{CM},i}]$ if $\gamma.\mathsf{flag} = 1$ or either $[\mathtt{TX}^A_{\mathtt{CM},i}]$ or $[\mathtt{TX}^A_{\mathtt{CM},i+1}]$ otherwise. If the channel is closed peacefully, A calls the signing oracle to receive the signature on $[\mathtt{TX}]$ with $\mathtt{TX}.\mathsf{Output} = \gamma.\mathsf{st}$. Therefore, the reduction has never called the oracle for the message m^*.

Lemma 14. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\mathcal{H}_5,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \approx$$
$$\{\mathrm{EXE}^{\mathcal{L}(\Delta,\Sigma),\mathcal{F}_{clock}}_{\varphi_{\mathcal{F}},\mathcal{S},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}.$$

Proof. The two experiments are identical, and hence, indistinguishability follows.

This concludes the proof of Lemma.

*

Proof. This theorem follows directly from Lemma 1, Lemma 4, Lemma 5 and Lemma 8.

H Performance analysis

H.1 Lightning channel

1) Non-collaborative Closure: To compute the total number of published bytes in the non-collaborative closure scenario, we assume that the latest commit transaction includes $\frac{m}{2}$ offered HTLC outputs and $\frac{m}{2}$ received HTLC outputs (in addition to the to_local and to_remote outputs). We also assume that once the latest commit transaction is published, half of the offered HTLC outputs are redeemed by payee (through publishing some transactions that we call Redeem transactions) and the rest are claimed back by the payer through publishing the HTLC-timeout transactions. Also, half of the received HTLC outputs are claimed by the payee through publishing the HTLC-Success transactions and the rest are claimed back by the payer (through publishing some transactions that we call Claimback transactions). We compute the size of each transaction, hereinafter.

Commit transaction

The commit transaction with m HTLC outputs contains 224 bytes of witness data and $125 + 43m$ bytes of non-witness data [1].

HTLC-Timeout transaction

Each HTLC-Timeout transaction contains 287 bytes of witness data and 94 bytes of non-witness data [1].

HTLC-Success transaction

Each HTLC-Success transaction contains 326 bytes of witness data and 94 bytes of non-witness data [1].

Redeem transaction

The Redeem transaction contains (244 bytes of witness data and 82 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim an offered HTLC output: 242 bytes.
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input including 41 bytes
  - number of outputs: 1 byte
  - one P2WPKH output: 31 bytes
  - locktime: 4 bytes

The witness to claim an offered HTLC output includes (242 bytes):

- number of witness elements: 1 byte
- preimage length: 1 byte
- preimage: 32 byte
- signature length: 1 byte
- signature: 73 bytes
- witness_script_length: 1 byte
- witness_script: 133 bytes

The witness script contains 133 bytes [1].

Claimback transaction

The Claimback transaction contains (219 bytes of witness data and 82 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim a received HTLC output: 217 bytes.

- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input including 41 bytes
  - number of outputs: 1 byte
  - one P2WPKH output: 31 bytes
  - locktime: 4 bytes

The witness to claim a received HTLC output includes (217 bytes):

- number of witness elements: 1 byte
- 0: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- witness_script_length: 1 byte
- witness_script: 140 bytes

The witness script contains 140 bytes [1].

So, one commit transaction, $\frac{m}{4}$ HTLC-Timeout, $\frac{m}{4}$ HTLC-Success, $\frac{m}{4}$ Redeem and $\frac{m}{4}$ Claimback transactions contain 224+269m bytes of witness data and $125 + 131m$ bytes of non-witness data, in total.

2) Dishonest Closure: If a revoked commit transaction is published, we assume that the victim claims all revoked outputs, i.e. $\frac{m}{2}$ offered HTLC outputs, $\frac{m}{2}$ received HTLC outputs, and one *to_local* output, through a revocation transaction. The revocation transaction contains (157+246.5m byte of witness data and 82+41m bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim to_local output: 155 bytes.
  - witnesses to claim $\frac{m}{2}$ offered HTLC outputs, each including 243 bytes: in total 121.5m
  - witnesses to claim $\frac{m}{2}$ received HTLC outputs, each including 250 bytes: in total 125m bytes.
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - $m + 1$ inputs, each including 41 bytes: in total $41 + 41m$ bytes
  - number of outputs: 1 byte
  - one P2WPKH output: 31 bytes
  - locktime: 4 bytes

The witness to claim to_local output includes (155 bytes):

- number of witness elements: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- 1: 1 byte

- witness_script_length: 1 byte
- witness_script: 78 bytes

The witness script is as follows (78 bytes):

```
OP_IF (1 byte)
   #Penalty transaction
   ⟨revocationpubkey⟩ (1 byte for OP_DATA and 33 bytes for the public key)
OP_ELSE (1 byte)
   'to_self_delay'(4 bytes)
   OP_CHECKSEQUENCEVERIFY (1 byte)
   OP_DROP (1 byte)
   ⟨local_delayedpubkey⟩ (1 byte for OP_DATA and 33 bytes for the public
key)
OP_ENDIF (1 byte)
OP_CHECKSIG (1 byte)
```

The witness to claim an offered output includes (243 bytes):

- number of witness elements: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- public key length: 1 byte
- public key: 33 bytes
- witness_script_length: 1 byte
- witness_script: 133 bytes

The witness to claim a received output includes (250 bytes):

- number of witness elements: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- public key length: 1 byte
- public key: 33 bytes
- witness_script_length: 1 byte
- witness_script: 140 bytes

The witness script for accepted output and offered output can be found in [1].

So, one commit transaction and a revocation transaction contain $381{+}246.5m$ bytes of witness data and $207 + 84m$ bytes of non-witness data.

3) Number of Operations: For each channel update, channel parties must create the following signatures:

- signature on the commit transaction for their counter-party
- $\frac{m}{4}$ signatures on HTLC-Timeout transactions for their counter-party
- $\frac{m}{4}$ signatures on HTLC-Success transactions for their counter-party

Each party must verify the above signatures from their counter-party. Each party must also create the following signatures to give to the watchtower:

- one signature for the revocation transaction spending each output of the commit transaction (except to_remote output): $m+1$ signatures in total
- one signature for the revocation transaction spending the output of each HTLC-Timeout transaction: $\frac{m}{4}$ signatures in total
- one signature for the revocation transaction spending the output of each HTLC-Success transaction: $\frac{m}{4}$ signatures in total

Each party must also perform one exponentiation operation to generate a revocation key pair and one exponentiation operation to verify the revocation key pair of their counter-party.

H.2 Generalized channel

1) Non-collaborative Closure: In the non-collaborative closure scenario, the latest commit and split transactions are published on-chain. Then, we assume half of the HTLC outputs are redeemed by the payee (via Redeem' transactions) and the rest are claimed back by the payer (via Claimback' transaction). As we will see, specifications of Redeem' and Claimback' transactions are different from their corresponding transactions in a Lightning channel. We compute the size of each transaction, hereinafter.

Commit transaction

The commit transaction contains (224 byte of witness data and 94 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim a 2-of-2 multisignature output: 222 bytes.
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - one P2WSH output: 43 bytes
  - locktime: 4 bytes

The witness to claim a 2-of-2 multisignature output includes (222 bytes):

- number of witness elements: 1 byte
- 0: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- signature length: 1 byte
- signature: 73 bytes
- witness_script_length: 1 byte
- witness_script: 71 bytes

The witness script is as follows (71 bytes):
OP_2 (1 byte)
⟨pubkey1⟩ (1 byte for OP_DATA and 33 bytes for the public key)
⟨pubkey2⟩ (1 byte for OP_DATA and 33 bytes for the public key)
OP_2 (1 byte)
OP_CHECKMULTSIG (1 byte)

Split transaction

The split transaction (with two P2WPKH and m HTLC P2WSH outputs) contains (380 byte of witness data and 113+43m bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim commit transaction's output: 378 bytes.
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - two P2WPKH outputs: 62 bytes
  - m P2WSH outputs: 43m bytes
  - locktime: 4 bytes

The witness to claim commit transaction's output includes (378 bytes):

- number of witness elements: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- signature length: 1 byte
- signature: 73 bytes
- witness_script_length: 1 byte
- witness_script: 228 bytes

The witness script is as follows [4] (228 bytes):

```
⟨pubkeyA⟩ (1 byte for OP_DATA and 33 bytes for the public key)
OP_CHECKSIG (1 byte)
OP_SWAP (1 byte)
⟨pubkeyB⟩ (1 byte for OP_DATA and 33 bytes for the public key)
OP_CHECKSIG (1 byte)
OP_IF (1 byte)
  OP_IF (1 byte)
    delta (4 bytes)
    OP_CHECKSEQUENCEVERIFY (1 byte)
    OP_DROP (1 byte)
  OP_ELSE (1 byte)
    adaptor_pubkeyA (1 byte for OP_DATA and 33 bytes for the public key)
    OP_CHECKSIGVERIFY (1 byte)
    OP_HASH256 (1 byte)
    hashedsecret_rev_A (1 byte for OP_DATA and 32 bytes for the secret)
    OP_EQUALVERIFY (1 byte)
  OP_ENDIF (1 byte)
OP_ELSE (1 byte)
  OP_IF (1 byte)
    adaptor_pubkeyB (1 byte for OP_DATA and 33 bytes for the public key)
    OP_CHECKSIGVERIFY (1 byte)
    OP_HASH256 (1 byte)
    hashedsecret_rev_B (1 byte for OP_DATA and 32 bytes for the secret)
    OP_EQUALVERIFY (1 byte)
  OP_ELSE (1 byte)
    OP_RETURN (1 byte)
  OP_ENDIF (1 byte)
OP_ENDIF (1 byte)
1 (1 byte)
```

Redeem' transaction

The Redeem' transaction contains (212 bytes of witness data and 82 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness for HTLC output: 210 bytes.
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input including 41 bytes
  - number of outputs: 1 byte
  - one P2WPKH output: 31 bytes
  - locktime: 4 bytes

The witness to claim the HTLC output includes (210 bytes):

- number of witness elements: 1 byte
- preimage length: 1 byte
- preimage: 32 bytes
- signature length: 1 byte
- signature: 73 bytes
- witness_script_length: 1 byte
- witness_script: 101 bytes

The witness script contains (101 bytes) [32]:

```
  OP_SHA160 (1 byte)   ⟨digest⟩ (1 byte for OP_DATA and 20 bytes for the digest)
OP_EQUAL (1 byte)
OP_IF (1 byte)
  ⟨payee_pubkey⟩ (1 byte for OP_DATA and 33 bytes for public key)
OP_ELSE (1 byte)
```

```
    T (4 bytes)
    OP_CHECKSEQUENCEVERIFY (1 byte)
    OP_DROP (1 byte)
    ⟨payer_pubkey⟩ (1 byte for OP_DATA and 33 bytes for public key)
OP_ENDIF (1 byte)
OP_CHECKSIG (1 byte)
```

Claimback' transaction

The Claimback' transaction contains (180 bytes of witness data and 82 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness for HTLC output: 178 bytes.
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input including 41 bytes
  - number of outputs: 1 byte
  - one P2WPKH output: 31 bytes
  - locktime: 4 bytes

The witness to claim HTLC output includes (178 bytes):

- number of witness elements: 1 byte
- 0: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- witness_script_length: 1 byte
- witness_script: 101 bytes

So, one commit, one split, $\frac{m}{2}$ Redeem' and $\frac{m}{2}$ Claimback' transactions contain $624{+}195m$ bytes of witness data and $207{+}125m$ bytes of non-witness data, in total.

2) Dishonest Closure: If a revoked commit transaction is published, the victim must create a revocation transaction and sends all the channel funds to his own P2WPKH address.

The revocation transaction contains (414 bytes of witness data and 82 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness for commit transaction's output: 412 bytes
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - one P2WPKH outputs: 31 bytes

- locktime: 4 bytes

The witness to claim the commit transaction's output includes (412 bytes):

- number of witness elements: 1 byte
- revocation secret length: 1 byte
- revocation secret: 32 bytes
- signature length: 1 byte
- signature: 73 bytes
- 0: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- witness_script_length: 1 byte
- witness_script: 228 bytes

So, commit and revocation transactions contain 638 bytes of witness data and 176 bytes of non-witness data, in total.

3) Number of Operations: For each channel update, channel parties must create the following signatures:

- pre-signature on the commit transaction for their counter-party
- signature on the split transaction for their counter-party
- signature on the revocation transaction for the watchtower

Each party must verify the first two above signatures from their counterparty. Each party must also perform an exponentiation operation to generate a statement/witness pair to be used in the adaptor signature.

H.3 Daric channel

1) Non-collaborative Closure: In a non-collaborative closure, the latest commit and split transactions are published on-chain. Then, we assume half of the HTLC outputs are redeemed by the payee (via Redeem' transactions) and the rest are claimed back by the payer (via Claimback' transaction).

Commit transaction

The commit transaction contains 224 bytes of witness data and 94 bytes of non-witness data (For more details, see the figures for the Generalized channel).

Split transaction

The split transaction (with two P2WPKH and m HTLC P2WSH outputs) contains (311 bytes of witness data and 113+43m bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim the commit transaction's output: 309 bytes.
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes

- number of outputs: 1 byte
- two P2WPKH outputs: 62 bytes bytes
- m P2WSH outputs: $43m$ bytes
- locktime: 4 bytes

The witness to claim the commit transaction's output includes (309 bytes):

- number of witness elements: 1 byte
- 0: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- signature length: 1 byte
- signature: 73 bytes
- 0: 1 byte
- witness_script_length: 1 byte
- witness_script: 157 bytes

The witness script is as follows (157 bytes):

```
⟨absolute time S_0 + i ⟩ (4 byte)
OP_CHECKLOCKTIMEVERIFY (1 byte)
OP_DROP (1 byte)
OP_IF (1 byte)
   # Revocation
   2 (1 byte)
   ⟨Rev_pubkeyA⟩ (1 byte for OP_DATA and 33 bytes for the public key)
   ⟨Rev_pubkeyB⟩ (1 byte for OP_DATA and 33 bytes for the public key)
   2 (1 byte)
   OP_CHECKMULTISIG (1 byte)
OP_ELSE (1 byte)
   # Split
   ⟨delay T⟩ (4 bytes)
   OP_CHECKSEQUENCEVERIFY (1 bytes)
   OP_DROP (1 bytes)
   2 (1 bytes)
   ⟨Spl_pubkeyA⟩ (1 byte for OP_DATA and 33 bytes for the public key)
   ⟨Spl_pubkeyB⟩ (1 byte for OP_DATA and 33 bytes for the public key)
   2 (1 bytes)
   OP_CHECKMULTISIG (1 bytes)
OP_ENDIF (1 bytes)
```

Redeem' transaction

The Redeem' transaction contains 212 bytes of witness data and 82 bytes of non-witness data (For more details, see the figures for the Generalized channel).

Claimback' transaction

The Claimback' transaction contains 180 bytes of witness data and 82 bytes of non-witness data (For more details, see the figures for the Generalized channel).

So, one commit, one split, $\frac{m}{2}$ Redeem' and $\frac{m}{2}$ Claimback' transactions contain 535+196m bytes of witness data and 207+125m bytes of non-witness data, in total.

2) Dishonest Closure: If a revoked commit transaction is published, the victim must create a revocation transaction and sends all the channel funds to his own P2WPKH address. We computed the size of a commit transaction in the previous section.

The revocation transaction contains (311 bytes of witness data and 82 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim the commit transaction's output: 309 bytes.
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - one P2WPKH outputs: 31 bytes bytes
  - locktime: 4 bytes

The witness to claim the commit transaction's output includes (309 bytes):

- number of witness elements: 1 byte
- 0: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- signature length: 1 byte
- signature: 73 bytes
- 1: 1 byte
- witness_script_length: 1 byte
- witness_script: 157 bytes

So, commit and revocation transactions contain 535 byte of witness data and 176 bytes of non-witness data, in total.

3) Number of Operations: For each channel update, channel parties must create the following signatures:

- signature on the commit transaction for their counter-party
- signature on the split transaction for their counter-party
- signature on their counter-party's revocation transaction
- signature on their own revocation transaction for their watchtower

Each party must verify the first three above signatures

H.4 eltoo

An eltoo channel has a *trigger* transaction between the funding transaction and all update transactions. Since the update transactions are floating, if the public keys that are used in the output of the funding transaction are the same as the update public keys that are used in the output of the update transactions, it is possible to remove the trigger transaction. In this section, we consider such a simpler version.

1) Non-collaborative Closure: In a non-collaborative closure, the latest update and settlement transactions are published on-chain. Similar to other payment channels, we assume that the settlement transaction contains m HTLC outputs where half of the HTLC outputs are redeemed by the payee (via Redeem' transactions) and the rest are claimed back by the payer (via Claimback' transaction). We compute the size of each transaction, hereinafter.

update transaction

The update transaction contains (332 byte of witness data and 125 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim a 2-of-2 multisignature output: 222 bytes.
  - witness to claim a P2WPKH output: 108 bytes (This input is added to the transaction for fee purposes).
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - one P2WSH output: 43 bytes
  - one P2WPKH output: 31 bytes (This input is added to the transaction for fee purposes)
  - locktime: 4 bytes

The witness to claim a 2-of-2 multisignature output includes (222 bytes):

- number of witness elements: 1 byte
- 0: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- signature length: 1 byte
- signature: 73 bytes
- witness_script_length: 1 byte
- witness_script: 71 bytes

The witness script is as follows (71 bytes):
`OP_2` (1 byte)
⟨`pubkey1`⟩ (1 byte for `OP_DATA` and 33 bytes for the public key)

⟨pubkey2⟩ (1 byte for OP_DATA and 33 bytes for the public key)
OP_2 (1 byte)
OP_CHECKMULTSIG (1 byte)
The witness to claim a P2WPKH output includes (108 bytes):

- signature length: 1 byte
- signature: 73 bytes
- public key length: 1 byte
- public key: 33 bytes

Settlement transaction The settlement transaction (with two P2WPKH and m HTLC P2WSH outputs) contains (304 byte of witness data and 113+43m bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim the update transaction's output: 302 bytes.
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - two P2WPKH outputs: 62 bytes bytes
  - m P2WSH outputs: 43m bytes
  - locktime: 4 bytes

The witness to claim the update transaction's output includes (302 bytes):

- number of witness elements: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- signature length: 1 byte
- signature: 73 bytes
- 1: 1 byte
- witness_script_length: 1 byte
- witness_script: 151 bytes

The witness script is as follows [18] (151 bytes):

```
OP_IF (1 byte)
   # settlement
   2 (1 byte)
   ⟨set_pubkeyA⟩ (1 byte for OP_DATA and 33 bytes for the public key)
   ⟨set_pubkeyB⟩ (1 byte for OP_DATA and 33 bytes for the public key)
   2 (1 byte)
   OP_CHECKMULTISIG (1 byte)
OP_ELSE (1 byte)
   # Update
```

```
    ⟨delay T⟩ (4 bytes)
    OP_CHECKSEQUENCEVERIFY (1 bytes)
    OP_DROP (1 bytes)
    2 (1 bytes)
    ⟨upd_pubkeyA⟩ (1 byte for OP_DATA and 33 bytes for the public key)
    ⟨upd_pubkeyB⟩ (1 byte for OP_DATA and 33 bytes for the public key)
    2 (1 bytes)
    OP_CHECKMULTISIG (1 bytes)
OP_ENDIF (1 bytes)
```

Redeem' transaction

The Redeem' transaction contains 212 bytes of witness data and 82 bytes of non-witness data (For more details, see the figures for the Generalized channel).

Claimback' transaction

The Claimback' transaction contains 180 bytes of witness data and 82 bytes of non-witness data (For more details, see the figures for the Generalized channel).

So, one update, one settlement, $\frac{m}{2}$ Redeem' and $\frac{m}{2}$ Claimback' transactions contain $636+196m$ bytes of witness data and $238+125m$ bytes of non-witness data, in total.

2) Dishonest Closure: If an old update transaction is published, the victim must publish the latest update transaction and then the latest settlement transaction and finally all Claimback' and Redeem' transaction. In the previous section, we computed the size of all mentioned transactions except an update transaction which spends output of an old update transaction. We compute the size of such a transaction, hereinafter.

The latest update transaction contains (412 byte of witness data and 125 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim the old update transaction's output: 302 bytes.
  - witness to claim a P2WPKH output: 108 bytes (This input is added to the transaction for fee purposes).
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - one P2WSH output: 43 bytes
  - one P2WPKH output: 31 bytes (This input is added to the transaction for fee purposes)
  - locktime: 4 bytes

The witness to claim the old update transaction's output includes (302 bytes):

- number of witness elements: 1 byte
- signature length: 1 byte

- signature: 73 bytes
- signature length: 1 byte
- signature: 73 bytes
- 0: 1 byte
- witness_script_length: 1 byte
- witness_script: 151 bytes

So, one old update transaction, the latest update transaction, the latest settlement transaction, $\frac{m}{2}$ Redeem' and $\frac{m}{2}$ Claimback' transactions contain 940+196m bytes of witness data and 332+125m bytes of non-witness data, in total.

3) Number of Operations: For each channel update, channel parties must create the following signatures:

- signature on the update transaction for their counter-party
- signature on the settlement transaction for their counter-party

Each party must verify two above signatures from their counter-party. Each party must also perform one exponentiation operation to create a settlement key pair to be used in the condition of the update transaction.

H.5 FPPW channel

1) Non-collaborative Closure: In a non-collaborative closure, the latest commit and split transactions are published on-chain. Then, we assume half of the HTLC outputs are redeemed by the payee (via Redeem' transactions) and the rest are claimed back by the payer (via Claimback' transaction).

Commit transaction

The commit transaction contains (224 bytes of witness data and 137 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim a 2-of-2 multisignature output: 222 bytes.
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - two P2WSH output: 86 bytes
  - locktime: 4 bytes

Split transaction

The split transaction (with two P2WPKH and m HTLC P2WSH outputs) contains (338 bytes of witness data and 113+43m bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness for commit transaction's output: 336 bytes.

- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - two P2WPKH outputs: 62 bytes bytes
  - m P2WSH outputs: $43m$ bytes
  - locktime: 4 bytes

The witness to claim the commit transaction's output includes (336 bytes):

- number of witness elements: 1 byte
- 0: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- signature length: 1 byte
- signature: 73 bytes
- 0: 1 byte
- witness_script_length: 1 byte
- witness_script: 184 bytes

The witness script is as follows (184 bytes):

```
OP_IF (1 byte)
    # Revocation
    3 (1 byte)
    ⟨Rev_pubkeyA⟩ (1 byte for OP_DATA and 33 bytes for the public key)
    ⟨Rev_pubkeyB⟩ (1 byte for OP_DATA and 33 bytes for the public key)
    ⟨Rev_pubkeyW⟩ (1 byte for OP_DATA and 33 bytes for the public key)
    3 (1 byte)
    OP_CHECKMULTISIG (1 byte)
OP_ELSE (1 byte)
    # Split
    ⟨delay t ⟩ (4 bytes)
    OP_CHECKSEQUENCEVERIFY (1 byte)
    OP_DROP (1 byte)
    2 (1 byte)
    ⟨Spl_pubkeyA⟩ (1 byte for OP_DATA and 33 bytes for the public key)
    ⟨Spl_pubkeyB⟩ (1 byte for OP_DATA and 33 bytes for the public key)
    2 (1 byte)
OP_ENDIF (1 byte)
```

Redeem' transaction

The Redeem' transaction contains 212 bytes of witness data and 82 bytes of non-witness data (For more details, see the figures for the Generalized channel).

Claimback' transaction

The Claimback' transaction contains 180 bytes of witness data and 82 bytes of non-witness data (For more details, see the figures for the Generalized channel).

So, one commit, one split, $\frac{m}{2}$ Redeem' and $\frac{m}{2}$ Claimback' transactions contain 562+196m bytes of witness data and 250+125m bytes of non-witness data, in total.

2) Dishonest Closure: If a revoked commit transaction is published, the victim must publish a revocation transaction to spend both outputs of the commit transaction and sends all the channel funds to his own P2WSH address. We computed the size of a commit transaction in the previous section.

The revocation transaction contains (897 bytes of witness data and 94 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim the first output of the commit transaction: 410 bytes.
  - witness to claim the second output of the commit transaction: 485 bytes.
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - one P2WSH outputs: 43 bytes bytes
  - locktime: 4 bytes

The witness to claim the first output of the commit transaction includes (410 bytes):

- number of witness elements: 1 byte
- 0: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- signature length: 1 byte
- signature: 73 bytes
- signature length: 1 byte
- signature: 73 bytes
- 1: 1 byte
- witness_script_length: 1 byte
- witness_script: 184 bytes

The witness to claim the second output of the commit transaction includes (485 bytes):

- number of witness elements: 1 byte
- 0: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- signature length: 1 byte
- signature: 73 bytes
- signature length: 1 byte

- signature: 73 bytes
- 1: 1 byte
- witness_script_length: 1 byte
- witness_script: 259 bytes

The witness script is as follows (259 bytes):

```
OP_IF (1 byte)
    # Revocation
    3 (1 byte)
    ⟨Rev_pubkeyA⟩ (1 byte for OP_DATA and 33 bytes for the public key)
    ⟨Rev_pubkeyB⟩ (1 byte for OP_DATA and 33 bytes for the public key)
    ⟨Rev_pubkeyW⟩ (1 byte for OP_DATA and 33 bytes for the public key)
    3 (1 byte)
    OP_CHECKMULTISIG (1 byte)
OP_ELSE (1 byte)
    ⟨delay t⟩ (4 byte)
    OP_CHECKSEQUENCEVERIFY (1 byte)
    OP_DROP (1 byte)
    OP_IF (1 byte)
        # Penalty1 or Penalty2 by party B
        2 (1 byte)
        ⟨Pen_pubkeyB⟩ (1 byte for OP_DATA and 33 bytes for the public key)
        ⟨YA⟩ (1 byte for OP_DATA and 33 bytes for the public key)
        2 (1 byte)
        OP_CHECKMULTISIG (1 byte)
    OP_ELSE (1 byte)
        # Penalty1 or Penalty2 by party A
        2 (1 byte)
        ⟨Pen_pubkeyA⟩ (1 byte for OP_DATA and 33 bytes for the public key)
        ⟨YB⟩ (1 byte for OP_DATA and 33 bytes for the public key)
        2 (1 byte)
        OP_CHECKMULTISIG (1 byte)
    OP_ENDIF (1 byte)
OP_ENDIF (1 byte)
```

So, commit and revocation transactions contain 1121 bytes of witness data and 231 bytes of non-witness data, in total.

3) Number of Operations: For each channel update, channel parties must create the following signatures:

- pre-signature on the commit transaction for their counter-party
- signature on the split transaction for their counter-party
- two signature on the revocation transactions for their counter-party and the watchtower
- two signature on penalty transactions for their counter-party

Each party must verify 6 above signatures from their counter-party as well as four signatures from the watchtower on the revocation transaction and penalty transactions. Each party must also perform one exponentiation operation to create a statement/witness pair.

H.6 Cerberus channel

To compute the total number of published bytes in different channel closure scenarios, we assume that the channel states do not contain any HTLC output.

1) Non-collaborative Closure: In this scenario, a commit transaction is published on-chain. We compute the size of this transaction, hereinafter.

Commit transaction

The commit transaction contains (224 bytes of witness data and 137 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim a 2-of-2 multisignature output: 222 bytes.
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - two P2WSH outputs: 86 bytes
  - locktime: 4 bytes

2) Dishonest Closure: If a revoked commit transaction is published, the victim uses the corresponding revocation transaction to claim both outputs of the commit transaction. The revocation transaction contains (534 bytes of witness data and 123 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim the to_local output: 266 bytes.
  - witness to claim the to_remote output: 266 bytes.
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - 2 inputs, each including 41 bytes: in total 82 bytes
  - number of outputs: 1 byte
  - one P2WPKH output: 31 bytes
  - locktime: 4 bytes

The witness to claim the to_local (and similarly to_remote) output includes (266 bytes):

- number of witness elements: 1 byte

- 0: 1 byte
- signature length: 1 byte
- signature: 73 bytes
- signature length: 1 byte
- signature: 73 bytes
- witness_script_length: 1 byte
- witness_script: 115 bytes

The witness script is as follows (115 bytes) [11]:

```
OP_IF (1 byte)
   # Revocation
   2 (1 byte)
   ⟨revocation_pubkey1⟩ (1 byte for OP_DATA and 33 bytes for the public key)
   ⟨revocation_pubkey2⟩ (1 byte for OP_DATA and 33 bytes for the public key)
   2 (1 byte)
   OP_CHECKMULTISIG (1 byte)
OP_ELSE (1 byte)
   # Normal
   ⟨delay T⟩ (4 bytes)
   OP_CHECKSEQUENCEVERIFY (1 bytes)
   OP_DROP (1 bytes)
   ⟨delayed_pubkeyA⟩ (1 byte for OP_DATA and 33 bytes for the public key)
   OP_CHECKMULTISIG (1 bytes)
OP_ENDIF (1 bytes)
```

So, one commit transaction and its revocation transaction contain 758 bytes of witness data and 260 bytes of non-witness data.

3) Number of Operations: For each channel update, channel parties must create the following signatures:

- signature on the commit transaction for their counter-party
- signature on their counter-party's revocation transaction for their counter-party
- signature on their own revocation transaction for their watchtower

Each party must verify 2 signatures from their counter-party on commit and revocation transactions as well as two signatures from the watchtower on the revocation transaction. Each party must also verify two signatures from the watchtower on penalty transaction.

H.7 Outpost channel

To compute the total number of published bytes in different channel closure scenarios, we assume that the channel states do not contain any HTLC output.

1) Non-collaborative Closure: In this scenario, a commit transaction 1, an auxiliary transaction and a commit transaction 2 are published on-chain. We compute the size of each transaction, hereinafter.

Commit transaction 1

The commit transaction 1 contains (224 bytes of witness data and 168 bytes of non-witness data):

– witness data:
  - header: 2 bytes
  - witness to claim a 2-of-2 multisignature output: 222 bytes.

– non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - two P2WSH outputs: 86 bytes
  - one P2WPKH output: 31 bytes
  - locktime: 4 bytes

Auxiliary transaction

The auxiliary transaction contains (224 bytes of witness data and 240 bytes of non-witness data):

– witness data:
  - header: 2 bytes
  - witness to claim a 2-of-2 multisignature output: 222 bytes

– non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - one P2WSH outputs: 43 bytes
  - one OP_RETURN output: two 73 byte signatures
  - locktime: 4 bytes

Commit transaction 2

The commit transaction 2 contains (446 bytes of witness data and 123 bytes of non-witness data):

– witness data:
  - header: 2 bytes
  - witness to claim the multisignature condition in the first output of the commit transaction 1: 222 bytes.
  - witness to claim the multisignature condition in the first output of the auxiliary transaction: 222 bytes.

– non-witness data:
  - version: 4 bytes

  - number of inputs: 1 byte
  - two inputs: 82 bytes
  - number of outputs: 1 byte
  - one P2WPKH outputs: 31 bytes
  - locktime: 4 bytes

So, commit transaction 1, auxiliary transaction and commit transaction 2 contain 894 bytes of witness data and 531 bytes of non-witness data.

2) Dishonest Closure: In the dishonest closure, a revoked commit transaction 1 and then its corresponding auxiliary transaction and justice transaction are published on the blockchain. The justice transaction contains (224 bytes of witness data and 82 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim the multisignature condition in the first output of the commit transaction 1: 222 bytes.
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input with 41 bytes
  - number of outputs: 1 byte
  - one P2WPKH output: 31 bytes
  - locktime: 4 bytes

So, commit transaction 1, its corresponding auxiliary transaction and justice transaction contain 672 bytes of witness data and 490 bytes of non-witness data.

3) Number of Operations: For each channel update, channel parties must create the following signatures:

- signature on the commit transaction 1 for their counter-party
- signature on the auxiliary transaction for their counter-party
- 2 signatures on the commit transaction 2 for their counter-party
- signature on the justice transaction for their counter-party

Each party must verify 4 signatures from their counter-party on the above mentioned transactions.

H.8 Sleepy channel

To compute the total number of published bytes in different channel closure scenarios, we assume that the channel states do not contain any HTLC output.

1) Non-collaborative Closure: In this scenario, a payment transaction, a fast-finish transaction and an enabler transaction are published on-chain. We compute the size of each transaction, hereinafter.

Payment transaction

The payment transaction contains (224 bytes of witness data and 168 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim a 2-of-2 multisignature condition: 222 bytes
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - two P2WSH outputs: 86 bytes
  - one P2WPKH output: 31 bytes
  - locktime: 4 bytes

Fast-finish transaction

The fast-finish transaction contains (224 bytes of witness data and 125 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim a 2-of-2 multisignature condition: 222 bytes
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one input: 41 bytes
  - number of outputs: 1 byte
  - one P2WSH outputs: 43 bytes
  - one P2WPKH output: 31 bytes
  - locktime: 4 bytes

Enabler transaction

The enabler transaction contains (446 bytes of witness data and 123 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim the 2-of-2 multisignature condition in the second output of the payment transaction: 222 bytes
  - witness to claim the 2-of-2 multisignature condition in the first output of the fast-finish transaction: 222 bytes
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - two inputs: 82 bytes
  - number of outputs: 1 byte
  - one P2WPKH output: 31 bytes
  - locktime: 4 bytes

So, payment transaction, fast-payment transaction and enabler transaction contain 894 bytes of witness data and 416 bytes of non-witness data.

1) Dishonest Closure: In this scenario, payment transaction, punishment transaction and fast-finish transaction are published on the blockchain. We compute the size of each transaction, hereinafter. **Payment transaction**

The payment transaction contains 224 bytes of witness data and 168 bytes of non-witness data.

Punishment transaction

The punishment transaction contains (224 bytes of witness data and 82 bytes of non-witness data):

- witness data:
  - header: 2 bytes
  - witness to claim the 2-of-2 multisignature condition in the second output of the payment transaction: 222 bytes
- non-witness data:
  - version: 4 bytes
  - number of inputs: 1 byte
  - one inputs: 41 bytes
  - number of outputs: 1 byte
  - one P2WPKH output: 31 bytes
  - locktime: 4 bytes

Fast-finish transaction

The fast-finish transaction contains 224 bytes of witness data and 125 bytes of non-witness data.

So, payment transaction, punishment transaction and fast-payment transaction contain 672 bytes of witness data and 375 bytes of non-witness data.

3) Number of Operations: For each channel update, channel parties must create the following signatures:

- signature on the payment transaction for their counter-party
- signature on the fast-finish transaction for their counter-party
- signature on the punishment transaction for their counter-party
- two signatures on the enabler transaction for their counter-party

Each party must verify 5 signatures from their counter-party on the above mentioned transactions.