Bitcoin-Compatible Virtual Channels

Lukas Aumayr
lukas.aumayr@tuwien.ac.at
TU Wien

Oğuzhan Ersoy
O.Ersoy@tudelft.nl
TU Delft

Andreas Erwig
andreas.erwig@tu-darmstadt.de
TU Darmstadt

Sebastian Faust
sebastian.faust@tu-darmstadt.de
TU Darmstadt

Kristina Hostáková
kristina.hostakova@tu-darmstadt.de
TU Darmstadt

Matteo Maffei
matteo.maffei@tuwien.ac.at
TU Wien

Pedro Moreno-Sanchez
pedro.sanchez@tuwien.ac.at
TU Wien

Siavash Riahi
siavash.riahi@tu-darmstadt.de
TU Darmstadt

ABSTRACT

Current permissionless cryptocurrencies such as Bitcoin suffer from a limited transaction rate and slow confirmation time, which hinders their large scale adoption. Payment channels are one of the most promising solutions to address these problems, as they allow two end-points of the channel to perform arbitrarily many payments in a peer-to-peer fashion while uploading only two transactions on the blockchain. This concept has been generalized into payment-channel networks where a path of payment channels is used to settle the payment between two users that might not share a channel between them. However, this approach requires the active involvement of each user in the path, making the system less reliable (they might be offline), more expensive (they charge fees per payment) and slower (intermediaries need to be actively involved in the payment). To mitigate this issue, recent work has introduced the concept of virtual channels, which involve intermediaries only in the initial creation of a bridge between payer and payee, who can later on independently perform arbitrarily many off-chain transactions. Unfortunately, existing constructions are only available for Ethereum, as they rely on its account model and Turing-complete scripting language. The realization of virtual channels in other blockchain technologies with limited scripting capabilities, like Bitcoin, was considered so far an open challenge.

In this work, we present the first virtual channel protocols that are built on the UTXO-model and require a script language supporting only a digital signature scheme and a timelock functionality, being thus backwards compatible with virtually every cryptocurrency, including Bitcoin. We formalize the security properties of virtual channels as an ideal functionality in the Universal Composability framework, and prove that our protocol constitutes a secure realization thereof. We have prototyped and evaluated our protocol on the Bitcoin blockchain, demonstrating its efficiency: for n sequential payments, they require an off-chain exchange of $11 + 2 \cdot (n - 1)$ transactions or a total of $4219 + 695 \cdot (n - 1)$ bytes, with no on-chain footprint in the optimistic case.

1 INTRODUCTION

Permissionless cryptocurrencies such as Bitcoin [19] have spurred increasing interest over the last years, putting forward a revolutionary, from both a technical and economical point of view, payment paradigm. Instead of relying on a central authority for transaction validation and accounting, Bitcoin relies at its core on a decentralized consensus protocol for these tasks. The consensus protocol establishes and maintains a distributed ledger that tracks each single transaction, thereby enabling public verifiability. This approach, however, severely limits the transaction throughput and confirmation time, which in the case of Bitcoin is around ten transactions per second, and confirmation of an individual transaction can take up to 60 minutes. This is in stark contrast to central payment providers that offer instantaneous transaction confirmation and support orders of magnitude higher transaction throughput. These scalability issues hinder permissionless cryptocurrencies such as Bitcoin from serving a growing base of payments.

Within other research efforts [6, 14, 23], payment channels [1] have emerged as one of the most promising scalability solution, being currently deployed in Bitcoin as the so-called Lightning network [21], which at the time of writing hosts deposits worth more than 7M USD. A payment channel enables an arbitrary number of payments between users while committing only two transactions onto the blockchain. In a bit more detail, a payment channel between Alice and Bob is first created by a single on-chain transaction that deposits Bitcoins into a multi-signature address controlled by both users. The parties additionally ensure that they can get their Bitcoins back at a mutually agreed expiration time. They can then pay to each other (possibly many times) by exchanging authenticated off-chain messages that represent an update of their share of coins in the multi-signature address. The payment channel is finally closed when a user submits the last authenticated distribution of Bitcoins to the blockchain (or after the channel has expired).

Interestingly, it is possible to leverage a path of opened payment channels from the sender to the receiver with enough capacity to settle their payments off-chain, thereby creating a payment channel network (PCN) [17, 21]. Assume that Alice wants to pay Bob and they do not have a payment channel between each other but rather are connected through an intermediary user Ingrid. Upon a successful off-chain update of the payment channel between Alice and Ingrid, the latter would update her payment channel with Bob to make the overall transaction effective. The key challenge is how to perform the sequence of updates atomically in order to prevent Ingrid from stealing the money from Alice without paying to Bob. The standard technique for constructing payment channel networks requires the intermediary (e.g., Ingrid in the example from above) to be actively involved in each payment. This has

the disadvantage of making the system less reliable (e.g., Ingrid might have to go offline), increasing the latency of each payment, augmenting its costs since each intermediary charges a fee per transaction, and revealing possibly sensitive payment information to the intermediaries [15, 20, 22].

An alternative approach for connecting multiple payment channels was introduced by Dziembowski et al. [11]. They propose the concept of *virtual channels* – an off-chain protocol that enables direct off-chain transactions without the involvement of the intermediary. Following our running example, a virtual channel can be created between Alice and Bob using their individual payment channels with Ingrid. Ingrid must collaborate with Alice and Bob only to create such virtual channel, which can then be used by Alice and Bob to perform arbitrarily many off-chain payments without involving Ingrid. Virtual channels offer strong security guarantees: each user does not lose money even if the others collude. A salient application of virtual payment channels are so called payment hubs [11]: since establishing a payment channel requires a deposit and active monitoring, the number of channels a user can establish is limited. With payment hubs [11], users have to establish just one payment channel with the hub and can then dynamically open and close virtual channels between each other on demand. Interestingly, since in a virtual channel the hub is not involved in the individual payments, even transactions worth fractions of cents can be carried out with low latency.

The design of secure virtual channels is extremely challenging, since, as previously mentioned, it has to account for all possible compromise and collusion scenarios. For this purpose, existing virtual channel constructions [11] require smart contracts programmed over a Turing-complete language and the account model, as supported in Ethereum. This significantly simplifies the construction, since the deposit of a channel and its distribution between the end-points are stored in memory and can be programmatically updated, but it limits in fact the deployment of virtual channels to Ethereum.

It was an *open question* until now if virtual channels could be implemented at all in UTXO-based cryptocurrencies featuring only a limited scripting language, like Bitcoin and virtually all other permissionless cryptocurrencies. In this setting, the channel deposit and its distribution between the end-points is represented by past transactions, which are persistent by nature, and the update of both the deposit and the distribution requires sophisticated cryptographic techniques and carefully orchestrated transaction protocols, as opposed to mere programming.

1.1 Our contributions

In this work, we give a positive answer to the above question, developing novel protocols for building a virtual channel hub over Bitcoin and providing a comprehensive formal analysis of our constructions. Concretely, our contributions are summarized below.

- We present the first protocols for virtual channel hubs that are built on the UTXO-model and require a scripting language supporting only a digital signature scheme and timelock functionality, being thus backwards compatible with virtually every cryptocurrency, including Bitcoin. Since in the Lightning network currently only 10 supernodes are involved in more than 25% of all channels, our technique can be used to reduce the load on these nodes, and thereby help to reduce latency.
- We offer two constructions that differ on whether (i) the virtual channel is guaranteed to stay off-chain for an encoded validity period, or (ii) the intermediary Ingrid can decide to offload the virtual channel (i.e., convert it into a direct channel between Alice and Bob, which requires various on-chain transactions), thereby removing its involvement in it. These two variants support different business and functionality models, analogous to non-preemptible and preemptible virtual machines in the cloud setting, with Ingrid playing the role of the service provider.
- We formalize the security properties of virtual channels as an ideal functionality in the UC framework [7], proving that our protocol constitutes a secure realization thereof. Since our virtual channels are built in the UTXO-model, our ideal functionality and formalization significantly differs from earlier work [11].
- We evaluate our protocol and show that for n sequential payments, they require an off-chain exchange of $11 + 2 \cdot (n-1)$ transactions or a total of $4219 + 695 \cdot (n-1)$ bytes, as compared to $8 \cdot n$ transactions or $3026 \cdot n$ bytes when Ingrid routes the payment actively through the PCN. We have interacted with the Bitcoin blockchain to store the required transactions, demonstrating the backwards compatibility of our protocol.

As a result of this work, we enable for the first time in Bitcoin off-chain payments between users connected by payment channels via a hub without requiring the presence of any intermediary, thereby increasing the reliability and, at the same time, reducing the latency and costs of Bitcoin PCNs.

2 BACKGROUND & SOLUTION OVERVIEW

In this section, we first introduce the notation used in this paper. We then overview the basics of payment and virtual channels, referring the reader to [3, 11, 17, 18] for further details. We finally overview our solution for Bitcoin-compatible virtual channels.

2.1 Notation

We adopt the notation from [4] and we shorty review it below.

Attribute tuples. Let T be a tuple of values, we refer to these values as attributes. Each attribute in T is identified by a unique keyword, e.g., attr and referred to as $T.\mathsf{attr}$.

Outputs and transactions. We focus on blockchains based on the Unspent Transaction Output (UTXO) model, such as Bitcoin. In the UTXO model, coins are held in *outputs* of transactions. Formally, an output θ is an attribute tuple $(\theta.\mathsf{cash}, \theta.\varphi)$, where $\theta.\mathsf{cash}$ denotes the amount of coins associated to the output and $\theta.\varphi$ denotes the conditions that need to be satisfied in order to spend the output. The condition $\theta.\varphi$ can contain any script supported by the considered blockchain. We say that a user P controls or owns an output θ if $\theta.\varphi$ contains only a signature verification w.r.t. the public key of P.

In a nutshell, a *transaction* in the UTXO model, maps one or more existing outputs to a list of new outputs. The existing outputs are called *transaction inputs*. Formally, a transaction tx is an attribute tuple and consists of the following attributes $(\mathsf{tx.txid}, \mathsf{tx.Input}, \mathsf{tx.Output}, \mathsf{tx.TimeLock}, \mathsf{tx.Witness})$. The attribute $\mathsf{tx.txid} \in \{0,1\}^*$ is called the identifier of the transaction. The identifier is calculated

as tx.txid := $\mathcal{H}([\mathsf{tx}])$, where $\mathcal{H}$ is a hash function which is modeled as a random oracle and [tx] is the *body of the transaction* defined as [tx] := (tx.Input, tx.Output, tx.TimeLock). The attribute tx.Input is a vector of strings which identify the inputs of tx. Similarly, the outputs of the transaction tx.Output is the vector of new outputs of the transaction tx. The attribute tx.TimeLock $\in \mathbb{N} \cup \{0\}$ denotes the absolute time-lock of the transaction, which intuitively means that transaction tx will not be accepted by the blockchain before the round defined by tx.TimeLock. The time-lock is by default set to 0, meaning that no time-lock is in place. Lastly, tx.Witness $\in \{0,1\}^*$ called the transaction's witness, contains the witness of the transaction that is required to spend the transaction inputs.

We use charts in order to visualize the transaction flow in the rest of this work in order to improve readability and provide a road map for the protocol execution. We first explain the notation used in the charts and how they should be read. Transactions are shown using rectangles with rounded corners. Double edge rectangles are used to represent transactions that are already published on the blockchain. Single edge rectangles are transactions that could be published on the blockchain but they are not yet. Each transaction contains one or more boxes (i.e., with squared corners) that represent the outputs of that transaction. The amount of coins allocated to each output is written inside the output box. In addition, the output condition is written on the arrow coming from the output.

In order to be concise, we use the following abbreviations for the frequently used conditions. Most outputs can only be spent by a transaction which is signed by a set of parties. In order to depict this condition, we write the public keys of all these parties *below* the arrow. Other conditions are written *above* the arrow. In other words, the "owners" of the output are identified given the public keys below the arrows and the additional spending conditions are given above the arrows. The output script can have a relative time lock i.e. a condition that is satisfied if and only if at least t rounds are passed since the transaction was published on the blockchain. We denote this output condition writing the string "$+t$" *above* the arrow. In addition to relative time locks, an output can also have an absolute time lock, i.e., a condition that is satisfied only if t rounds elapsed since the blockchain was created and the first transaction was posted on it. We write the string "$> t$" *above* the arrow for this condition. Lastly, an output's spending condition might be a disjunction of multiple conditions. In other words it can be written as $\varphi = \varphi_1 \vee \cdots \vee \varphi_n$ for some $n \in \mathbb{N}$ where φ is the output script. In this case, we add a diamond shape to the corresponding transaction output. Each of the subconditions φ_i is then written above a separate arrow. An example is given in Figure 1.

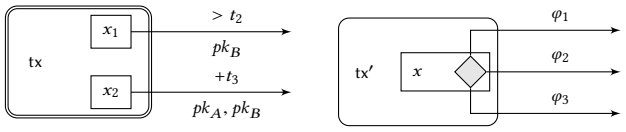


Figure 1: (Left) Transaction tx is published on the blockchain. The output of value x_1 can be spent by a transaction signed w.r.t. pk_B after round t_2, and the output of value x_2 can be spent by a transaction signed w.r.t. pk_A and pk_B but only if at least t rounds passed since tx was accepted by the blockchain. (Right) Transaction tx′ is not published on the ledger. Its only output, which is of value x, can be spent by a transaction whose witness satisfies the output condition $\varphi_1 \vee \varphi_2 \vee \varphi_3$.

2.2 Payment channels

A payment channel enables arbitrarily many transactions between users while requiring only two on-chain transactions. The cornerstone of payment channels is depositing coins into an output controlled by two users, who then authorize new deposit balances in a peer-to-peer fashion while having the guarantee that all coins are refunded at a mutually agreed time. In a bit more detail, a payment channel has three operations: *open*, *update* and *close*. We necessarily keep the description short and refer to [4, 14] for further reading.

Open: Assume that Alice and Bob want to create a payment channel with an initial deposit of x_A and x_B coins respectively. For that, Alice and Bob agree on a *funding transaction* (that we denote by $\mathsf{TX_f}$) that sets as inputs two outputs controlled by Alice and Bob holding x_A and x_B coins respectively and transfers them to an output controlled by both Alice and Bob. When $\mathsf{TX_f}$ is added to the blockchain, the payment channel is effectively open.
Update: Assume now that Alice wants to pay $\alpha \leq x_A$ coins to Bob. For that, they create a new *commit transaction* $\mathsf{TX_c}$ representing the commitment from both users to the new channel state. The commit transaction spends the output of $\mathsf{TX_f}$ into two new outputs: (i) one holding $x_A - \alpha$ coins controlled by Alice; and (ii) the other holding $x_B + \alpha$ coins controlled by Bob. Finally, parties exchange the signatures on the commit transaction. At this point, Alice (resp. Bob) could add $\mathsf{TX_c}$ to the blockchain. Instead, they keep it locally in their memory and overwrite it when they agree on another commitment transaction $\overline{\mathsf{TX}}_\mathsf{c}$ representing a newer channel state. This, however, leads to several commitment transactions that can possibly be added to the blockchain. Since all of them are spending the same output, only one can be accepted by the blockchain. Since it is impossible to prevent a malicious user from publishing an old commit transaction, payment channels require a mechanism that punishes such behavior. Such mechanism is typically called *revocation* and enables that an honest user can take all the coins locked in the channel if the dishonest user publishes an old commitment transaction.
Close: Assume finally that Alice and Bob no longer wish to use the channel. Then, they can collaboratively close the channel by submitting the last commitment transaction $\overline{\mathsf{TX}}_\mathsf{c}$ that they have agreed upon to the blockchain. After it is accepted, the coins initially locked at the channel creation are redistributed to both users according to the last agreed state. As aforementioned, if one of the users submits an old commitment transaction instead, the counterparty can punish the former through the revocation mechanism.

The Lightning Network [21] defines the state-of-the-art payment channel construction for Bitcoin. A recent work [4] proposes *generalized channels*, an alternative construction for payment channels (see Figure 2) that reduces the communication complexity both on-chain and off-chain in case of revocation while providing the same functionality as the Lightning Network and extending it to any transaction that can be expressed with Bitcoin scripting language. We instantiate our virtual channel protocol within the framework of generalized channels for efficiency and generality reasons, but the

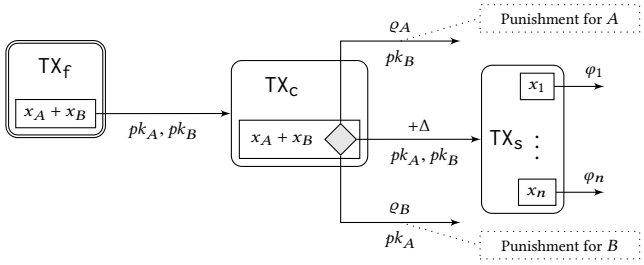


Figure 2: A generalized channel in the state $((x_1, \varphi_1), \ldots, (x_n, \varphi_n))$. The value of Δ upper bounds the time needed to publish a transaction on a blockchain. The condition ϱ_A represents the verification of A' revocation secret and ϱ_B represents the verification of B' revocation secret.

same ideas could be integrated in the original Lightning Network construction too. To avoid confusion with the term channel, we hereby use the term ledger channel to denote a generalized channel as described in [4].

Limitations of ledger channels A ledger channel enables payments between the two channel counterparties only. To overcome this limitation, the Lightning Network integrates a cryptographic protocol for routing payments between users not sharing a ledger channel (say Alice and Bob) but being instead connected through an intermediary user (Ingrid), thereby enabling a channel network [17]. This approach has unfortunately several drawbacks: (i) *high-cost* as Ingrid charges a fee for each payment between Alice and Bob; (ii) *low reliability* because the success of payments relies on Ingrid being online; and (iii) *low privacy* as Ingrid observes each payment between Alice and Bob. *Virtual channels* [10] address these problems, allowing Alice and Bob to open a (virtual) channel between each other, along which payments can be directly routed bypassing Ingrid, who is only required to participate in the initial creation of the virtual channel and can otherwise be offline and anyway does not see any payment routed on the virtual channel between Alice and Bob. However, the construction in [10] builds upon the account model (instead of UTXO) and relies on Turing-complete smart contracts, and it is thus not supported in Bitcoin or virtually any other cryptocurrency. We show for the first time in this work that virtual channels do not require Turing-complete smart contracts, devising a construction that only makes use of a limited set of operations, all of them supported in the current Bitcoin scripting language, while providing the security guarantees of interest.

2.3 Overview of our solution

A virtual channel between Alice and Bob requires an intermediary (say Ingrid) and it is constructed on top of two ledger channels, namely, the ledger channels Alice-Ingrid and Ingrid-Bob. The cornerstone of our approach is to construct the virtual channel as a normal ledger channel with the only but key difference that its funding transaction is not funded directly by a transaction already published on the blockchain but rather by coins from the current state of the two underlying ledger channels. Informally speaking, one can thus see a virtual channel as a "ledger channel on top of two ledger channels".

The key challenge is thus to design the virtual channel funding mechanism in such a way that the involved honest users never lose coins. Indeed, some parties might misbehave but, in contrast to standard ledger channels who rely on funding transactions known to the two end-points, virtual channels rely on funding transactions from the two underlying ledger channels, and neither Alice nor Bob have control over both of them.

This opens the door to attacks and collusion scenarios that are not present in the case of standard payment channels. To illustrate this, assume that Alice and Bob opened a virtual channel where Alice contributed x coins from the ledger channel Alice-Ingrid and Bob contributed y coins from the ledger channel Ingrid-Bob. Further assume that after several payments between Alice and Bob, the current balance of the virtual channel is such that Bob holds all $x+y$ coins. At this point, Alice could collude with Ingrid and close their ledger channel into a state other than the one that contributed x coins to the virtual channel. Bob cannot longer publish the funding transaction of the virtual channel (i.e., the required inputs are not on a blockchain transaction), effectively losing the balance of $x + y$ coins. A similar situation can be considered where Alice loses coins from the virtual channel when Bob and Ingrid collude.

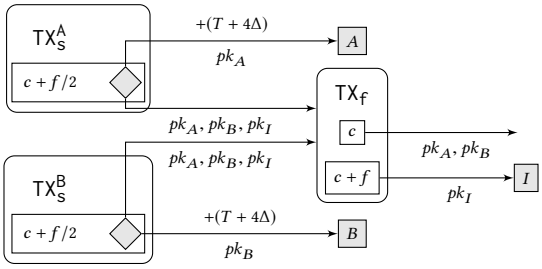
To solve this problem, we design a novel 3-party synchronization protocol that compensates Alice or Bob if they try to publish the virtual channel funding transaction honestly and this cannot be done because one of the ledger channels has been maliciously updated or closed.

In particular, we have two different designs for the 3-party synchronization. Assume that Alice and Bob have opened a virtual channel with a capacity of c coins. Both protocols have in common that Ingrid must also contribute c coins as *collateral* to be able to punish her when colluding with any of the other users, that is, Alice or Bob. The two protocols, however, differ in the operations to *offload the virtual channel*. Given that, we first describe the offloading of a virtual channel and then overview our two designs.

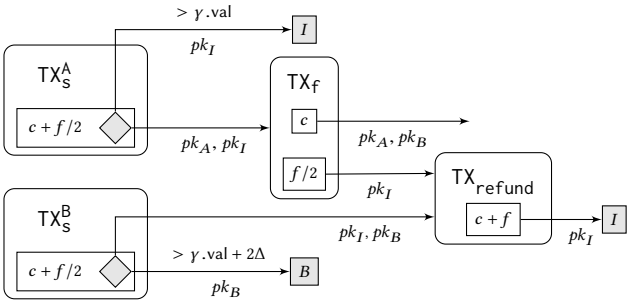
Virtual channel offload. The offload of a virtual channel is its conversion into a ledger channel, meaning that its funding transaction is published on-chain. This requires two steps. First, close the underlying ledger channels to publish the inputs required for the funding transaction of the virtual channel on the blockchain. Second, publish the funding transaction itself.

Our two designs differ in the role that Ingrid plays during the offload of a virtual channel between Alice and Bob. In the first design, Ingrid has to be proactive and she is responsible for offloading the virtual channel at any time. In the second design, no one except for Alice can offload the virtual channel before a certain *validity* timeout. Given this structural difference, we tag this second design as *virtual channels with validity (VC-V)* and the first design as *virtual channels without validity (VC-NV)*.

These two designs are meant to support different business models for virtual channels. On the one hand, similar to preemptible virtual machines in cloud services, VC-NV allows Ingrid, which can be seen as the service provider in virtual channels, to offload the virtual channel on her own and unilaterally. Ingrid may do so to get her collateral back (e.g., to reuse it in a financially more profitable or more active virtual channel). We note that, if they wish, both Alice and Bob can also offload their virtual channel if Ingrid does not do so. On the other hand, similar to non-preemptible virtual machines,



(a) Funding of a virtual channel γ without validity.



(b) Funding of a virtual channel γ with validity $\gamma.\mathsf{val}$.

Figure 3: Funding of virtual channel. In both figures, $c := \gamma.\mathsf{cash}$, $f := \gamma.\mathsf{fee}$, $A := \gamma.\mathsf{Alice}$, $B := \gamma.\mathsf{Bob}$ and $I := \gamma.\mathsf{Ingrid}$.

VC-V gives Alice a quality of service guarantee, i.e., her virtual channel cannot be offloaded during the complete *validity* period. In practice, as Ingrid does not have the capability of recovering her collateral before validity, she might charge a higher fee to Alice in the virtual channel creation process.

We now give a more detailed overview of both designs, focusing in particular on the transactions required to open a virtual channel in both cases, which constitutes their cornerstone.

3-party synchronization for VC-NV. In this approach (illustrated in Figure 3a) the funding transaction of the virtual channel TX_f takes as input both the ledger channel Alice-Ingrid $\mathsf{TX}_\mathsf{s}^\mathsf{A}$ and the ledger channel Ingrid-Bob $\mathsf{TX}_\mathsf{s}^\mathsf{B}$. Both ledger channels contribute a total of $2c + f$ coins so that c are later used to setup the virtual channel and the rest $c + f$ are used as collateral from Ingrid plus the fee to provide the service for Alice and Bob.

In the honest case, Ingrid can publish both $\mathsf{TX}_\mathsf{s}^\mathsf{A}$ and $\mathsf{TX}_\mathsf{s}^\mathsf{B}$ on the blockchain, thereby enabling TX_f to be published, eventually letting Ingrid get her collateral back from the output $c + f$ in TX_f, and Alice and Bob the coins reflected in the last state of the virtual channel, as enforced by the subsequent protocol operations. If Ingrid colludes with Alice (the case with Bob is symmetric) and decides not to publish $\mathsf{TX}_\mathsf{s}^\mathsf{A}$ as expected, Bob can get compensated by sending the $c + f/2$ coins from $\mathsf{TX}_\mathsf{s}^\mathsf{B}$ to himself and on his own after $T + 4\Delta$ rounds. This timeout is set so that an honest Ingrid can the funding transaction TX_f before.

3-party synchronization for VC-V. In this approach (illustrated in Figure 3b) the funding transaction of the virtual channel TX_f takes as input only $\mathsf{TX}_\mathsf{s}^\mathsf{A}$ while the refund of the collateral from Ingrid as encoded in $\mathsf{TX}_\mathsf{refund}$ depends on both $\mathsf{TX}_\mathsf{s}^\mathsf{B}$ and TX_f itself. Note that, as in the previous design, the virtual channel is funded with c coins while Ingrid is able to recover $c + f$ coins.

Intuitively, this design allows Alice to publish TX_f on her own which in turn lets Ingrid recover her collateral on her own as well. In a bit more detail, Alice can publish $\mathsf{TX}_\mathsf{s}^\mathsf{A}$ and TX_f, thereby letting Ingrid publish $\mathsf{TX}_\mathsf{s}^\mathsf{B}$ and $\mathsf{TX}_\mathsf{refund}$. This design also differs in the handling of the dishonest case. Assume that Alice is malicious and does not publish TX_f before the pre-agreed validity period (val) expires, then Ingrid and Bob can punish Alice and get their coins back. In particular, Ingrid can get the coins from $\mathsf{TX}_\mathsf{s}^\mathsf{A}$ after val. Similarly, Bob can take the coins from $\mathsf{TX}_\mathsf{s}^\mathsf{B}$ after val + 2Δ rounds. Similar to our design without validity, the timeout val + 2Δ gives Ingrid enough time to claim her refund if TX_f is published.

Arbitrary funds in ledger channels. In the high level overview given above, we assume that the ledger channels Alice-Ingrid and Ingrid-Bob have the exact amount of coins that are needed for the virtual channel Alice-Bob which is in practice quite unlikely. Let us stress that this assumption is made just to simplify the exposition as it can avoided using the channel splitting technique discussed in [4]. This means that before constructing the virtual channel Alice-Bob, parties would first *split* each underlying ledger channel off-chain in two channels: (i) one would contain the exact amount of coins for the virtual channel and (ii) the other one would contain the remaining coins.

3 VIRTUAL CHANNELS

3.1 Definitions and security model

Security model. We formally model the security of our virtual channel constructions using a synchronous version of the global UC framework (GUC) [8]. Financial transactions are handled by a blockchain which we model as a global ideal functionality $\widehat{\mathcal{L}}(\Delta, \Sigma)$. The parameter Δ upper bounds on the blockchain delay (number of rounds it takes to publish a transaction) and Σ defines the signature scheme used by the blockchain. We denote by $\mathcal{P}$ the set of all parties participating in the protocols considered in this work. For more details about our model, we refer the reader to Appendix B.

Channels. We briefly recall some notation and definition for generalized channels as presented in [4] and extend the definition to generalized virtual channels. In order to make the distinction between the two types of channels, we call the former generalized *ledger* channel (or ledger channels for short).

A *generalized ledger channel* is defined as a tuple $\gamma := (\gamma.\mathsf{id}, \gamma.\mathsf{Alice}, \gamma.\mathsf{Bob}, \gamma.\mathsf{cash}, \gamma.\mathsf{st})$, where $\gamma.\mathsf{id} \in \{0,1\}^*$ is the identifier of the channel, $\gamma.\mathsf{Alice}, \gamma.\mathsf{Bob} \in \mathcal{P}$ are the identities of the parties using the channel, $\gamma.\mathsf{cash} \in \mathbb{R}^{\geq 0}$ is a finite precision real number that represents the total amount of coins locked in this channel and $\gamma.\mathsf{st} = (\theta_1, \ldots, \theta_n)$ is the state of the channel. This state is composed of a list of *outputs*. Recall that each output θ_i has two attributes: the output value $\theta_i.\mathsf{cash} \in \mathbb{R}^{\geq 0}$ and the output condition $\theta_i.\varphi\colon \{0,1\}^* \times \mathbb{N} \times \mathbb{N} \to \{0,1\}$. For convenience, we define a set $\gamma.\mathsf{endUsers} := \{\gamma.\mathsf{Alice}, \gamma.\mathsf{Bob}\}$ and a function $\gamma.\mathsf{otherParty}\colon \gamma.\mathsf{endUsers} \to \gamma.\mathsf{endUsers}$, which on input $\gamma.\mathsf{Alice}$ outputs $:= \gamma.\mathsf{Bob}$ and on input $\gamma.\mathsf{Bob}$ returns $\gamma.\mathsf{Alice}$.

A generalized *virtual channel* (or just virtual channel for short) is defined as a tuple $\gamma := (\gamma.\mathsf{id}, \gamma.\mathsf{Alice}, \gamma.\mathsf{Bob}, \gamma.\mathsf{cash}, \gamma.\mathsf{st}, \gamma.\mathsf{Ingrid},$

γ.subchan, γ.fee, γ.val). The attributes γ.id, γ.Alice, γ.Bob, γ.cash, γ.st are defined as in the case of ledger channels. The additional attribute γ.Ingrid $\in \mathcal{P}$ denotes the identity of the *intermediary* of the virtual channel γ. The set γ.endUsers and the function γ.otherParty are defined as before. Additionally, we also define the set γ.users := $\{\gamma.\text{Alice}, \gamma.\text{Bob}, \gamma.\text{Ingrid}\}$. The attribute γ.subchan is a function mapping γ.endUsers to a channel identifier; namely, the value γ.subchan(γ.Alice) refers to the identifier of the channel between γ.Alice and γ.Ingrid; similarly, the value γ.subchan(γ.Bob) refers to the identifier of the channel between γ.Bob and γ.Ingrid. The value γ.fee $\in \mathbb{R}^{\geq 0}$ represents the fee charged by γ.Ingrid for her service of being an intermediary of γ. Finally, we introduce the attribute γ.val $\in \mathbb{N} \cup \{\perp\}$. If γ.val $\neq \perp$, then we call γ a *virtual channel with validity* and the value of γ.val represents the round number until which γ remains an open virtual channel. Channels with γ.val $= \perp$ are called *virtual channels without validity*.

Rooted transactions. UTXO based blockchains can be viewed as a directed acyclic graph, where each node represents a transaction. Nodes corresponding to transactions tx_i and tx_j are connected with an edge if at least one of the outputs of tx_i is an input of tx_j, i.e, tx_i is (partially) funding tx_j. We denote the transitive reachability relation between nodes, which constitutes a partial order, as $\leq$. We say that a transaction tx is *rooted* in the set of transactions R if (1) $\forall \text{tx}_i \leq \text{tx}.\exists \text{tx}_j \in R.\text{tx}_j \leq \text{tx}_i \vee \text{tx}_i \leq \text{tx}_j$, (2) $\forall \text{tx}_i, \text{tx}_j \in R.\text{tx}_i \neq \text{tx}_j, \text{tx}_i \not\leq \text{tx}_j$, (3) $\text{tx} \notin R$. Intuitively, this definition allows us to argue that certain set transactions fully funds the transaction tx. Clearly, the set of transactions defined by the tx.Input (if they are not connected) does satisfy such definition. However, there are many other sets that do so as well, e.g. the set of all inputs of all transaction define by tx.Input. An example is given in Appendix A.

3.2 Security and efficiency goals

We briefly recall the properties of generalized channels as defined by the ideal functionality in [4] and we informally state the additional properties that we require from virtual channels.

Security goals. Generalized ledger channels must satisfy three security properties, namely (S1) Consensus on creation, (S2) Consensus on update and (S3) Instant finality with punish. Intuitively, properties (S1) and (S2) guarantee that successful creation of a new channel as well as successful update of an existing channel happens if and only if both parties agree on the respective action. Property (S3) states that if a channel γ is successfully updated to the state γ.st and γ.st is the last state that the channel is updated to, then an honest party $P \in \gamma$.endUsers has the guarantee that either this state can be enforced on the ledger or P can enforce a state where she gets all the coins locked in the channel. By saying that the state st is *enforced* on the ledger we mean that a transaction with this state appears on the ledger.

Since virtual channels are generalized channels whose funding transaction is not posted on the ledger yet, the above stated properties should hold for virtual channels as well with a subtle but important difference: the creation of a virtual channel involves three parties (Alice, Ingrid and Bob) and hence consensus on creation for virtual channels can only be fulfilled if all three parties agree on the creation. In addition to the above properties, virtual channels should also satisfy the following properties:

(V1) Balance security If γ is a virtual channel and γ.Ingrid is honest, she never loses coins, even if γ.Alice and γ.Bob collude.

(V2) Offload with punish If γ is a virtual channel *without* validity (VC-NV), then γ.Ingrid is either able to transform γ to a ledger channel or she gets financially compensated.

(V3)Validity with punish In a virtual channel γ *with* validity (VC-V), γ.endUsers have the guarantee that γ remains a virtual channel until round γ.val or the honest parties get financially compensated. Additionally, γ.Ingrid has the guarantee that γ will be closed before γ.val or she gets financially compensated.

We first note that the instant finality with punish property (S3) does not provide any guarantees for Ingrid $\notin \gamma$.endUsers, and hence we need to define (V1) for virtual channels. In addition, properties (V2) and (V3) point out the main difference between VC-V and VC-NV. In a VC-NV γ, Ingrid is able to free her collateral from γ at any time by transforming the channel between Alice and Bob from a virtual channel to a ledger channel. Instead, in a VC-V γ, each honest party in γ.endUsers is guaranteed that either γ is closed at latest in round γ.val or the party gets financially compensated from the underlying ledger channels.

Efficiency goals. Lastly, we define the following efficiency goals:

(E1) Constant round creation Successful creation of a virtual channel takes constant number of rounds.

(E2) Optimistic update For a channel γ, this property guarantees that in the optimistic case when both parties in γ.endUsers are honest, a channel update takes constant number of rounds.

(E3) Optimistic closure In the optimistic case when all parties in γ.users are honest, the closure of a virtual channel takes constant number of rounds.

Let us stress that property (E2) is common for all off-chain channels (i.e. both ledger and virtual channels). The properties (E1) and (E3) capture the additional property of virtual channels that in the optimistic case when all parties behave honestly, the entire life-cycle of the channel is performed completely off-chain.

We compare the security and efficiency goals for different types of channels in Table 1.

3.3 Ideal functionality for virtual channels

We are now prepared to define the ideal functionality $\mathcal{F}_V$ that describes the ideal behavior of both ledger and virtual channels. Hence it can be viewed as an extension of the ledger channel functionality $\mathcal{F}_L$ defined in [4] (we recall this functionality in Appendix C for completeness). The functionality $\mathcal{F}_V$ is parameterized by a parameter T which upper bounds the maximum number of off-chain communication rounds between two parties required for any of the

	L-Security	V-Security			Efficiency		
	S1 – S3	V1	V2	V3	E1	E2	E3
L	✓	-	-	-	✗	✓	✗
VC-V	✓	✓	✗	✓	✓	✓	✓
VC-NV	✓	✓	✓	✗	✓	✓	✓

Table 1: Comparison of security and efficiency goals for ledger channels (L), virtual channels with validity (VC-V) and virtual channels without validity (VC-NV).

operations in $\mathcal{F}_L$. The ideal functionality $\mathcal{F}_V$ communicates with the parties $\mathcal{P}$, the simulator $\mathcal{S}$ and the ledger $\widehat{\mathcal{L}}$. It maintains a channel space Γ where it stores all currently opened ledger channels (together with their funding transaction tx) and virtual channels. Before we define $\mathcal{F}_V$ formally, we describe it on a high level.

Messages related to ledger channels. For any message related to a ledger channel, $\mathcal{F}_V$ behaves as the functionality $\mathcal{F}_L$. That is, the corresponding code of $\mathcal{F}_L$ is executed when a message about a ledger channel γ is received. For the rest of this section, we discuss the behavior of $\mathcal{F}_V$ upon receiving a message about a virtual channel.

Create. The creation of a virtual channel is equivalent to synchronously updating two ledger channels. Therefore, if all parties, namely γ.Alice, γ.Bob and γ.Ingrid, follow the protocol, i.e., update their ledger channels correctly, a virtual channel is successfully created. This is captured in the "All agreed" case of the functionality. Hence, if all parties send the CREATE message, the functionality returns CREATED to γ.users, keeps the underlying ledger channels locked and adds the virtual channel to its channel space Γ. We note that the simulator is responsible for informing the functionality whether corrupted parties behave honestly and follow the protocol.

On the other hand, the creation of the virtual channel fails if after some time at least one of the parties does not send CREATE to the functionality. There are three possible situations in which the creation can fail. First, the update is peacefully rejected and parties simply abort the virtual channel creation (e.g., γ.Ingrid rejects updating her channel upon receiving the update request). Second, both channels are forcefully closed, in order to prevent a situation where one of the channels is updated and the other one is not (e.g., the parties start the update procedure of their channels, but γ.Alice or γ.Bob aborts the procedure prematurely). Finally the functionality waits Δ rounds and checks if γ.Ingrid has published the old state of one of her channels to the ledger. If not it forcefully closes the ledger channels using the new state. This models the case where γ.Ingrid behaves maliciously and causes a situation where she can publish both the old and new states of the ledger channels, while γ.Alice or γ.Bob can only publish the new state (e.g., in case γ.Alice revokes her previous state but γ.Ingrid does not).

The property (S1) is guaranteed because the parties in γ.endUsers only output CREATED if message CREATE was received from all parties, and property (E1) holds since the functionality waits at most a constant number of rounds for all the CREATED messages.

Update. Since a virtual channel can be seen as a ledger channel where the funding transaction is not published on the ledger, the update procedure for the virtual channel works in the same way as for ledger channels. Hence, also the functionality code in this operation is the same as $\mathcal{F}_L$, with the only difference that in case of any disputes during the execution, the functionality calls V–ForceClose instead of L–ForceClose (therefore the properties (S2), (E2) for virtual channels follow directly from (S2), (E2) of the ledger channels functionality).

Offload. The offloading mechanism allows the parties to transform a virtual channel into a ledger channel. Upon completion, I's collateral would be unlocked and she will receive a fee for facilitating the virtual channel. In addition the remaining funds in the underlying ledger channels will finance the funding transaction of virtual channel on the ledger. We consider two types of offloading depending on whether the virtual channel is with or without validity. In the first case, offloading is initiated by one of the γ.endUsers before round γ.val, while for channels without validity, Ingrid can initiate the offloading at any time. We note that for channels without validity if both parties in γ.endUsers cooperate, they also can offload the virtual channel, while one party alone does not have the power to do so. Yet in practice, if both parties cooperate and behave honestly, there is no situation in which these parties would have to offload. Since offloading a virtual channel requires closure of the underlying subchannels, the functionality merely checks if either funding transaction of γ.subchan has been spent until round $T_1 + \Delta$. If not, the functionality outputs a message (ERROR). Similarly to [4], the ERROR message represents an impossible situation which should not happen as long as one of the parties are honest.

Close - channels without validity. Similar to the Create procedure, closing a virtual channel corresponds to a synchronous update of two ledger channels. Upon receiving (CLOSE, *id*) from all parties in γ.users within $T_1 \leq 6T$ rounds (where the exact value of T_1 is specified by $\mathcal{S}$), all parties have peacefully agreed on closing the virtual channel, which is indicated by the "All Agreed" case. Suppose γ was in state $\gamma.\mathsf{st} = \{(c_A, \mathtt{One\text{–}Sig}_A), (c_B, \mathtt{One\text{–}Sig}_B)\}$, meaning that Alice $:= \gamma$.Alice has c_A coins and Bob $:= \gamma$.Bob has c_B coins. In this case the functionality updates the underlying sub-channels $\alpha = \gamma$.subchan(Alice) by assigning c_A coins to Alice and $c_B + \gamma.\mathsf{fee}/2$ to γ.Ingrid. Similarly in the sub-channel $\beta = \gamma$.subchan(Bob) party Bob gets assigned c_B coins, while γ.Ingrid receives $c_A + \gamma.\mathsf{fee}/2$. When the update of Γ is completed, the ideal functionality sends CLOSED to all users. Due to the peaceful closure in this "All Agreed" case, the functionality defines property (E3).

We put a restriction on the state of the virtual channel in which peaceful closure is possible. Namely, we assume that peaceful closure is only possible for states that merely assign the coins in γ between γ.Alice and γ.Bob. That is, we do not allow any peaceful closure while the state of γ still contains more complex objects like conditional payments or another virtual channel.

If one of the messages (CLOSE, *id*) was not received within T_1 rounds ("Wait for others" case), then the closing procedure failed and the functionality distinguishes the following two cases. In the first case, the update procedure of an underlying ledger channel was aborted prematurely by γ.Alice or γ.Bob. In that case the virtual channel is forcefully closed. In the second case, γ.Ingrid behaves maliciously and refuses to revoke her state during the update of either one of the underlying ledger channels. Like in the virtual channel creation, the functionality waits Δ rounds and checks if γ.Ingrid has published the old state to the ledger. If not it executes forceful closure of the ledger channels using the new state.

Close - channels with validity. The close procedure of a virtual channel with validity starts in round $\gamma.\mathsf{val} - (4\Delta + 7T)$ such that there is enough time to forcefully close the channel if necessary. If within $T_1 \leq 6T$ rounds (where the exact value of T_1 is specified by $\mathcal{S}$) all γ.users agreed on closing the channel or if the simulator instructs the functionality to close the channel, the same steps as in the all agreed case for channels without validity are executed.

Otherwise, after T_1 rounds, the functionality executes the forceful closure of the virtual channel.

Punish. The punishment procedure is executed at the end of each round. It checks for every virtual channel γ if any of γ.subchan has just been closed and distinguishes if the consequence of closure was offloading or punishment. If after T_1 rounds (where T_1 is set by $\mathcal{S}$) two transactions tx_1 and tx_2 are published on the ledger, where tx_1 refunds the collateral γ.cash + γ.fee to γ.Ingrid and tx_2 funds γ on-chain, then the virtual channel has been offloaded. Hence, the functionality sends a message (OFFLOADED) to γ.users.

If after T_1 rounds, the ledger contains only one transaction tx, which assigns γ.cash coins to a single honest party P and which spends the funding transaction of only one of γ.subchan, the functionality sends (PUNISHED) to P.

In all other cases, the functionality outputs (ERROR) to γ.users.

Force close. Since a virtual channel is not funded on the ledger, forcing the closure of a virtual channel consists of two steps. First, the functionality executes the subprocedure Offload, in order to make sure that the virtual channel is funded on-chain. Once this execution finishes successfully, the virtual channel is a funded ledger channel and hence, the functionality can execute the subprocedure L−ForceClose for ledger channels.

We note that (S3) holds because γ.endUsers can either close their virtual channel or punish the misbehaving party. For channels without validity, I can either offload or punish the misbehaving users. For channels with validity if the channel gets closed before validity, the parties can either post the latest state of the channel or punish the misbehaving party as in ledger channels. After validity the responsible party in γ.endUsers is punished for not closing the channel. Hence (V1)-(V3) hold as well.

Notation and assumptions. We do not aim to make any claims regarding the privacy of our protocols and hence we (implicitly) assume that all messages that are received/sent to/from the ideal functionalities are directly forwarded to $\mathcal{S}$. However virtual channels improve privacy compared to the lightning network since each single payment is not routed through Ingrid. The formal description of $\mathcal{F}_V$ is simplified by excluding several natural checks which are listed (as a functionality wrapper) in Appendix E. We write $m \overset{t}{\hookrightarrow} P$ as a short hand form for "send the message m to party P in round t." and $m \overset{t}{\hookleftarrow} P$ for "receive a message m from party P in round t".

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

Below we abbreviate $A := \gamma$.Alice, $B := \gamma$.Bob, $I = \gamma$.Ingrid. For $P \in \gamma$.endUsers, we denote $Q := \gamma$.otherParty(P).

Upon receiving a message about ledger channels, behave as $\mathcal{F}_L(T, 1)$.

Create

Upon (CREATE, γ) $\overset{\tau}{\hookleftarrow} P$, let $\mathcal{S}$ define $T_1 \leq 8T$. If $P \in \gamma$.endUsers, then define a set S, where $S := \{id_P\} := \gamma$.subchan(P), otherwise define S as $S := \{id_P, id_Q\} := \gamma$.subchan. Lock all channels in S and distinguish the following cases:

All agreed: If you already received both (CREATE, γ) $\overset{\tau_1}{\hookleftarrow} Q_1$ and (CREATE, γ) $\overset{\tau_2}{\hookleftarrow} Q_2$, where $Q_1, Q_2 \in \gamma$.users $\setminus \{P\}$ and $\tau - T_1 \leq \tau_1 \leq \tau_2$, then in round $\tau_3 := \tau_1 + T_1$ proceed as follows:

(1) Let $\mathcal{S}$ define $\vec{\theta}_A$ and $\vec{\theta}_B$ and set $(id_A, id_B) := \gamma$.subchan.
(2) Execute UpdateState($id_A, \vec{\theta}_A$) and UpdateState($id_B, \vec{\theta}_B$), set $\Gamma(\gamma.\mathrm{id}) := \gamma$, send (CREATED, γ) $\overset{\tau_3}{\hookrightarrow} \gamma$.endUsers and stop.

Wait for others: Else wait for at most T_1 rounds to receive (CREATE, γ) $\overset{\tau_1 \leq \tau + T_1}{\longleftarrow} Q_1$ and (CREATE, γ) $\overset{\tau_2 \leq \tau + T_1}{\longleftarrow} Q_2$ where $Q_1, Q_2 \in \gamma$.users $\setminus \{P\}$ (in that case option "All agreed" is executed). If at least one of those messages does not arrive before round $\tau + T_1$, do the following. For all $id_i \in S$, let $(\gamma_i, \mathsf{tx}_i) := \Gamma(id_i)$ and distinguish the following cases:

- If $\mathcal{S}$ sends (peaceful−reject, id_i), unlock id_i and stop.
- If γ.Ingrid is honest or if instructed by $\mathcal{S}$, execute the subprocedure L−ForceClose(id_i) and stop.
- Otherwise wait for Δ rounds. If tx_i still unspent, then set $\vec{\theta}_{old} := \gamma_i$.st, γ_i.st $:= \{\vec{\theta}_{old}, \vec{\theta}\}$ and $\Gamma(id_i) := (\gamma_i, \mathsf{tx}_i)$. Execute the subprocedure L−ForceClose(id_i) and stop.

Update

Upon (UPDATE, id, $\vec{\theta}$, t_{stp}) $\overset{\tau_0}{\hookleftarrow} P$, where $P \in \gamma$.endUsers, behave as $\mathcal{F}_L(T, 1)$ yet replace the calls to L−ForceClose in $\mathcal{F}_L(T, 1)$ with calls to V−ForceClose.

Offload

Upon (OFFLOAD, id) $\overset{\tau_0}{\hookleftarrow} P$, execute the subprocedure Offload(id).

Close

Channels without validity:

Upon (CLOSE, id) $\overset{\tau}{\hookleftarrow} P$, where $\gamma(id)$.val $= \bot$, let $\mathcal{S}$ define $T_1 \leq 6T$. If $P \in \gamma_i$.endUsers, define a set S, where $S := \{id_P\} := \gamma_i$.subchan($P$), else define S as $S := \{id_P, id_Q\} := \gamma_i$.subchan and distinguish:

All agreed: If you received both messages (CLOSE, id) $\overset{\tau_1}{\hookleftarrow} Q_1$ and (CLOSE, id) $\overset{\tau_2}{\hookleftarrow} Q_2$, where $Q_1, Q_2 \in \gamma$.users $\setminus \{P\}$ and $\tau - T_1 \leq \tau_1 \leq \tau_2$, then in round $\tau_3 := \tau_1 + T_1$ proceed as follows:

(1) Let $\gamma := \Gamma(id)$, $(id_A, id_B) := \gamma$.subchan.
(2) Parse γ.st $= \{(c_A, \mathsf{One\text{−}Sig}_A), (c_B, \mathsf{One\text{−}Sig}_B)\}$ and define

$$\vec{\theta}_A := ((c_A, \mathsf{One\text{−}Sig}_A), (c_B + \gamma.\mathrm{fee}/2, \mathsf{One\text{−}Sig}_I)),$$

$$\vec{\theta}_B := ((c_A + \gamma.\mathrm{fee}/2, \mathsf{One\text{−}Sig}_I), (c_B, \mathsf{One\text{−}Sig}_B)),$$

(3) Unlock both subchannels and execute UpdateState($id_A, \vec{\theta}_A$) and UpdateState($id_B, \vec{\theta}_B$). Set $\Gamma(id) := \bot$ and send (CLOSED, γ) $\overset{\tau_3}{\hookrightarrow} \gamma$.endUsers.

Wait for others: Else wait for at most T_1 rounds to receive (CLOSE, γ) $\overset{\tau_1 \leq \tau + T_1}{\longleftarrow} Q_1$ and (CLOSE, γ) $\overset{\tau_2 \leq \tau + T_1}{\longleftarrow} Q_2$ where $Q_1, Q_2 \in \gamma$.users $\setminus \{P\}$ (in that case option "All agreed" is executed). For all $id_i \in S$ let $(\gamma_i, \mathsf{tx}_i) := \Gamma(id_i)$, if such messages are not received until round $\tau + T_1$, set $\vec{\theta}_{old} := \gamma'$.st and distinguish:

- If γ.Ingrid is honest or if instructed by $\mathcal{S}$, execute the subprocedure V−ForceClose(id_i) and stop.
- Else wait for Δ rounds. If tx_i still unspent, set γ_i.st $:= \{\vec{\theta}_{old}, \vec{\theta}\}$ and $\Gamma(id_i) := (\gamma_i, \mathsf{tx}_i)$. Execute L−ForceClose($id_i$) and stop.

Channels with validity:

For every $\gamma \in \Gamma$ s.t. γ.val $\neq \bot$, in round $\tau_0 := \gamma$.val $- (4\Delta + 7T)$ proceed as follows: let $\mathcal{S}$ set $T_1 \leq 6T$ and distinguish:

Peaceful close: If all parties in γ.users are honest or if instructed by $\mathcal{S}$, execute steps (1)–(3) of the "All agreed" case for channels without validity with $\tau_3 := \tau_0 + T_1$.

Force close: Else in round τ_3 execute V–ForceClose(γ.id).

Punishment (executed at the end of every round)

For every id, where $\gamma := \Gamma(id)$ is a virtual channel, set $(id_A, id_B) := \gamma$.subchan. If this is the first round when $\Gamma(id_A) = (\bot, \mathsf{tx}_A)$ or $\Gamma(id_B) = (\bot, \mathsf{tx}_B)$, i.e., one of the subchannels was just closed, then let $\mathcal{S}$ set $t_1 \leq T'$, where $T' := \tau_0 + T + 5\Delta$ if γ.val $= \bot$ and $T' := \gamma$.val $+ 3\Delta$ if γ.val $\neq \bot$, and distinguish the following cases:

Offloaded: Latest in round t_1 the ledger $\widehat{\mathcal{L}}$ contains both

- a transaction tx_1 rooted at $\{\mathsf{tx}_A, \mathsf{tx}_B\}$ with an output (γ.cash + γ.fee, One–Sig$_I$). In this case (OFFLOADED, id) $\overset{\tau_1}{\hookrightarrow} I$, where τ_1 is the round tx_1 appeared on $\widehat{\mathcal{L}}$.
- a transaction tx_2 with an output of value γ.cash and rooted at $\{\mathsf{tx}_A, \mathsf{tx}_B\}$, if γ.val $= \bot$, and rooted at $\{\mathsf{tx}_A\}$, if γ.val $\neq \bot$. Let τ_2 be the round when tx_2 appeared on $\widehat{\mathcal{L}}$. Then output (OFFLOADED, id) $\overset{\tau_2}{\hookrightarrow}$ γ.endUsers, set $\gamma' = \gamma$, γ'.Ingrid $= \bot$, γ'.subchan $= \bot$, γ.val $= \bot$ and define $\Gamma(id) := (\gamma', \mathsf{tx}_2)$.

Punished: Else for every honest party $P \in \gamma$.users, check the following: the ledger $\widehat{\mathcal{L}}$ contains in round $\tau_1 \leq t_1$ a transaction tx rooted at either tx_A or tx_B with (γ.cash + γ.fee/2, One–Sig$_P$) as output. In that case, output (PUNISHED, id) $\overset{\tau_1}{\hookrightarrow} P$. Set $\Gamma(id) = \bot$ in the first round when PUNISHED was sent to all honest parties.

Error: If the above case is not true, then (ERROR) $\overset{t_1}{\hookrightarrow}$ γ.users.

V–ForceClose(id): Let τ_0 be the current round and $\gamma := \Gamma(id)$. Execute subprocedure Offload(id). Let $T' := \tau_0 + 2T + 8\Delta$ if γ.val $= \bot$ and $T' := \gamma$.val $+ 3\Delta$ if γ.val $\neq \bot$. If in round $\tau_1 \leq T'$ it holds that $\Gamma(id) = (\gamma, \mathsf{tx})$, execute subprocedure L–ForceClose(id).

Subprocedure Offload(id): Let τ_0 be the current round, $\gamma := \Gamma(id)$, $(id_\alpha, id_\beta) := \gamma$.subchan, $(\alpha, \mathsf{tx}_A) := \Gamma(id_\alpha)$ and $(\beta, \mathsf{tx}_B) := \Gamma(id_\beta)$. If within Δ rounds, neither tx_A nor tx_B is spent, then output (ERROR) $\overset{\tau_0+\Delta}{\hookrightarrow}$ γ.users.

Subprocedure UpdateState($id, \vec{\theta}$): Let $(\alpha, \mathsf{tx}) := \Gamma(id)$. Set α.st $:= \vec{\theta}$ and update $\Gamma(id) := (\alpha, \mathsf{tx})$.

4 VIRTUAL CHANNEL PROTOCOL

In this section, we present the rationale of the design of our protocol realizing the ideal functionality $\mathcal{F}_V(T)$. We aim at a modular approach that builds upon the protocol for ledger channels as described in [4], leveraging them in a black-box manner. Technically, we design a protocol $\Pi_V(T)$ that works in the $\mathcal{F}_L(T, 1)$-hybrid world where T is defined in Section 3.3 and 1 is the number of commit transactions used in the ledger channel (see Appendix B for the definition of a hybrid world). In this section, we firstly elaborate on our modular approach, explaining its main technical challenges, and thereafter describe our virtual channel protocol. To simplify the exposition, we use the following abbreviations in the rest of this section: $\mathcal{F}_V := \mathcal{F}_V(T)$, $\mathcal{F}_L := \mathcal{F}_L(T, 1)$ and $\Pi_V := \Pi_V(T)$.

4.1 Modular approach

Parties in the protocol Π_V make use of the hybrid ideal functionality $\mathcal{F}_L$ in the following cases.

(1) When a party receives a message about a ledger channel from the environment, it simply forwards this message to the hybrid ideal functionality $\mathcal{F}_L$, waits for its reply and forwards this reply to the environment. Hence, the party acts as a so-called *dummy party*.
(2) During virtual channel creation and closure, parties instruct the hybrid ideal functionality $\mathcal{F}_L$ to *update* both subchannels of the virtual channel.

There is one technical problem with the above modular design coming from the fact that we allow virtual channels to be *offloaded*. Recall that offloading is a transformation that turns a virtual channel into a ledger channel which, effectively, means that its funding transaction appears on-chain. Assume that channel γ was created as a virtual channel and offloaded later. Now γ is a ledger channel and hence falls under the rule (1) described above. Namely, if the environment asks a user of this channel to update the state of the channel, this party simply forwards the update instruction to the hybrid ideal functionality $\mathcal{F}_L$. However, the hybrid functionality does not "know" this ledger channel, that is, it does not have it in its channels space since the channel was not created via $\mathcal{F}_L$.

To overcome this technical modeling problem, we extend the the functionality $\mathcal{F}_L$ such that it supports a new operation: *prepare* a ledger channel. In other words, it allows parties to create and maintain a ledger channel "in their head" for some time and only later turn it into a full-fledged ledger channel by publishing its funding transaction on-chain. In a bit more detail, we define $\mathcal{F}_{preL}$, a functionality that behaves exactly as $\mathcal{F}_L$ but is extended with the following additional features.(i) creating a ledger channel whose funding transaction does not have to be published on-chain (although its identifier must be known and registered in the channel space at creation time); (ii) updating such channel and (iii) constantly monitoring the ledger so that once the funding transaction appears on-chain, the channel in preparation becomes a full-fledged ledger channel. In order to distinguish this additional features from the standard ones, we use the prefix *pre-* for the channels in preparation, e.g., *pre-create, pre-update*.

Let us stress that until the channel becomes a full-fledged ledger channel, no guarantees are provided to the channel users. However, once the funding transaction is published, all security and efficiency properties of ledger channels (as defined in [4]) apply. Due to space restriction, we refer the reader for the formal functionality description to Appendix C, where we also show how to realize this new ideal functionality.

In conclusion, we design a virtual channel protocol Π_V in the $\mathcal{F}_{preL}$-hybrid world where parties interact with $\mathcal{F}_{preL}$ in the cases (1) and (2) as described previously in this section and additionally:

(3) During virtual channel creation, parties *pre-create* a ledger channel via $\mathcal{F}_{preL}$.
(4) During virtual channel update, parties *pre-update* their previously prepared ledger channel via $\mathcal{F}_{preL}$.

These two additional cases guarantee that once the virtual channel is offloaded, the hybrid functionality knows about its existence and, importantly, also about its latest agreed-on state.

4.2 High level protocol description

We now present a high-level description of our virtual channel protocol Π_V. Since our goal is that Π_V UC-realizes the functionality $\mathcal{F}_V$, we need to discuss how parties handle messages about both ledger channels and virtual channel with and without validity. As discussed in the overview of our modular approach (Section 4.1), we design Π_V in the hybrid world of the ledger channel functionality $\mathcal{F}_{preL}$ which allows parties in the protocol to forward all messages about ledger channels to $\mathcal{F}_{preL}$ (recall the case (1) above). In the rest of this section we focus on the more interesting case, that is, how parties deal with messages about virtual channels. To this end, we discuss the following subprotocols: create, update, close, offload, and punish. The formal description of our virtual channel protocol can be found in Appendix D.

Create. Let γ be a virtual channel that $A := \gamma.\text{Alice}$ and $B := \gamma.\text{Bob}$ want to create, using their ledger channels with $I := \gamma.\text{Ingrid}$. On a high level, the creation procedure of a virtual channel is a synchronous update of the underlying ledger channels. Following the guidelines on how to use the ledger channel functionality $\mathcal{F}_{preL}$ given in [4, Sec. 6], we proceed as follows (see Figure 4).

As a first step, each party $P \in \{A, B\}$ initiates an update of the respective ledger channel with I (step ①), who, upon receiving both update requests, checks if the requested states (i.e., θ_A and θ_B) are consistent. The ideal functionality $\mathcal{F}_{preL}$ informs parties about the output identifiers tid_A and tid_B that can be used to build up the virtual channel (step ②). Next, all three parties engage in a setup phase, in which the structure of the virtual channel is being built (step ③). More concretely, all three parties agree on a funding transaction of the virtual channel and the end-users of the virtual channel, A and B, *pre-create* the channel via $\mathcal{F}_{preL}$. When the setup phase is completed, i.e., the virtual channel structure has been built, the parties complete the ledger channel update procedures (step ④). It is very important for the intermediary I to have the role of a reacting party during both channel updates. This gives her the power to wait until she is sure that both updates will complete successfully and only then give her final update agreement (step ⑤). Finally, the functionality informs all parties that updates were successfully completed (step ⑥), which implies that the virtual channel γ was successfully created. Let us stress that this description is simplified since it excludes the revocation steps. We refer the reader to the formal protocol description for details on that.

It remains to discuss how parties generate the new channel states θ_A and θ_B in step ①, and how parties setup the virtual channel construction in step ③. Since this differs for channels with and without validity, we discuss each channel type separately.

Without validity (Figure 5): First, I needs to inform both A and B about the last state of the ledger channels that are used as input for $\mathsf{TX_f}$. Second, upon creating $\mathsf{TX_f}$, A and B can then pre-create the rest of transactions required for the virtual channel γ. Finally, the three parties exchange the signatures on the funding transaction $\mathsf{TX_f}$ before communicating the update of both ledger channels to the functionality $\mathcal{F}_{preL}$.

With validity (Figure 6): Here, A can create $\mathsf{TX_f}$ on her own from the last state with her ledger channel with I. As second step, A and B can already create the transactions required for the virtual channel γ. Additionally, I and B create the refund transaction where I is refunded if the virtual channel is used. The last two steps are used to sign, from the right to left, the transactions created in previous steps. In particular, in step 4, B signs $\mathsf{TX}_{\mathsf{refund}}$ so that I is sure that she can publish it. Finally, I signs $\mathsf{TX_f}$ and provides the signature to A, effectively authorizing her to publish $\mathsf{TX_f}$ when A and B are done with using the virtual channel γ.

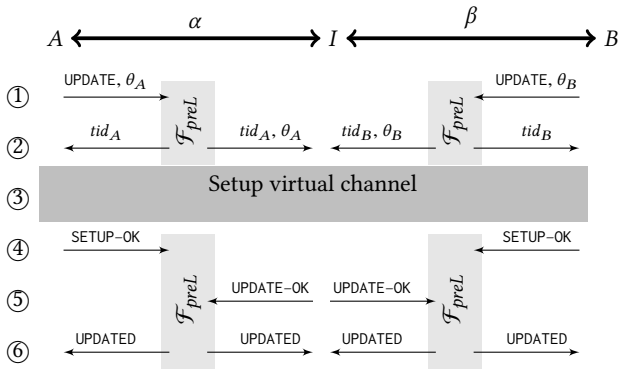


Figure 4: Simplified creation procedure of a virtual channel on top of two ledger channels α and β.

Update. In order to update the state of a virtual channel, one of the end-users, which we call the initiating party, is instructed by the environment to initiate the update. Recall that during virtual channel creation, the end-users A and B pre-created a channel via $\mathcal{F}_{preL}$. Hence, an update of the virtual channel is essentially just a pre-update of the prepared channel. To this end, the parties forward the update instructions from the environment to the functionality and vice versa. As long as the update is successful or peacefully rejected (meaning that the reacting party rejects the update), the parties act as dummy parties. The situation is more delicate when the pre-update fails because one of the parties misbehaved and aborted the pre-update.

We note that aborts during a channel update might cause a problematic asymmetry between the parties, for instance, when one party already signed the new state of the channel while the other one did not; or when one party already revoked the old state of

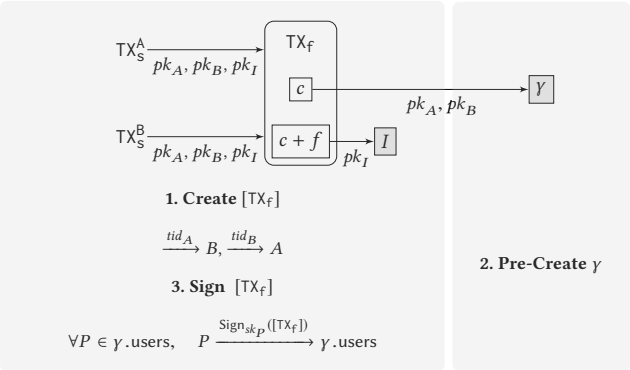


Figure 5: Setup of virtual channel without validity.

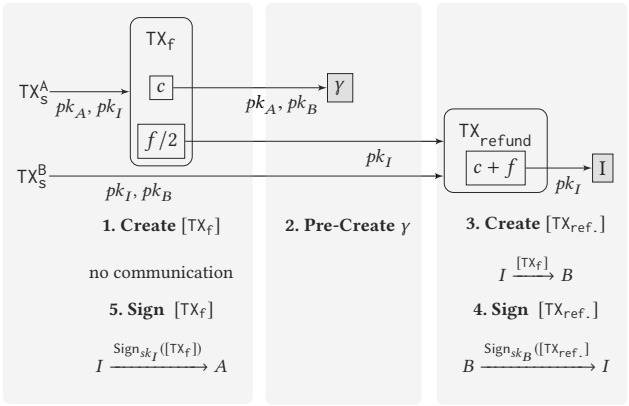


Figure 6: Setup of virtual channel with validity.

the channel but the other one did not. In a standard ledger channel, these disputes are resolved by a force close procedure. This means that the honest party publishes the latest valid state it is aware of on the blockchain. Hence, within a finite number of rounds, the dispute is resolved and the instant finality property is preserved. We apply a similar technique for virtual channels. The main difference is that a virtual channel is not funded on-chain. Hence, we first need to offload the virtual channel. In other words, we first need to transform a virtual channel into a ledger channel by publishing its funding transaction on-chain. Thereafter, the dispute is handled exactly as for ledger channels.

We note that virtual channels with validity should not be updated after round $\gamma.\text{val} - (4\Delta + 7T)$. This is simply because before round $\gamma.\text{val}$ the virtual channel must be closed and parties require $4\Delta + 7T$ rounds for the closing procedure in the worst case (i.e. when one of the parties aborts).

Close. The closure of a virtual channel is done by updating the ledger channels $\gamma.\text{subchan}(A)$ and $\gamma.\text{subchan}(B)$ according to the latest state of the virtual channel $\gamma.\text{st}$. Each party $P \in \{A, B\}$ computes the new state for the ledger channel and creates a new state $\vec{\theta}_P := \{(c_P, \texttt{One-Sig}_{pk_P}), (\gamma.\text{cash} - c_P, \texttt{One-Sig}_{pk_I})\}$ where c_P is the latest balance of P in γ. All parties update their ledger channels according to this state using $\mathcal{F}_{preL}$. Since both ledger channels must be updated synchronously, I waits for both parties to initiate the update procedure. In the end, after receiving the revocation of the previous state, the virtual channel is closed.

In the pessimistic case, parties must forcefully close their virtual channel, by publishing the funding transaction of the virtual channel (offloading) and then closing the resulting ledger channel.

Offload. During the offload procedure, parties try to publish the funding transaction of the virtual channel γ which transforms the virtual channel into a ledger channel. In a nutshell, during this procedure parties use the close interface of the $\mathcal{F}_{preL}$ in order to publish the commit and split transaction of both underlying ledger channels and afterwards the funding transaction of the virtual channel is published. Naturally, after all these transactions are published, I must be able to spend the collateral that she locked in both channels (as stated in property property (V1)).

Without validity In our virtual channel without validity, I is always able to offload γ which guarantees that I's collateral is not locked for unlimited amount of time. We note that $P \in \{A, B\}$ can also initiate the offloading by publishing the commit and split transaction of their respective ledger channels. This forces I to publish the commit and split transaction of the other ledger channel, since I loses her collateral to P otherwise.

With validity In our virtual channel with validity, only A can offload the virtual channel by publishing the commit and split transaction of her ledger channel with I. Although I and B are not able to offload the virtual channel, they have the guarantee that after round $\gamma.\text{val}$, either the channel is offloaded or closed, since otherwise they can punish A and get reimbursed. In order to prevent I from simply closing her channel with B and earning her collateral back, I is reimbursed only after the split transaction of her channel with A is posted. Afterwards I can post a transaction which uses an output of this split transaction and the split transaction from her ledger channel with B which repays her the collateral. Hence, this construction heavily relies on the fee which is being paid to I.

Punish. Misbehaving parties can be punished if offloading fails. The concept of punishment in virtual channels is similar to ledger channels, namely in case the latest state of the virtual channel cannot be posted on the ledger, honest A or B are compensated by receiving all coins of the virtual channel while honest I will not lose coins. If the funding transaction of the virtual channel is posted on the ledger, parties can execute the punishment protocol for their newly established ledger channel. Hence, in addition to the ledger channel's punishment, parties can punish if the funding transaction of γ cannot be published, in the following way:

Without validity Party $P \in \{A, B\}$ can punish I by taking all the coins on their respective ledger channels if the funding transaction of the virtual channel γ is not published on the ledger after $T + 4\Delta$ rounds. In other words, it is I's responsibility to ensure that the state of her ledger channels with A and B are not updated while γ is open.

With validity Here, only A can post the funding transaction of the virtual channel. Hence, if the virtual channel is not closed or offloaded by $\gamma.\text{val}$, A is punished. We note that A loses c_A coins to I (where c_A is the initial balance of A in γ) and I loses c_A coins to B (note that we are omitting I's fees to be concise).

Security analysis Due to the lack of space, here we only mention the main security theorem and refer to Appendix H for the proof.

Theorem 1. *Let Σ be a signature scheme that is existentially unforgeable against chosen message attacks. Then for any ledger delay $\Delta \in \mathbb{N}$, the protocol Π_V working in $\mathcal{F}_{preL}(3, 1)$-hybrid, UC-realizes the ideal functionality $\mathcal{F}_V(2)$.*

5 PERFORMANCE EVALUATION

In this section, we first study the storage overhead on the blockchain as well as the communication overhead between users to use virtual channels. For each of these aspects, we evaluate both constructions (i.e., with and without validity) and compare them. As testbed [2], the transactions are created in Python using the library `python-bitcoin-utils` and the Bitcoin *Script* language. To showcase compatibility and feasibility, we deployed these transactions successfully on the Bitcoin testnet. Later in this section, we evaluate

the advantages of virtual channels over ledger channels in terms of routing communication overhead and fee costs.

Communication overhead. We analyze the communication overhead imposed by the different operations, such as CREATE, UPDATE, OFFLOAD and CLOSE, by measuring the byte size of the transactions that need to be exchanged as well as the cost in USD necessary for posting the transactions that need to be published on-chain. The cost in USD is calculated by taking the price of 6853.05 USD per Bitcoin, the average transaction fee of 16 satoshis per byte and the size of the transactions that need to be posted on-chain, all of them at the time of writing. We detail in Table 2 the aforementioned costs measured for both virtual channel constructions.

As expected from the design itself of both constructions, they are similar for all operations except for the CREATE operation, where the VC-NV case requires one transaction less than the VC-V case. As illustrated in Section 2, this difference stems from the fact that we require only the $\mathsf{TX_f}$ in the VC-NV case and two transactions in the VC-V case, namely $\mathsf{TX_f}$ and $\mathsf{TX_{refund}}$. This very same difference of one transaction carries over to the cases of OFFLOAD and the pessimistic CLOSE operations.

In a bit more detail, for the creation of a virtual channel (CREATE operation), we need to update both ledger channels to a new state that can fund the virtual channel, requiring to exchange $2 \cdot 2$ transactions with 1494 (VC-NV) or 1422 (VC-V) bytes. Additionally, we need $\mathsf{TX_f}$ (640 bytes) or $\mathsf{TX_f}$ (309 bytes) and $\mathsf{TX_{refund}}$ (377 bytes) respectively and finally, we need a commitment (431 bytes) and a split transaction (264 bytes) for the virtual channel itself, spending from $\mathsf{TX_f}$. This complete process results in 7 (VC-NV) or 8 (VC-V) transactions with a total of 2829 (VC-NV) or 2803 (VC-V) bytes. Forcefully closing (CLOSE(pess) operation) and offloading (OFFLOAD operation) requires the same set of transactions as with CREATE, minus the commitment and the split transaction (695 bytes) of the virtual channel in the latter case. Finally, we observe that the UPDATE and the optimistic CLOSE(opt) operation require 2 transactions (695 bytes) for both constructions, as they are designed as an update of a ledger channel.

In the cases OFFLOAD and CLOSE(pess), the transactions mentioned above have to be published on-chain for both constructions. Most importantly however, in the CREATE and the optimistic CLOSE case we do not have any on-chain transactions at all, which is possibly one of the most relevant differences to ledger channels in practice, as it implies no on-chain fees for opening and closing virtual channels. For the operation UPDATE, we have also zero on-chain transactions, just as in ledger channels.

Operations	VC-NV					VC-V				
	on-chain			off-chain		on-chain			off-chain	
	# txs	size	cost	# txs	size	# txs	size	cost	# txs	size
CREATE	0	0	0	7	2829	0	0	0	8	2803
UPDATE	0	0	0	2	695	0	0	0	2	695
OFFLOAD	5	2134	2.34	0	0	6	2108	2.31	0	0
CLOSE (opt)	0	0	0	4	1390	0	0	0	4	1390
CLOSE(pess)	7	2829	3.10	0	0	8	2803	3.07	0	0

Table 2: Evaluation of the virtual channels. For each operation we show: the number of on-chain and off-chain transactions (*# txs*) and their *size* in bytes. For on-chain transactions, *cost* is in USD and denotes the estimated cost of publish them on the ledger.

5.1 Comparison to payment channel networks

In this section we compare virtual channels to multi-hop payments in a payment channel network (PCN). In a PCN, users route their payments via intermediaries. During the routing of a transaction tx, each intermediary party locks tx.cash coins as a "promise to pay" in their channels, a payment commitment that can technically be implemented as a Hash-Time Lock Contract (HTLC), e.g. as in the Lightning Network [21]. We now evaluate the difference in communication overhead and fee costs compared to virtual channels.

Routing communication overhead. When performing a payment between Alice and Bob via an intermediary Ingrid in a multi-hop payment, the participants need to update both generalized channels with a "promise to pay", which require 2 transactions or 818 bytes per channel when implemented as HTLC. If they are successful, both generalized channels need to be updated again to "confirm the payment" (again, 2 transactions or 695 bytes per channel). This whole process results in 8 transactions or $2 \cdot 818 + 2 \cdot 695 = 3026$ off-chain bytes that need to be exchanged. Generically, if the parties want to perform n sequential payments, they need to exchange $8 \cdot n$ transaction with a total of $3026 \cdot n$ bytes.

Assume now that Alice and Bob were to perform the payment over a virtual channel without validity instead and that this virtual channel is not yet created. As shown in Table 2, they need to open the virtual channel for 2829 bytes, where they set the balance of the virtual channel already to the correct state after the payment, and then close it again for 1390 bytes, resulting in a total of 4219 off-chain bytes. However, if we again consider n sequential payments, the result would be $11 + 2 \cdot (n-1)$ transactions or $4219 + 695 \cdot (n-1)$ bytes, which supposes a reduction of $2331 \cdot (n-1) - 1193$ bytes with respect to relying on generalized channels only. We obtain similar results if we consider virtual channels with validity instead.

Fee costs. In a multi-hop payment tx in a PCN, the intermediary user Ingrid charges a base fee (BF) for being online and offering the routing service and relative fee FR for locking the amounts of coins (tx.cash) and changing the balance in the channel, so that $\mathsf{fee}(\mathsf{tx}) := \mathsf{BF} + \mathsf{FR} \cdot \mathsf{tx.cash}$. Note that at the time of writing, the fees are $\mathsf{BF} = 1$ satoshi and $\mathsf{FR} = 0.000001$.

In a virtual channel setting, γ.Ingrid can charge a base fee to collaborate to open and close the virtual channel, and also a relative fee to lock collateral coins in the virtual channel. However, no fees per payment are charged by Ingrid as she does not participate in them. Let us now investigate the case of paying tx.cash in k micropayments of equal value. In PCN case, the total cost would be $\sum_{i=1}^{k} \mathsf{BF} + \mathsf{FR} \cdot \frac{\mathsf{tx.cash}}{k} = \mathsf{BF} \cdot k + \mathsf{FR} \cdot \mathsf{tx.cash}$. Whereas, in the virtual case, the parties first create a virtual channel γ with balance tx.cash, and they will handle the micropayments in γ. Thereby, the cost would be only the opening cost of the virtual channel, for which we assumed $\mathsf{BF} + \mathsf{FR} \cdot \mathsf{tx.cash}$. Thus, if Alice and Bob would make more than one transaction, i.e., $k > 1$, it is beneficial to use virtual channels for reducing the fee costs by $\mathsf{BF} \cdot (k-1)$.

6 CONCLUSION

Current PCNs route payments between two users through intermediate nodes, making the system less reliable (intermediate nodes might be offline), expensive (intermediate nodes charge a fee per payment) and privacy-invasive (intermediate nodes observe every payment they route). To mitigate this, recent work has introduced the concept of virtual channels, which involve intermediaries only in the creation of a bridge between payer and payee, who can later on independently perform arbitrarily many off-chain transactions. Unfortunately, existing constructions are only available for Ethereum, as they rely on its account model and Turing-complete scripting language.

In this work, we present the first virtual channel constructions that are built on the UTXO-model and require a script language supported by virtually every cryptocurrency, including Bitcoin. Our two protocols provide a tradeoff on who can offload the virtual channel (either payer and payee or intermediate one), similar to the preemptible vs non-preemptible virtual machines in the cloud setting. We formalize the security properties of virtual channels in the UC framework, proving that our protocols constitute a secure realization thereof. We have prototyped our protocols and evaluated their efficiency: for n sequential payments in the optimistic case, they require $11 + 2 \cdot (n - 1)$ off-chain transactions for a total of $4219 + 695 \cdot (n - 1)$ bytes, with no on-chain footprint.

We conjecture that it is possible to recursively build virtual channels on top of any two underlying channels (either ledger, virtual or a combination of them), requiring to adjust the timings for offloading channels: users of a virtual channel at layer k should have enough time to offload the (virtual/ledger) channels at layers 1 to $k - 1$. Additionally, we envision that while virtual channels without validity might serve as building block at any layer of recursion, virtual channels with validity period may be more suitable for the top layer as they have a predefined expiration time after which they would require to offload in any case all underlying layers. We plan to explore the recursive building of virtual channels in the near future. Additionally, we conjecture that virtual channels help with privacy, but we leave a formalization of this claim as an interesting future work, as it involves a quantitative analysis that falls of the scope of this work.

ACKNOWLEDGMENTS

This work was partly supported by the German Research Foundation (DFG) Emmy Noether Program *FA 1320/1-1*, by the *DFG CRC 1119 CROSSING* (project S7), by the German Federal Ministry of Education and Research (BMBF) *iBlockchain project* (grant nr. 16KIS0902), by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the *National Research Center for Applied Cybersecurity ATHENE*, by the European Research Council (ERC) under the European Unions Horizon 2020 research (grant agreement No 771527-BROWSEC), by the Austrian Science Fund (FWF) through PROFET (grant agreement P31621) and the Meitner program (grant agreement M 2608-G27), by the Austrian Research Promotion Agency (FFG) through the Bridge-1 project PR4DLT (grant agreement 13808694) and the COMET K1 projects SBA and ABC, and by CoBloX Labs.

REFERENCES

[1] 2018. Bitcoin Wiki: Payment Channels. https://en.bitcoin.it/wiki/Payment_channels.
[2] 2020. Bitcoin-Compatible Virtual Channels: Github repository. https://github.com/utxo-virtual-channels/vc.
[3] Andreas M. Antonopoulos. 2014. *Mastering Bitcoin: Unlocking Digital Crypto-Currencies* (1st ed.). O'Reilly Media, Inc.
[4] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostakova, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. 2020. Generalized Bitcoin-Compatible Channels. Cryptology ePrint Archive, Report 2020/476. https://eprint.iacr.org/2020/476.
[5] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. 2017. Bitcoin as a Transaction Ledger: A Composable Treatment. In *CRYPTO 2017, Part I (LNCS)*, Jonathan Katz and Hovav Shacham (Eds.), Vol. 10401. Springer, Heidelberg, 324–356.
[6] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. 2017. Consensus in the Age of Blockchains. *CoRR* abs/1711.03936 (2017). arXiv:1711.03936 http://arxiv.org/abs/1711.03936
[7] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd FOCS*. IEEE Computer Society Press, 136–145.
[8] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. 2007. Universally Composable Security with Global Setup. In *TCC 2007 (LNCS)*, Salil P. Vadhan (Ed.), Vol. 4392. Springer, Heidelberg, 61–85.
[9] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse, and Kristina Hostáková. 2019. Multi-party Virtual State Channels. In *EUROCRYPT 2019, Part I (LNCS)*, Vincent Rijmen and Yuval Ishai (Eds.). Springer, Heidelberg, 625–656. https://doi.org/10.1007/978-3-030-17653-2_21
[10] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. 2017. PERUN: Virtual Payment Channels over Cryptographic Currencies. Cryptology ePrint Archive, Report 2017/635. http://eprint.iacr.org/2017/635.
[11] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. 2017. Perun: Virtual Payment Hubs over Cryptographic Currencies. , 635 pages. http://eprint.iacr.org/2017/635 conference version accepted to the 40th IEEE Symposium on Security and Privacy (IEEE S&P) 2019.
[12] Stefan Dziembowski, Sebastian Faust, and Kristina Hostakova. to appear at ACM CCS 2018. General State Channel Networks. *IACR Cryptology ePrint Archive* (to appear at ACM CCS 2018). https://eprint.iacr.org/2018/320
[13] Oded Goldreich. 2006. *Foundations of Cryptography: Volume 1*. Cambridge University Press, New York, NY, USA.
[14] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. 2019. SoK: Off The Chain Transactions. Cryptology ePrint Archive, Report 2019/360. https://eprint.iacr.org/2019/360.
[15] George Kappos, Haaroon Yousaf, Ania M. Piotrowska, Sanket Kanjalkar, Sergi Delgado-Segura, Andrew Miller, and Sarah Meiklejohn. 2020. An Empirical Analysis of Privacy in the Lightning Network. *CoRR* abs/2003.12470 (2020). arXiv:2003.12470 https://arxiv.org/abs/2003.12470
[16] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. 2013. Universally Composable Synchronous Computation. In *TCC 2013 (LNCS)*, Amit Sahai (Ed.), Vol. 7785. Springer, Heidelberg, 477–498. https://doi.org/10.1007/978-3-642-36594-2_27
[17] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. 2017. Concurrency and Privacy with Payment-Channel Networks. In *ACM CCS 17*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 455–471.
[18] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. 2019. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. https://www.ndss-symposium.org/ndss-paper/anonymous-multi-hop-locks-for-blockchain-scalability-and-interoperability/
[19] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. http://bitcoin.org/bitcoin.pdf.
[20] Utz Nisslmueller, Klaus-Tycho Foerster, Stefan Schmid, and Christian Decker. 2020. Toward Active and Passive Confidentiality Attacks on Cryptocurrency Off-chain Networks. In *Proceedings of the 6th International Conference on Information Systems Security and Privacy, ICISSP 2020, Valletta, Malta, February 25-27, 2020*, Steven Furnell, Paolo Mori, Edgar R. Weippl, and Olivier Camp (Eds.). SCITEPRESS, 7–14. https://doi.org/10.5220/0009429200070014
[21] Joseph Poon and Thaddeus Dryja. 2016. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. Draft version 0.5.9.2, available at https://lightning.network/lightning-network-paper.pdf.
[22] Sergei Tikhomirov, Pedro Moreno-Sanchez, and Matteo Maffei. 2020. A Quantitative Analysis of Security, Anonymity and Scalability for the Lightning Network. Cryptology ePrint Archive, Report 2020/303. https://eprint.iacr.org/2020/303.
[23] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J. Knottenbelt.

2019. SoK: Communication Across Distributed Ledgers. Cryptology ePrint Archive, Report 2019/1128. https://eprint.iacr.org/2019/1128.

A EXAMPLE OF ROOTED TRANSACTIONS

In this section, we provide an example to clarify the concept of rooted transactions as described in Section 3.1.

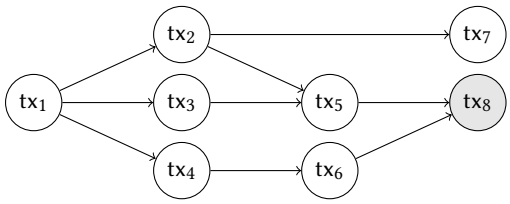


Figure 7: The root sets of transaction tx_8 **are** $\{\mathsf{tx}_1\}$**,** $\{\mathsf{tx}_2, \mathsf{tx}_3, \mathsf{tx}_4\}$**,** $\{\mathsf{tx}_5, \mathsf{tx}_6\}$**,** $\{\mathsf{tx}_4, \mathsf{tx}_5\}$ **and** $\{\mathsf{tx}_2, \mathsf{tx}_3, \mathsf{tx}_6\}$**.**

B ON THE USAGE OF THE UC-FRAMEWORK

To formally model the security of our construction, we use a synchronous version of the global UC framework (GUC) [8] which extends the standard UC framework [7] by allowing for a global setup. Since our model is essentially the same as in [4], which in turn follows [9, 12], parts of this section are taken verbatim from there.

Protocols and adversarial model. We consider a protocol π that runs between parties from the set $\mathcal{P} = \{P_1, \ldots, P_n\}$. A protocol is executed in the presence of an *adversary* $\mathcal{A}$ that takes as input a security parameter 1^λ (with $\lambda \in \mathbb{N}$) and an auxiliary input $z \in \{0,1\}^*$, and who can *corrupt* any party P_i at the beginning of the protocol execution (so-called static corruption). By corruption we mean that $\mathcal{A}$ takes full control over P_i and learns its internal state. Parties and the adversary $\mathcal{A}$ receive their inputs from a special entity – called the *environment* $\mathcal{E}$ – which represents anything "external" to the current protocol execution. The environment also observes all outputs returned by the parties of the protocol. In addition to the above entities, the parties can have access to ideal functionalities $\mathcal{H}_1, \ldots, \mathcal{H}_m$. In this case we say that the protocol π *works in the* $(\mathcal{H}_1, \ldots, \mathcal{H}_m)$*-hybrid model* and write $\pi^{\mathcal{H}_1, \ldots, \mathcal{H}_m}$.

Modeling time and communication. We assume a synchronous communication network, which means that the execution of the protocol happens in rounds. Let us emphasize that the notion of rounds is just an abstraction which simplifies our model and allows us to argue about the time complexity of our protocols in a natural way. We follow [9], which in turn follows [16], and formalize the notion of rounds via an ideal functionality $\widehat{\mathcal{F}}_{clock}$ representing "the clock". On a high level, the ideal functionality requires all honest parties to indicate that they are prepared to proceed to the next round before the clock is "ticked". We treat the clock functionality as a *global* ideal functionality using the GUC model. This means that all entities are always aware of the given round.

We assume that parties of a protocol are connected via authenticated communication channels with guaranteed delivery of exactly one round. This means that if a party P sends a message m to party Q in round t, party Q receives this message in beginning of round $t+1$. In addition, Q is sure that the message was sent by party P. The adversary can see the content of the message and can reorder messages that were sent in the same round. However, it can not modify, delay or drop messages sent between parties, or insert new messages. The assumptions on the communication channels are formalized as an ideal functionality $\mathcal{F}_{GDC}$. We refer the reader to [9] its formal description.

While the communication between two parties of a protocol takes exactly one round, all other communication – for example, between the adversary $\mathcal{A}$ and the environment $\mathcal{E}$ – takes zero rounds. For simplicity, we assume that any computation made by any entity takes zero rounds as well.

Handling coins. We model the money mechanics offered by UTXO cryptocurrencies, such as Bitcoin, via a *global* ideal functionality $\widehat{\mathcal{L}}$ using the GUC model. Our functionality is parameterized by a *delay parameter* Δ which upper bounded in the maximal number of rounds it takes to publish a valid transaction, and a signature scheme Σ. The functionality accepts messages from a fixed set of parties $\mathcal{P}$.

The ledger functionality $\widehat{\mathcal{L}}$ is initiated by the environment $\mathcal{E}$ via the following steps: (1) $\mathcal{E}$ instructs the ledger functionality to generate public parameter of the signature scheme pp; (2) $\mathcal{E}$ instructs every party $P \in \mathcal{P}$ to generate a key pair (sk_P, pk_P) and submit the public key pk_P to the ledger via the message (register, pk_P); (3) sets the initial state of the ledger meaning that it initialize a set $\mathtt{TX}$ defining all published transactions.

Once initialized, the state of $\widehat{\mathcal{L}}$ is public and can be accessed by all parties of the protocol, the adversary $\mathcal{A}$ and the environment $\mathcal{E}$. Any party $P \in \mathcal{P}$ can at any time post a transaction on the ledger via the message (post, tx). The ledger functionality waits for at most Δ rounds (the exact number of rounds is determined by the adversary). Thereafter, the ledger verifies the validity of the transaction and adds it to the transaction set $\mathtt{TX}$. The formal description of the ledger functionality follows.

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

The functionality accepts messages from all parties that are in the set $\mathcal{P}$ and maintains a $\mathtt{PKI}$ for those parties. The functionality maintains the set of all accepted transactions $\mathtt{TX}$ and all unspent transaction outputs $\mathtt{UTXO}$. The set $\mathcal{V}$ defines valid output conditions.

Initialize public keys: Upon (register, pk_P) $\overset{\tau_0}{\hookleftarrow} P$ and it is the first time P sends a registration message, add (pk_P, P) to $\mathtt{PKI}$.

Post transaction: Upon (post, tx) $\overset{\tau_0}{\hookleftarrow} P$, check that $|\mathtt{PKI}| = |\mathcal{P}|$. If not, drop the message, else wait until round $\tau_1 \leq \tau_0 + \Delta$ (the exact value of τ_1 is determined by the adversary). Then check if:

(1) The id is unique, i.e. for all $(t, \mathsf{tx}') \in \mathtt{TX}$, $\mathsf{tx}'.\mathsf{txid} \neq \mathsf{tx}.\mathsf{txid}$.
(2) All the inputs are unspent and the witness satisfies all the output conditions, i.e. for each $(tid, i) \in \mathsf{tx}.\mathsf{Input}$, there exists $(t, tid, i, \theta) \in \mathtt{UTXO}$ and $\theta.\varphi(\mathsf{tx}, t, \tau_1) = 1$.
(3) All outputs are valid, i.e. for each $\theta \in \mathsf{tx}.\mathsf{Output}$ it holds that $\theta.\mathsf{cash} > 0$ and $\theta.\varphi \in \mathcal{V}$.
(4) The value of the outputs is not larger than the value of the inputs. More formally, let $I := \{\mathtt{utxo} := (t, tid, i, \theta) \mid \mathtt{utxo} \in \mathtt{UTXO} \wedge$

$(tid, i) \in \mathsf{tx.Input}\}$, then it must hold that $\sum_{\theta' \in \mathsf{tx.Output}} \theta'.\mathsf{cash} \leq \sum_{\mathsf{utxo} \in I} \mathsf{utxo}.\theta.\mathsf{cash}$

(5) The absolute time-lock of the transaction has expired, i.e. it must hold that $\mathsf{tx.TimeLock} \leq \mathsf{now}$.

If all the above checks return true, add (τ_1, tx) to $\mathtt{TX}$, remove the spent outputs from $\mathtt{UTXO}$, i.e., $\mathtt{UTXO} := \mathtt{UTXO} \setminus I$ and add the outputs of tx to $\mathtt{UTXO}$, i.e., $\mathtt{UTXO} := \mathtt{UTXO} \cup \{(\tau_1, \mathsf{tx.txid}, i, \theta_i)\}_{i \in [n]}$ for $(\theta_1, \ldots, \theta_n) := \mathsf{tx.Output}$. Else, ignore the message.

Let us emphasize that our ledger functionality is fairly simplified. In reality, parties can join and leave the blockchain system dynamically. Moreover, we completely abstract from the fact that transactions are published in blocks which are proposed by parties and the adversary. Those and other features are captured by prior works, such as [5], that provide a more accurate formalization of the Bitcoin ledger in the UC framework [7]. However, interaction with such ledger functionality is fairly complex. To increase the readability of our channel protocols and ideal functionality, which is the main focus on our work, we decided for this simpler ledger.

The GUC-security definition. Let π be a protocol with access to the global ledger $\widehat{\mathcal{L}}(\Delta, \Sigma)$, the global clock $\widehat{\mathcal{F}}_{clock}$ and ideal functionalities $\mathcal{H}_1, \ldots, \mathcal{H}_m$. The output of an environment $\mathcal{E}$ interacting with a protocol π and an adversary $\mathcal{A}$ on input 1^λ and auxiliary input z is denoted as

$$\mathrm{EXE}^{\widehat{\mathcal{L}}(\Delta,\Sigma),\widehat{\mathcal{F}}_{clock},\mathcal{H}_1,\ldots,\mathcal{H}_m}_{\pi,\mathcal{A},\mathcal{E}}(\lambda, z).$$

Let $\phi_{\mathcal{F}}$ be the ideal protocol for an ideal functionality $\mathcal{F}$ with access to the global ledger $\widehat{\mathcal{L}}(\Delta, \Sigma)$ and the global clock $\widehat{\mathcal{F}}_{clock}$. This means that $\phi_{\mathcal{F}}$ is a trivial protocol in which the parties simply forward their inputs to the ideal functionality $\mathcal{F}$. The output of an environment $\mathcal{E}$ interacting with a protocol $\phi_{\mathcal{F}}$ and a adversary $\mathcal{S}$ (sometimes also call *simulator*) on input 1^λ and auxiliary input z is denoted as

$$\mathrm{EXE}^{\widehat{\mathcal{L}}(\Delta,\Sigma),\widehat{\mathcal{F}}_{clock}}_{\phi_{\mathcal{F}},\mathcal{S},\mathcal{E}}(\lambda, z).$$

We are now ready to state our main security definition which, informally, says that if a protocol π UC-realizes an ideal functionality $\mathcal{F}$, then any attack that can be carried out against the real-world protocol π can also be carried out against the ideal protocol $\phi_{\mathcal{F}}$.

Definition 1. *A protocol π working in a $(\mathcal{H}_1, \ldots, \mathcal{H}_m)$-hybrid model* UC-realizes an ideal functionality $\mathcal{F}$ with respect to a global ledger $\widehat{\mathcal{L}} := \widehat{\mathcal{L}}(\Delta, \Sigma)$ *and a global clock $\widehat{\mathcal{F}}_{clock}$ if for every adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that we have*

$$\left\{\mathrm{EXE}^{\widehat{\mathcal{L}},\widehat{\mathcal{F}}_{clock},\mathcal{H}_1,\ldots,\mathcal{H}_m}_{\pi,\mathcal{A},\mathcal{E}}(\lambda, z)\right\}_{\substack{\lambda \in \mathbb{N}, \\ z \in \{0,1\}^*}} \overset{c}{\approx} \left\{\mathrm{EXE}^{\widehat{\mathcal{L}},\widehat{\mathcal{F}}_{clock}}_{\phi_{\mathcal{F}},\mathcal{S},\mathcal{E}}(\lambda, z)\right\}_{\substack{\lambda \in \mathbb{N}, \\ z \in \{0,1\}^*}}$$

(where "$\overset{c}{\approx}$" denotes computational indistinguishability of distribution ensembles, see, e.g., [13]).

To simplify exposition, we omit the session identifiers *sid* and the sub-session identifiers *ssid*. Instead, we will use expressions like "message m is a reply to message m'". We believe that this approach improves readability.

C ADDITIONAL MATERIAL TO LEDGER CHANNELS

C.1 Ledger channel functionality

For completeness, we recall the ledger channel ideal functionality from [4].

Ideal Functionality $\mathcal{F}_L(T, k)$

We abbreviate $Q := \gamma.\mathsf{otherParty}(P)$ for $P \in \gamma.\mathsf{endUsers}$.

Create

Upon $(\mathtt{CREATE}, \gamma, tid_P) \overset{\tau_0}{\hookleftarrow} P$, let $\mathcal{S}$ define $T_1 \leq T$ and:

Both agreed: If already received $(\mathtt{CREATE}, \gamma, tid_Q) \overset{\tau}{\hookleftarrow} Q$, where $\tau_0 - \tau \leq T_1$, wait if in round $\tau_1 \leq \tau + \Delta + T_1$ a transaction tx, with $\mathsf{tx.Input} = (tid_P, tid_Q)$ and $\mathsf{tx.Output} = (\gamma.\mathsf{cash}, \varphi)$, appears on the ledger $\widehat{\mathcal{L}}$. If yes, set $\Gamma(\gamma.\mathsf{id}) := (\gamma, \mathsf{tx})$ and $(\mathtt{CREATED}, \gamma.\mathsf{id}) \overset{\tau_1}{\hookrightarrow} \gamma.\mathsf{endUsers}$. Else stop.

Wait for Q: Else store the message and stop.

Update

Upon $(\mathtt{UPDATE}, id, \vec{\theta}, t_{\mathrm{stp}}) \overset{\tau_0}{\hookleftarrow} P$, let $\mathcal{S}$ define $T_1, T_2 \leq T$, parse $(\gamma, \mathsf{tx}) := \Gamma(id)$ and proceed as follows:

(1) In round $\tau_1 \leq \tau_0 + T$, let $\mathcal{S}$ set $|\vec{tid}| = k$. Then $(\mathtt{UPDATE\text{-}REQ}, id, \vec{\theta}, t_{\mathrm{stp}}, \vec{tid}) \overset{\tau_1}{\hookrightarrow} Q$ and $(\mathtt{SETUP}, id, \vec{tid}) \overset{\tau_1}{\hookrightarrow} P$.

(2) If $(\mathtt{SETUP\text{-}OK}, id) \overset{\tau_2 \leq \tau_1 + t_{\mathrm{stp}}}{\longleftarrow} P$, then $(\mathtt{SETUP\text{-}OK}, id) \overset{\tau_2 + T_1}{\longrightarrow} Q$. Else stop.

(3) If $(\mathtt{UPDATE\text{-}OK}, id) \overset{\tau_2 + T_1}{\longleftarrow} Q$, then $(\mathtt{UPDATE\text{-}OK}, id) \overset{\tau_2 + 2T_1}{\longrightarrow} P$. Else distinguish:
- If Q honest or if instructed by $\mathcal{S}$, stop (update rejected).
- Else execute $\mathtt{L\text{-}ForceClose}(id)$ and stop.

(4) If $(\mathtt{REVOKE}, id) \overset{\tau_2 + 2T_1}{\longleftarrow} P$, $(\mathtt{REVOKE\text{-}REQ}, id) \overset{\tau_2 + 2T_1 + T_2}{\longrightarrow} Q$. Else execute $\mathtt{L\text{-}ForceClose}(id)$ and stop.

(5) If $(\mathtt{REVOKE}, id) \overset{\tau_2 + 2T_1 + T_2}{\longleftarrow} Q$, set $\gamma.\mathsf{st} = \vec{\theta}$ and $\Gamma(id) := (\gamma, \mathsf{tx})$. Then $(\mathtt{UPDATED}, id, \vec{\theta}) \overset{\tau_2 + 2T_1 + 2T_2}{\longrightarrow} \gamma.\mathsf{endUsers}$ and stop. Else distinguish:
- If Q honest, execute $\mathtt{L\text{-}ForceClose}(id)$ and stop.
- If Q corrupt, and wait for Δ rounds. If tx still unspent, then set $\vec{\theta}_{old} := \gamma.\mathsf{st}$, $\gamma.\mathsf{st} := \{\vec{\theta}_{old}, \vec{\theta}\}$ and $\Gamma(id) := (\gamma, \mathsf{tx})$. Execute $\mathtt{L\text{-}ForceClose}(id)$ and stop.

Close

Upon $(\mathtt{CLOSE}, id) \overset{\tau_0}{\hookleftarrow} P$, let $\mathcal{S}$ define $T_1 \leq T$ and distinguish:

Both agreed: If you received $(\mathtt{CLOSE}, id) \overset{\tau}{\hookleftarrow} Q$, where $\tau_0 - \tau \leq T_1$, let $(\gamma, \mathsf{tx}) := \Gamma(id)$ and distinguish:
- If in round $\tau_1 \leq \tau + T_1 + \Delta$ a transaction tx′, with $\mathsf{tx}'.\mathsf{Output} = \gamma.\mathsf{st}$ and $\mathsf{tx}'.\mathsf{Input} = \mathsf{tx.txid}$, appears on $\widehat{\mathcal{L}}$, set $\Gamma(id) := (\perp, \mathsf{tx})$, $(\mathtt{CLOSED}, id) \overset{\tau_1}{\hookrightarrow} \gamma.\mathsf{endUsers}$ and stop.
- If tx is still unspent in round $\tau + T_1 + \Delta$, output $(\mathtt{ERROR}) \overset{\tau + T_1 + \Delta}{\longrightarrow} \gamma.\mathsf{endUsers}$ and stop.

Wait for Q: Else wait for at most T_1 rounds to receive $(\mathtt{CLOSE}, id) \overset{\tau \leq \tau_0 + T_1}{\longleftarrow} Q$ (in that case option "Both agreed" is executed). If such message is not received, execute $\mathtt{L\text{-}ForceClose}(id)$ in round $\tau_0 + T_1$.

Punish (executed at the end of every round τ_0)

For each $(\gamma, \mathsf{tx}) \in \Gamma$ check if $\widehat{\mathcal{L}}$ contains tx' with $\mathsf{tx}'.\mathsf{Input} = \mathsf{tx}.\mathsf{txid}$. If yes, then distinguish:

Punish: For $P \in \gamma.\mathsf{endUsers}$ honest, the following must hold: in round $\tau_1 \leq \tau_0 + \Delta$, a transaction tx'' with $\mathsf{tx}''.\mathsf{Input} = \mathsf{tx}'.\mathsf{txid}$ and $\mathsf{tx}''.\mathsf{Output} = (\gamma.\mathsf{cash}, \mathtt{One\text{-}Sig}_{pk_P})$ appears on $\widehat{\mathcal{L}}$. Then send $(\mathtt{PUNISHED}, id) \overset{\tau_1}{\hookrightarrow} P$, set $\Gamma(id) := \bot$ and stop.

Close: Either $\Gamma(id) = (\bot, \mathsf{tx})$ before round $\tau_0 + \Delta$ (channels was peacefully closed) or in round $\tau_1 \leq \tau_0 + 2\Delta$ a transaction tx'', with $\mathsf{tx}''.\mathsf{Output} \in \gamma.\mathsf{st}$ and $\mathsf{tx}''.\mathsf{Input} = \mathsf{tx}'.\mathsf{txid}$, appears on $\widehat{\mathcal{L}}$ (channel is forcefully closed). In the latter case, set $\Gamma(id) := (\bot, \mathsf{tx})$ and $(\mathtt{CLOSED}, id) \overset{\tau_1}{\hookrightarrow} \gamma.\mathsf{endUsers}$.

Error: Otherwise $(\mathtt{ERROR}) \overset{\tau_0+2\Delta}{\longrightarrow} \gamma.\mathsf{endUsers}$.

Subprocedure $\mathtt{L\text{-}ForceClose}(id)$

Let τ_0 be the current round and $(\gamma, \mathsf{tx}) := \Gamma(id)$. If within Δ rounds tx is still an unspent transaction on $\widehat{\mathcal{L}}$, then $(\mathtt{ERROR}) \overset{\tau_0+\Delta}{\longrightarrow} \gamma.\mathsf{endUsers}$ and stop. Else, latest in round $\tau_0+3\Delta$, $m \in \{\mathtt{CLOSED}, \mathtt{PUNISHED}, \mathtt{ERROR}\}$ is output via Punish.

C.2 Wrapped ledger channel functionality

As discussed already in Section 4.1, for technical reason we cannot use the ledger channel functionality ($\mathcal{F}_L(T,k)$) for building virtual channels in a black-box way. The main problem comes from the offloading feature of virtual channel. In order to overcome these issues, we present $\mathcal{F}_{preL}(T,k)$, an ideal functionality that extends $\mathcal{F}_L(T,k)$ to supports the preparation generalized channels ahead of time and later registration of such prepared generalized channels. Technically, the functionality extension is done by *wrapping* the original functionality. Before we present the functionality wrapper $\mathcal{F}_{preL}(T,k)$ formally, let us explain each of its parts on high level.

Generalized channels. The functionality treats messages about standard generalized channels exactly as the functionality $\mathcal{F}_L(T,k)$ presented in Appendix C.1.

Creation. In order to pre-create a generalized channel γ, both endusers of the channel must the message $(\mathtt{PRE\text{-}CREATE}, \gamma, \mathsf{TX_f}, i, t_{ofl})$ to the ideal functionality. Here $\mathsf{TX_f}||i$ identifies the funding of the channel and $t_{ofl} \in \mathbb{N}$ represents the maximal number of round it should take to publish the channel funding transaction on-chain. If the functionality receives such a message from both parties within T rounds, it stores the channel γ, the funding identifier and the waiting time to a special channel set Γ_{pre}, and informs both parties about the successful pre-creation.

Update. The update process works similarly as for standard ledger channel with one difference. If the update process fails at some point (e.g., one of the parties does not revoke), the functionality does not call $\mathtt{L\text{-}ForceClose}$ since there is no ledger channel to be forcefully closed. Instead, it calls a subprocedure called $\mathtt{Wait\text{-}if\text{-}Register}$ which add a flag "$\mathtt{in\text{-}dispute}$" to the channel and waits for at most t_{ofl} rounds if the prepared channel is turned into a standard generalized channel (i.e., the corresponding funding transaction is added to the blockchain). If not, then it adds the new channel state back into the set of prepared (not yet full-fledged) channel states.

Register. The ideal functionality constantly monitors the ledger. Once the funding transaction of one of the channels in preparation appears on-chain, the functionality moves the information about the channel from the channel space Γ_{pre} to the channel space Γ. Moreover, if the channel in preparation was marked as "$\mathtt{in\text{-}dispute}$", then it immediately calls $\mathtt{L\text{-}ForceClose}$.

The formal functionality description on the functionality wrapper $\mathcal{F}_{preL}(T,k)$ follows.

Wrapped Ledger Channel Functionality $\mathcal{F}_{preL}(T,k)$

We abbreviate $Q := \gamma.\mathsf{otherParty}(P)$ for $P \in \gamma.\mathsf{endUsers}$.

Ledger Channels

Upon receiving a $\mathtt{CREATE}$, $\mathtt{UPDATE}$, $\mathtt{SETUP\text{-}OK}$, $\mathtt{UPDATE\text{-}OK}$, $\mathtt{REVOKE}$ or $\mathtt{CLOSE}$ message, then behave exactly as the functionality $\mathcal{F}_L(T,k)$.

Pre-Create

Upon $(\mathtt{PRE\text{-}CREATE}, \gamma, \mathsf{TX_f}, i, t_{ofl}) \overset{\tau_0}{\hookleftarrow} P$, let $\mathcal{S}$ define $T_1 \leq T$ and:

Both agreed: If already received $(\mathtt{PRE\text{-}CREATE}, \gamma, \mathsf{TX_f}, i, t_{ofl}) \overset{\tau}{\hookleftarrow} Q$, where $\tau_0 - \tau \leq T_1$, check that $\mathsf{TX_f}.\mathsf{Output}[i].\mathsf{cash} = \gamma.\mathsf{cash}$. If yes, set $\Gamma_{pre}(\gamma.\mathsf{id}) := (\gamma, \mathsf{TX_f}, t_{ofl})$ and $(\mathtt{PRE\text{-}CREATED}, \gamma.\mathsf{id}) \overset{\tau_0}{\hookrightarrow} \gamma.\mathsf{endUsers}$. Else stop.

Wait for Q: Else store the message and stop.

Pre-Update

Upon $(\mathtt{PRE\text{-}UPDATE}, id, \vec{\theta}, t_{\mathsf{stp}}) \overset{\tau_0}{\hookleftarrow} P$, let $\mathcal{S}$ define $T_1, T_2 \leq T$, parse $(\gamma, \mathsf{TX_f}, t) := \Gamma_{pre}(id)$ and proceed as follows:

(1) In round $\tau_1 \leq \tau_0 + T$, let $\mathcal{S}$ set $|\vec{tid}| = k$. Then $(\mathtt{PRE\text{-}UPDATE\text{-}REQ}, id, \vec{\theta}, t_{\mathsf{stp}}, \vec{tid}) \overset{\tau_1}{\hookrightarrow} Q$ and $(\mathtt{PRE\text{-}SETUP}, id, \vec{tid}) \overset{\tau_1}{\hookrightarrow} P$.

(2) If $(\mathtt{PRE\text{-}SETUP\text{-}OK}, id) \overset{\tau_2 \leq \tau_1 + t_{\mathsf{stp}}}{\longleftarrow} P$, then $(\mathtt{PRE\text{-}SETUP\text{-}OK}, id) \overset{\tau_2+T_1}{\longrightarrow} Q$. Else stop.

(3) In round $\tau_2 + T_1$ distinguish:
- If $(\mathtt{PRE\text{-}UPDATE\text{-}OK}, id) \overset{\tau_2+T_1}{\longleftarrow} Q$, then $(\mathtt{PRE\text{-}UPDATE\text{-}OK}, id) \overset{\tau_2+2T_1}{\longrightarrow} P$.
- If not and Q honest or if instructed by $\mathcal{S}$, $(\mathtt{PRE\text{-}UPDATE\text{-}REJECT}, id) \overset{\tau_2+2T_1}{\longrightarrow} P$.
- Else execute $\mathtt{Wait\text{-}if\text{-}Register}(id)$ and stop.

(4) If $(\mathtt{PRE\text{-}REVOKE}, id) \overset{\tau_2+2T_1}{\longleftarrow} P$, $(\mathtt{PRE\text{-}REVOKE\text{-}REQ}, id) \overset{\tau_2+2T_1+T_2}{\longrightarrow} Q$. Else execute $\mathtt{Wait\text{-}if\text{-}Register}(id)$ and stop.

(5) If $(\mathtt{PRE\text{-}REVOKE}, id) \overset{\tau_2+2T_1+T_2}{\longleftarrow} Q$, set $\gamma.\mathsf{st} = \vec{\theta}$ and $\Gamma_V(id) := (\gamma, \mathsf{TX_f})$. Then $(\mathtt{PRE\text{-}UPDATED}, id, \vec{\theta}) \overset{\tau_2+2T_1+2T_2}{\longrightarrow} \gamma.\mathsf{endUsers}$ and stop. Else $\mathtt{Wait\text{-}if\text{-}Register}(id)$ and stop.

Register – executed in every round

Let t_0 be the current round. For every $(\gamma, \mathsf{TX_f}) \in \Gamma_{pre}$ check if $\mathsf{TX_f}$ appears on the ledger $\widehat{\mathcal{L}}$. If yes, then $\Gamma_{pre}(\gamma.id) = \bot$ and $\Gamma(\gamma.id) = (\gamma, \mathsf{TX_f})$.

Subprocedure $\mathtt{Wait\text{-}if\text{-}Register}(id)$

Let τ_0 be the current round and $(\gamma, \mathsf{TX_f}, t_{ofl}) := \Gamma_{pre}(id)$.

(1) Set $\Gamma_{pre}(id) := (\gamma, \mathsf{TX_f}, t_{ofl}, \mathtt{in\text{-}dispute})$.

(2) Wait for t_{ofl} rounds. If after this time, $\Gamma_{pre}(id) \neq \bot$, then set $\vec{\theta}_{old} := \gamma.\mathsf{st}$, $\gamma.\mathsf{st} := \{\vec{\theta}_{old}, \vec{\theta}\}$ and $\Gamma_{pre}(id) := (\gamma, \mathsf{TX_f}, t_{ofl}, \mathtt{in\text{-}dispute})$.

C.3 Adaptor Signatures

Adaptor signatures have been introduced and used in the cryptocurrencies community for some time, but have been formalized for the fist time in [4]. These signatures not only allow for authentication as normal signatures schemes do, but also reveal a secret value upon publishing. Here we recall the definition of an adaptor signature scheme from [4]. In a nutshell, an adaptor signature is generated in two phases. First a pre-signature is computed w.r.t. some statement Y of a hard relation R e.g. $Y = g^y$ where g is the generator of the group $\mathbb{G}$ in which computing the discrete logarithm is hard. We define L_R to be the associated language for R defined as $L_R := \{Y \mid \exists y \text{ s.t. } (Y, y) \in R\}$. This pre-signature can be adapted to a full signature given a witness y for the statement Y, i.e. $(Y, y) \in R$. Furthermore, given the pre-signature and the adapted full signature one can extract a witness y. We now recall the definition for adaptor signature schemes from [4].

Definition 2 (Adaptor Signature Scheme). *An adaptor signature scheme wrt. a hard relation R and a signature scheme Σ = (Gen, Sign, Vrfy) consists of four algorithms $\Xi_{R.\Sigma}$ = (pSign, Adapt, pVrfy, Ext) defined as:*

- $\mathsf{pSign}_{sk}(m, Y)$**:** *is a PPT algorithm that on input a secret key sk, message $m \in \{0,1\}^*$ and statement $Y \in L_R$, outputs a pre-signature $\tilde{\sigma}$.*
- $\mathsf{pVrfy}_{pk}(m, Y; \tilde{\sigma})$**:** *is a DPT algorithm that on input a public key pk, message $m \in \{0,1\}^*$, statement $Y \in L_R$ and pre-signature $\tilde{\sigma}$, outputs a bit b.*
- $\mathsf{Adapt}(\tilde{\sigma}, y)$**:** *is a DPT algorithm that on input a pre-signature $\tilde{\sigma}$ and witness y, outputs a signature σ.*
- $\mathsf{Ext}(\sigma, \tilde{\sigma}, Y)$**:** *is a DPT algorithm that on input a signature σ, pre-signature $\tilde{\sigma}$ and statement $Y \in L_R$, outputs a witness y such that $(Y, y) \in R$, or $\perp$.*

We now briefly recall the properties that an adaptor signature scheme must satisfy and refer the reader to [4] for the formal definitions.

Correctness: An adaptor signature should not only satisfy the standard signature correctness, but it must also satisfy *pre-signature correctness*. This property guarantees that if a pre-signature is generated honestly (wrt. a statement $Y \in L_R$), it can be *adapted* into a valid signature such that a witness for Y can be extracted.

Existential unforgeablity under chosen message attack for adaptor signatures: Unforgeability for adaptor signatures is very similar to the normal definition of existential unforgeability under chosen message attacks for digital signatures, but it additionally requires that producing a forged signature σ for a message m is hard even if the adversary is given a pre-signature on the challenge message m w.r.t. a random statement $Y \in L_R$.

Pre-signature adaptability: Intuitively it is required that any *valid* pre-signature w.r.t. Y (even when produced by a malicious signer) can be completed into a valid signature using the witness y where $(Y, y) \in R$.

Witness extractability: In a nutshell, this property states that given a valid signature/pre-signatue pair $(\sigma, \tilde{\sigma})$ for a message m with respect to a statement Y, one can extract the corresponding witness y.

C.4 Realizing the wrapped functionality

While [4] presents a protocol Π_L that realizes the ideal functionality $\mathcal{F}_L$, it does not say anything about our wrapped functionality $\mathcal{F}_{preL}$. In order to have such protocol, we design a *protocol wrapper* around the protocol Π_L and prove that such wrapped protocol, which we denote Π_{preL} realizes the ideal functionality $\mathcal{F}_{preL}$.

Let us stress that the protocol wrapper very closely follows the protocol for ledger channel. Below we stress the main difference and thereafter we formally define the protocol for completeness.

Pre-Create. The only deference between the pre-create and create is that in pre-create $\mathtt{TX_f}$ is neither generated by the parties nor posted on the ledger and is given as an input from the environment. Intuitively this is a funding transactions that might be posted in the future. Hence such channels are called pre-created or prepared channels.

Pre-Update. During the pre-update procedure, parties update the state of the pre-created channel as in normal ledger channels, but parties cannot directly force-close the channel since the funding transaction is not posted on the ledger yet. Hence in case of dispute parties first have to post this transaction on the ledger this is captured in calls to `Wait-if-Register` sub-procedure.

Register. This is a new procedure in order to capture the situation during which the funding transaction of a pre-created channel is posted on the ledger. In this case the pre-created channel is transformed into a normal ledger channel and is added to the list of ledger channels. Furthermore if this channel was in dispute, it is directly force closed.

To summarize, parties upon receiving one of the `PRE-UPDATE`, `PRE-SETUP-OK`, `PRE-UPDATE-OK` or `PRE-REVOKE` messages, behave as in the protocol Π_L with the following changes:

- Use the channel space Γ^P_{pre} instead of Γ^P.
- Add t_{ofl} rounds to the absolute time lock of new $\mathtt{TX_c}$.
- Replace calls to $\mathtt{L\text{-}ForceClose}^P$ by calls to $\mathtt{Wait\text{-}if\text{-}Register}^P$ which marks a channel to be in dispute.
- In case the reacting party peacefully rejects the update, output `PRE-UPDATE-REJECT` before you stop.
- When the protocol instructs you to output a m-message, where $m \in \{$`UPDATE-REQ`, `SETUP`, `SETUP-OK`, `UPDATE-OK`, `REVOKE-REQ`, `UPDATED`$\}$, then output `PRE-`m.

Wrapped Ledger Channel Protocol Π_{preL}

Below, we abbreviate $Q := \gamma.\mathsf{otherParty}(P)$ for $P \in \gamma.\mathsf{endUsers}$.

Ledger channels

Upon receiving a `CREATE`, `UPDATE`, `SETUP-OK`, `UPDATE-OK`, `REVOKE` or `CLOSE` message, then behave exactly as in the protocol Π_L.

Pre-Create

Party P upon $(\mathtt{PRE\text{-}CREATE}, \gamma, \mathtt{TX_f}, i, t_{ofl}) \overset{t_0}{\hookleftarrow} \mathcal{E}$:

(1) If $\mathtt{TX_f}.\mathsf{Output}[i].\mathsf{cash} \neq \gamma.\mathsf{cash}$, then ignore the message.

(2) Set $id := \gamma.\mathrm{id}$, generate $(R_P, r_P) \leftarrow \mathrm{GenR}$, $(Y_P, y_P) \leftarrow \mathrm{GenR}$ and send (createInfo, id, $\mathtt{TX_f}$, i, t_{ofl}, R_P, Y_P) $\overset{t_0}{\hookrightarrow} Q$.

(3) If (createInfo, id, $\mathtt{TX_f}$, i, t_{ofl}, R_Q, Y_Q) $\overset{t_0+1}{\hookleftarrow} Q$, create:

$$[\mathtt{TX_c}] := \mathtt{GenCommit}([\mathtt{TX_f}], I_P, I_Q, 0)$$
$$[\mathtt{TX_s}] := \mathtt{GenSplit}([\mathtt{TX_c}].\mathrm{txid} \| 1, \gamma.\mathrm{st})$$

for $I_P := (pk_P, R_P, Y_P)$, $I_Q := (pk_Q, R_Q, Y_Q)$. Else stop.

(4) Compute $s_\mathsf{c}^P \leftarrow \mathrm{pSign}_{sk_P}([\mathtt{TX_c}], Y_Q)$, $s_\mathsf{s}^P \leftarrow \mathrm{Sign}_{sk_P}([\mathtt{TX_s}])$ and send (createCom, id, s_c^P, s_s^P) $\overset{t_0+1}{\hookrightarrow} Q$.

(5) If (createCom, id, s_c^Q, s_s^Q) $\overset{t_0+2}{\hookleftarrow} Q$, s.t. $\mathrm{pVrfy}_{pk_Q}([\mathtt{TX_c}], Y_P; s_\mathsf{c}^Q) = 1$ and $\mathrm{Vrfy}_{pk_Q}([\mathtt{TX_s}]; s_\mathsf{s}^Q) = 1$, set

$$\mathtt{TX_c} := ([\mathtt{TX_c}], \{\mathrm{Sign}_{sk_P}([\mathtt{TX_c}]), \mathrm{Adapt}(s_\mathsf{c}^Q, y_P)\})$$
$$\mathtt{TX_s} := ([\mathtt{TX_s}], \{s_\mathsf{s}^P, s_\mathsf{s}^Q\})$$
$$\Gamma_{pre}^P(\gamma.\mathrm{id}) := (\gamma, \mathtt{TX_f}, (\mathtt{TX_c}, r_P, R_Q, Y_Q, s_\mathsf{c}^P), \mathtt{TX_s}, t_{ofl}).$$

and send ($\mathtt{PRE\text{-}CREATED}$, id) $\overset{t_0+2}{\hookrightarrow} \mathcal{E}$.

Pre-Update

Party P upon ($\mathtt{PRE\text{-}UPDATE}$, id, $\vec{\theta}$, t_{stp}) $\overset{t_0^P}{\hookleftarrow} \mathcal{E}$

(1) Generate $(R_P, r_P) \leftarrow \mathrm{GenR}$, $(Y_P, y_P) \leftarrow \mathrm{GenR}$ and send the message (updateReq, id, $\vec{\theta}$, t_{stp}, R_P, Y_P) $\overset{t_0^P}{\hookrightarrow} Q$.

Party Q upon (updateReq, id, $\vec{\theta}$, t_{stp}, R_P, Y_P) $\overset{t_0^Q}{\hookleftarrow} P$

(2) Generate $(R_Q, r_Q) \leftarrow \mathrm{GenR}$ and $(Y_Q, y_Q) \leftarrow \mathrm{GenR}$.

(3) Extract $\mathtt{TX_f}$ and t_{ofl} from $\Gamma_{pre}^P(id)$.

(4) Set $t_{\mathrm{lock}} := t_0^Q + t_{\mathrm{stp}} + 4 + \Delta + t_{ofl}$ and

$$[\mathtt{TX_c}] := \mathtt{GenCommit}([\mathtt{TX_f}], I_P, I_Q, t_{\mathrm{lock}})$$
$$[\mathtt{TX_s}] := \mathtt{GenSplit}([\mathtt{TX_c}].\mathrm{txid} \| 1, \vec{\theta})$$

where $I_P := (pk_P, R_P, Y_P)$, $I_Q := (pk_Q, R_Q, Y_Q)$.

(5) Sign $s_\mathsf{s}^Q \leftarrow \mathrm{Sign}_{sk_Q}([\mathtt{TX_s}])$, send (updateInfo, id, R_Q, Y_Q, s_s^Q) $\overset{t_0^Q}{\hookrightarrow} P$, ($\mathtt{PRE\text{-}UPDATE\text{-}REQ}$, id, $\vec{\theta}$, t_{stp}, $\mathtt{TX_s}.\mathrm{txid}$) $\overset{t_0^Q+1}{\hookrightarrow} \mathcal{E}$.

Party P upon (updateInfo, id, h_Q, Y_Q, s_s^Q) $\overset{t_0^P+2}{\hookleftarrow} Q$

(6) Extract $\mathtt{TX_f}$ and t_{ofl} from $\Gamma_{pre}^Q(id)$.

(7) Set $t_{\mathrm{lock}} := t_0^P + t_{\mathrm{stp}} + 5 + \Delta + t_{ofl}$, and

$$[\mathtt{TX_c}] := \mathtt{GenCommit}([\mathtt{TX_f}], I_P, I_Q, t_{\mathrm{lock}})$$
$$[\mathtt{TX_s}] := \mathtt{GenSplit}([\mathtt{TX_c}].\mathrm{txid} \| 1, \vec{\theta}),$$

for $I_P := (pk_P, R_P, Y_P)$ and $I_Q := (pk_Q, R_Q, Y_Q)$. If it holds that $\mathrm{Vrfy}_{pk_Q}([\mathtt{TX_s}]; s_\mathsf{s}^Q) = 1$, ($\mathtt{PRE\text{-}SETUP}$, id, $\mathtt{TX_s}.\mathrm{txid}$) $\overset{t_0^P+2}{\hookrightarrow} \mathcal{E}$. Else stop.

(8) If ($\mathtt{PRE\text{-}SETUP\text{-}OK}$, id) $\overset{t_1^P \le t_0^P+2+t_{\mathrm{stp}}}{\hookleftarrow} \mathcal{E}$, compute the values $s_\mathsf{c}^P \leftarrow \mathrm{pSign}_{sk_P}([\mathtt{TX_c}], Y_Q)$, $s_\mathsf{s}^P \leftarrow \mathrm{Sign}_{sk_P}([\mathtt{TX_s}])$ and send the message (updateComP, id, s_c^P, s_s^P) $\overset{t_1^P}{\hookrightarrow} Q$. Else stop.

Party Q

(9) If (updateComP, id, s_c^P, s_s^P) $\overset{t_1^Q \le t_0^Q+2+t_{\mathrm{stp}}}{\hookleftarrow} P$, s.t. $\mathrm{pVrfy}_{pk_P}([\mathtt{TX_c}], Y_Q; s_\mathsf{c}^P) = 1$ and $\mathrm{Vrfy}_{pk_P}([\mathtt{TX_s}]; s_\mathsf{s}^P) = 1$, output ($\mathtt{PRE\text{-}SETUP\text{-}OK}$, id) $\overset{t_1^Q}{\hookrightarrow} \mathcal{E}$. Else stop.

(10) If ($\mathtt{PRE\text{-}UPDATE\text{-}OK}$, id) $\overset{t_1^Q}{\hookleftarrow} \mathcal{E}$, pre-sign $s_\mathsf{c}^Q \leftarrow \mathrm{pSign}([\mathtt{TX_c}], Y_P)$ and send (updateComQ, id, s_c^Q) $\overset{t_1^Q}{\hookrightarrow} P$. Else send the message (updateNotOk, id, r_Q) $\overset{t_1^Q}{\hookrightarrow} P$ and stop.

Party P

(11) In round $t_1^P + 2$ distinguish the following cases:

- If (updateComQ, id, s_c^Q) $\overset{t_1^P+2}{\hookleftarrow} Q$, s.t. $\mathrm{pVrfy}_{pk_Q}([\mathtt{TX_c}], Y_P; s_\mathsf{c}^Q) = 1$, output ($\mathtt{PRE\text{-}UPDATE\text{-}OK}$, id) $\overset{t_1^P+2}{\hookrightarrow} \mathcal{E}$.
- If (updateNotOk, id, r_Q) $\overset{t_1^P+2}{\hookleftarrow} Q$, s.t. $(R_Q, r_Q) \in R$, add $\Theta^P(id) := \Theta^P(id) \cup ([\mathtt{TX_c}], r_Q, Y_Q, s_\mathsf{c}^P)$, output the message ($\mathtt{PRE\text{-}UPDATE\text{-}REJECT}$) $\overset{t_1^P+2}{\hookrightarrow} \mathcal{E}$ and stop.
- Else, execute the procedure $\mathtt{Wait\text{-}if\text{-}Register}^P(id)$ and stop.

(12) If ($\mathtt{PRE\text{-}REVOKE}$, id) $\overset{t_1^P+2}{\hookleftarrow} \mathcal{E}$, parse $\Gamma_{pre}^P(id)$ as $(\gamma, \mathtt{TX_f}, (\overline{\mathtt{TX}}_\mathsf{c}, \bar{r}_P, \bar{R}_Q, \bar{Y}_Q, \bar{s}_{\mathsf{Com}}^P), \overline{\mathtt{TX}}_\mathsf{s})$ and update the channel space as $\Gamma_{pre}^P(id) := (\gamma, \mathtt{TX_f}, (\mathtt{TX_c}, r_P, R_Q, Y_Q, s_\mathsf{c}^P), \mathtt{TX_s})$, for $\mathtt{TX_s} := ([\mathtt{TX_s}], \{s_\mathsf{s}^P, s_\mathsf{s}^Q\})$ and $\mathtt{TX_c} := ([\mathtt{TX_c}], \{\mathrm{Sign}_{sk_P}([\mathtt{TX_c}]), \mathrm{Adapt}(s_\mathsf{c}^Q, y_P)\})$., and send (revokeP, id, $\bar{r}_P$) $\overset{t_1^P+2}{\hookrightarrow} Q$. Else, execute $\mathtt{Wait\text{-}if\text{-}Register}^P(id)$ and stop.

Party Q

(13) Parse $\Gamma_{pre}^Q(id)$ as $(\gamma, \mathtt{TX_f}, (\overline{\mathtt{TX}}_\mathsf{c}, \bar{r}_Q, \bar{R}_P, \bar{Y}_P, \bar{s}_{\mathsf{Com}}^Q), \overline{\mathtt{TX}}_\mathsf{s})$. If (revokeP, id, $\bar{r}_P$) $\overset{t_1^Q+2}{\hookleftarrow} P$, s.t. $(\bar{R}_P, \bar{r}_P) \in R$, ($\mathtt{PRE\text{-}REVOKE\text{-}REQ}$, id) $\overset{t_1^Q+2}{\hookrightarrow} \mathcal{E}$. Else execute $\mathtt{Wait\text{-}if\text{-}Register}^Q(id)$ and stop.

(14) If ($\mathtt{PRE\text{-}REVOKE}$, id) $\overset{t_1^Q+2}{\hookleftarrow} \mathcal{E}$ as a reply, set

$$\Theta^Q(id) := \Theta^Q(id) \cup ([\overline{\mathtt{TX}}_\mathsf{c}], \bar{r}_P, \bar{Y}_P, \bar{s}_{\mathsf{Com}}^Q)$$
$$\Gamma_{pre}^Q(id) := (\gamma, \mathtt{TX_f}, (\mathtt{TX_c}, r_Q, R_P, Y_P, s_\mathsf{c}^Q), \mathtt{TX_s}),$$

for $\mathtt{TX_s} := ([\mathtt{TX_s}], \{s_\mathsf{s}^P, s_\mathsf{s}^Q\})$, $\mathtt{TX_c} := ([\mathtt{TX_c}], \{\mathrm{Sign}_{sk_Q}([\mathtt{TX_c}]), \mathrm{Adapt}(s_\mathsf{c}^P, y_Q)\})$, and send (revokeQ, id, $\bar{r}_Q$) $\overset{t_1^Q+2}{\hookrightarrow} P$. In the next round ($\mathtt{PRE\text{-}UPDATED}$, id) $\overset{t_1^Q+3}{\hookrightarrow} \mathcal{E}$ and stop. Else, in round $t_1^Q + 2$, execute $\mathtt{Wait\text{-}if\text{-}Register}^Q(id)$ and stop.

Party P

(15) If (revokeQ, id, $\bar{r}_Q$) $\overset{t_1^P+4}{\hookleftarrow} Q$ s.t. $(\bar{R}_Q, \bar{r}_Q) \in R$, then set $\Theta^P(id) := \Theta^P(id) \cup ([\overline{\mathtt{TX}}_\mathsf{c}], \bar{r}_Q, \bar{Y}_Q, \bar{s}_{\mathsf{Com}}^P)$ and ($\mathtt{PRE\text{-}UPDATED}$, id) $\overset{t_1^P+4}{\hookrightarrow} \mathcal{E}$. Else execute $\mathtt{Wait\text{-}if\text{-}Register}^P(id)$ and stop.

Register

Party P in every round t_0: For each $id \in \{0,1\}^*$ s.t. $\Gamma_{pre}^P(id) \neq \perp$:

(1) Parse $\Gamma_{pre}^P(id) := (\gamma, \mathtt{TX_f}, (\mathtt{TX_c}, r_P, R_Q, Y_Q, s_\mathsf{c}^P), \mathtt{TX_s}, t_{ofl}, x)$

(2) If $\mathtt{TX_f}$ appeared on-chain in this round, then

(a) Set $\Gamma(id) := (\gamma, \mathtt{TX_f}, (\mathtt{TX_c}, r_P, R_Q, Y_Q, s_c^P), \mathtt{TX_s})$.
(b) Set $\Gamma^P_{pre}(id) := \bot$
(c) If $x = \mathtt{in\text{-}dispute}$, then call $\mathtt{L\text{-}ForceClose}^P(id)$.

Subprocedures

$\mathtt{GenCommit}([\mathtt{TX_f}], (pk_P, R_P, Y_P), (pk_Q, R_Q, Y_Q), t)$:
Let $(c, \mathtt{Multi\text{-}Sig}_{pk_P, pk_Q}) := \mathtt{TX_f}.\mathsf{Output}[1]$ and denote

$$\varphi_1 := \mathtt{Multi\text{-}Sig}_{\mathsf{ToKey}(R_Q), \mathsf{ToKey}(Y_Q), pk_P},$$
$$\varphi_2 := \mathtt{Multi\text{-}Sig}_{\mathsf{ToKey}(R_P), \mathsf{ToKey}(Y_P), pk_Q},$$
$$\varphi_3 := \mathtt{CheckRelative}_\Delta \wedge \mathtt{Multi\text{-}Sig}_{pk_P, pk_Q}.$$

Return [tx], where $\mathsf{tx.Input} = \mathtt{TX_f}.\mathsf{txid} \| 1$, $\mathsf{tx.Output} := (c, \varphi_1 \vee \varphi_2 \vee \varphi_3)$ and set tx.TimeLock to t if $t >$ now and to 0 otherwise.

$\mathtt{GenSplit}(tid, \vec{\theta})$:
Return [tx], where $\mathsf{tx.Input} := tid$ and $\mathsf{tx.Output} := \vec{\theta}$.

$\mathtt{Wait\text{-}if\text{-}Register}^P(id)$:
Let t_0 be the current round. Let $X := \Gamma^P_{pre}(id)$. Then set $\Gamma^P_{pre}(id) := (X, \mathtt{in\text{-}dispute})$.

Theorem 2. *Let Σ be a signature scheme that is existentially unforgeable against chosen message attacks, R a hard relation and $\Xi_{R,\Sigma}$ a secure adaptor signature scheme. Then for any ledger delay $\Delta \in \mathbb{N}$, the protocol Π_{preL} UC-realizes the ideal functionality $\mathcal{F}_{preL}(3, 1)$.*

D ADDITIONAL MATERIAL FOR VIRTUAL CHANNELS

We now formally describe the protocol $\Pi_V(T)$ that was discussed on a high level in Section 4. Since our goal it to prove that $\Pi_V(T)$ UC-realizes $\mathcal{F}_V(T)$, we need to discuss about parties deal with instruction about ledger channel as well as virtual channel.

D.1 Ledger Channels

As a first step, we discuss how parties deal with message about ledger channel or prepared ledger channel. On a high level, parties simply forward these instructions to the hybrid ideal functionality $\mathcal{F}_{preL}(T, 1)$. If the functionality sends a reply, the party forwards this reply to the environment. In addition to the message forwarding, parties stores information about the ledger channels in a channel space Γ_L. More precisely, once the a ledger channel is created or pre-created, party adds this channels to Γ_L. Once an existing ledger channel is updated or pre-updated, the party updates the latest state of the channel stored in Γ_L.

There is one technicality that we need to take care of. There are two different situations in which a party of a virtual channel protocol instructs the hybrid ideal functionality $\mathcal{F}_{preL}(T, 1)$ to precerate (resp. pre-update) a channel γ:

(1) Party receives a pre-create, resp. pre-update, instruction from the environment. As discussed above, in this case the party acts as a dummy party and forward the message to $\mathcal{F}_{preL}(T, 1)$.
(2) Party is creating, resp. updating, a virtual channel and hence is sending pre-create, resp. pre-update, messages to $\mathcal{F}_{preL}(T, 1)$.

Let us stress that while channels pre-created via option (1) exist in both the real and ideal world, channels pre-created via option (2) exist only in the real world. This is because the pre-creation of these channels was not initiated by the environment but by the parties of the virtual channel protocol. Hence, we need to make sure that the environment cannot "accidentally" update a channel pre-created via (2) since this would help the environment distinguish between the real and ideal world.

To this end, party in the case (1) modifies the identifier of the channel by adding a prefix "ledger". More precisely, if the environment makes a request about a channel with identifier id it forwards the instruction to the hybrid functionality but replaces id with ledger$\|id$. Analogously, if the hybrid functionality replies to this message, the party removes the prefix. This ensure that the environment cannot directly make any change on the ledger channels pre-created via option (2).

D.2 Create

The creation of a virtual channel was described on a high level in Section 4. The main idea is to update the two subchannels of the virtual channel and pre-create a new ledger channel corresponding to the virtual channel. Importantly, the update of the subchannel needs to be synchonized in order to ensure that either both updates complete (in which case the virtual channel is created) or both updates are rejected (in which case the virtual channel creation fails).

Since large part of the creation process is the same for channel with and without validity, our formal description is modularized.

Create a virtual channels - modular

Below we abbreviate $\mathcal{F}_{preL} := \mathcal{F}_{preL}(T, 1)$, $A := \gamma.\mathsf{Alice}$, $B := \gamma.\mathsf{Bob}$, $I = \gamma.\mathsf{Ingrid}$. For $P \in \gamma.\mathsf{endUsers}$, we denote $Q := \gamma.\mathsf{otherParty}(P)$.

Party $P \in \{A, B\}$

Upon receiving $(\mathtt{CREATE}, \gamma) \overset{t_0^P}{\longleftrightarrow} \mathcal{E}$ proceed as follows:

(1) Let $id_\alpha := \gamma.\mathsf{subchan}(P)$ and compute
$$\theta_P := \mathtt{GenVChannelOutput}(\gamma, P).$$
(2) Send $(\mathtt{UPDATE}, id_\alpha, \theta_P, t_{\mathsf{stp}}) \overset{t_0^P}{\hookrightarrow} \mathcal{F}_{preL}$.
(3) Upon receiving $(\mathtt{SETUP}, id_\alpha, tid_P) \overset{t_1^P \le t_0^P + T}{\longleftarrow} \mathcal{F}_{preL}$, engage in the subprotocol $\mathtt{SetupVChannel}$ with input (γ, tid_P).

Party I

Upon receiving $(\mathtt{CREATE}, \gamma) \overset{t_0^I}{\longleftrightarrow} \mathcal{E}$ proceed as follows:

(1) Set $id_\alpha = \gamma.\mathsf{subchan}(A)$, $id_\beta = \gamma.\mathsf{subchan}(B)$ and generate
$$\theta_A := \mathtt{GenVChannelOutput}(\gamma, A)$$
$$\theta_B := \mathtt{GenVChannelOutput}(\gamma, B)$$
(2) If in round $t_1^I \le t_0^I + T$ you have received both $(\mathtt{UPDATE\text{-}REQ}, id_\alpha, \theta_A, t_{\mathsf{stp}}, tid_A) \hookleftarrow \mathcal{F}_{preL}$ and $(\mathtt{UPDATE\text{-}REQ}, id_\beta, \theta_B, t_{\mathsf{stp}}, tid_B) \hookleftarrow \mathcal{F}_{preL}$, then engage in the subprotocol $\mathtt{SetupVChannel}$ with inputs (γ, tid_A, tid_B). Else stop.

Party $P \in \{A, B\}$

Wait until $t_2^P := t_1^P + t_{\mathsf{stp}}$. If the subprotocol completed successfully, then send $(\mathtt{SETUP\text{-}OK}, id_\alpha) \overset{t_2^P}{\hookrightarrow} \mathcal{F}_{preL}$. Else stop.

Party I

If in round $t_2^I \leq t_1^I + t_{\text{stp}} + T$ you receive both $(\texttt{SETUP-OK}, id_\alpha)) \hookleftarrow \mathcal{F}_{preL}$ and $(\texttt{SETUP-OK}, id_\beta)) \hookleftarrow \mathcal{F}_{preL}$, send $(\texttt{UPDATE-OK}, id_\alpha) \overset{t_2}{\hookrightarrow} \mathcal{F}_{preL}$ and $(\texttt{UPDATE-OK}, id_\beta) \overset{t_2}{\hookrightarrow} \mathcal{F}_{preL}$. Otherwise stop.

Party $P \in \{A, B\}$

(1) If you receive $(\texttt{UPDATE-OK}, id_\alpha) \overset{t_2^P \leq t_1^P + 2T}{\longleftarrow} \mathcal{F}_{preL}$, reply with $(\texttt{REVOKE}, id_\alpha) \overset{t_2^P + T}{\longrightarrow} \mathcal{F}_{preL}$. Otherwise stop.

Party I

If in round $t_3^I \leq t_2^I + 4T$ you have received both $(\texttt{REVOKE-REQ}, id_\alpha) \hookleftarrow \mathcal{F}_{preL}$ and $(\texttt{REVOKE-REQ}, id_\beta) \hookleftarrow \mathcal{F}_{preL}$, reply $(\texttt{REVOKE}, id_\alpha) \overset{t_3^I}{\hookrightarrow} \mathcal{F}_{preL}$ and $(\texttt{REVOKE}, id_\beta) \overset{t_3^I}{\hookrightarrow} \mathcal{F}_{preL}$ and update $\Gamma^I(\gamma.\text{id})$ from $(\bot, x)$ to (γ, x). Otherwise stop.

Party $P \in \{A, B\}$

Upon receiving $(\texttt{UPDATED}, id_\alpha) \overset{t_3^P \leq t_2^P + 3T}{\longleftarrow} \mathcal{F}_{preL}$, mark γ as created, i.e. update $\Gamma^P(\gamma.\text{id})$ from $(\bot, x)$ to (γ, x), and output $(\texttt{CREATED}, \gamma.\text{id}) \overset{t_3^P}{\hookrightarrow} \mathcal{E}$.

Function $\texttt{GenVChannelOutput}(\gamma, P)$

Return θ, where $\theta.\texttt{cash} = \gamma.\texttt{cash} + \gamma.\texttt{fee}/2$ and $\theta.\varphi$ is defined as follows

$$\theta.\varphi = \begin{cases} \texttt{Multi-Sig}_{\gamma.\texttt{users}} \vee (\texttt{One-Sig}_P \wedge \texttt{CheckRelative}_{(T+4\Delta)}), \\ \qquad\qquad \text{if } \gamma.\texttt{val} = \bot \\ \texttt{Multi-Sig}_{A,I} \vee (\texttt{One-Sig}_I \wedge \texttt{CheckLockTime}_{\gamma.\texttt{val}}), \\ \qquad\qquad \text{if } \gamma.\texttt{val} \neq \bot \wedge P = A \\ \texttt{Multi-Sig}_{B,I} \vee (\texttt{One-Sig}_B \wedge \texttt{CheckLockTime}_{\gamma.\texttt{val}+2\Delta}), \\ \qquad\qquad \text{if } \gamma.\texttt{val} \neq \bot \wedge P = B \end{cases}$$

Subprotocol `SetupVChannel`

Let t_0 be the current round.

Channels without validity

Party $P \in \{A, B\}$ on input (γ, tid_P)

(1) Create the body of the funding transactions:

$$\texttt{TX}_\texttt{f}^\gamma.\texttt{Input} := (tid_P, tid_Q)$$
$$\texttt{TX}_\texttt{f}^\gamma.\texttt{Output} := ((\gamma.\texttt{cash}, \texttt{Multi-Sig}_{\{\gamma.\texttt{endUsers}\}}), (\gamma.\texttt{cash} + \gamma.\texttt{fee}, \texttt{One-Sig}_{pk_I}))$$

(2) Send $(\texttt{PRE-CREATE}, \gamma, \texttt{TX}_\texttt{f}, 1, t_{ofl}) \overset{t_0}{\hookrightarrow} \mathcal{F}_{preL}$, where $t_{ofl} = 2T + 8\Delta$.

(3) If $(\texttt{PRE-CREATED}, \gamma.\text{id}) \overset{t_1 \leq t_0 + T}{\longleftarrow} \mathcal{F}_{preL}$, then sign the funding transaction, i.e. $s_\texttt{f}^P \leftarrow \text{Sign}_{sk_P}([\texttt{TX}_\texttt{f}^\gamma])$ and send $(\text{createFund}, \gamma.\text{id}, s_\texttt{f}^P, [\texttt{TX}_\texttt{f}^\gamma]) \overset{t_1}{\hookrightarrow} I$. Else stop.

Party I on input (γ, tid_A, tid_B)

(4) If you receive $(\text{createFund}, \gamma.\text{id}, s_\texttt{f}^A, [\texttt{TX}_\texttt{f}^\gamma]) \overset{t_2 \leq t_0 + T + 1}{\longleftarrow} A$ and $(\text{createFund}, \gamma.\text{id}, s_\texttt{f}^B, [\texttt{TX}_\texttt{f}^\gamma]) \overset{t_2}{\hookleftarrow} B$, verify the funding transaction and signatures of A and B, i.e. check:

$$\text{Vrfy}_{pk_A}([\texttt{TX}_\texttt{f}^\gamma]; s_\texttt{f}^A) = 1$$
$$\text{Vrfy}_{pk_B}([\texttt{TX}_\texttt{f}^\gamma], s_\texttt{f}^B) = 1$$
$$(tid_A, tid_B) = \texttt{TX}_\texttt{f}^\gamma.\texttt{Input}$$
$$(\gamma.\texttt{cash} + \gamma.\texttt{fee}, \texttt{One-Sig}_{pk_I}) \in \texttt{TX}_\texttt{f}^\gamma.\texttt{Output}.$$

(5) If all checks pass, sign the funding transaction, i.e. compute

$$s_\texttt{f}^I := \text{Sign}_{sk_I}([\texttt{TX}_\texttt{f}^\gamma]),$$
$$\texttt{TX}_\texttt{f}^\gamma := \{([\texttt{TX}_\texttt{f}^\gamma], s_\texttt{f}^A, s_\texttt{f}^B, s_\texttt{f}^I)\}.$$

Store $\Gamma^I(\gamma.\text{id}) := (\bot, \texttt{TX}_\texttt{f}^\gamma)$. Then send $(\text{createFund}, \gamma.\text{id}, s_\texttt{f}^B, s_\texttt{f}^I) \overset{t_2}{\hookrightarrow} A$ and $(\text{createFund}, \gamma.\text{id}, s_\texttt{f}^A, s_\texttt{f}^I) \overset{t_2}{\hookrightarrow} B$, and consider procedure successfully completed. Else stop.

Party $P \in \{A, B\}$

(6) Upon receiving $(\text{createFund}, \gamma.\text{id}, s_\texttt{f}^Q, s_\texttt{f}^I) \overset{t_1 + 1}{\longleftarrow} I$, verify all signatures, i.e. check:

$$\text{Vrfy}_{pk_Q}([\texttt{TX}_\texttt{f}^\gamma]; s_\texttt{f}^Q) = 1$$
$$\text{Vrfy}_{pk_I}([\texttt{TX}_\texttt{f}^\gamma], s_\texttt{f}^I) = 1.$$

If all checks pass define $\texttt{TX}_\texttt{f}^\gamma := \{([\texttt{TX}_\texttt{f}^\gamma], s_\texttt{f}^P, s_\texttt{f}^Q, s_\texttt{f}^I)\}$ and set $\Gamma^P(\gamma.\text{id}) := (\bot, \texttt{TX}_\texttt{f}^\gamma, tid_P)$ and consider procedure successfully completed. Else stop.

Channels with validity

Party A on input (γ, tid_A)

(1) Send $(\text{createInfo}, \gamma.\text{id}, tid_A) \overset{t_0}{\hookrightarrow} B$

(2) In round $t_1 := t_0 + 1$, create the body of the funding transaction:

$$\texttt{TX}_\texttt{f}^\gamma.\texttt{Input} := (tid_A)$$
$$\texttt{TX}_\texttt{f}^\gamma.\texttt{Output} := ((\gamma.\texttt{cash}, \texttt{Multi-Sig}_{\{\gamma.\texttt{endUsers}\}}), (\gamma.\texttt{fee}/2, \texttt{One-Sig}_{pk_I}))$$

(3) Send $(\texttt{PRE-CREATE}, \gamma, \texttt{TX}_\texttt{f}, 1, t_{ofl}) \overset{t_1}{\hookrightarrow} \mathcal{F}_{preL}$, for $t_{ofl} = \gamma.\texttt{val} + 3\Delta$.

(4) If $(\texttt{PRE-CREATED}, \gamma.\text{id}) \overset{t_2 \leq t_1 + T}{\longleftarrow} \mathcal{F}_{preL}$, then goto step (10). Else stop.

Party B on input (γ, tid_B)

(5) If $(\text{createInfo}, \gamma.\text{id}, tid_A) \overset{t_1 := t_0 + 1}{\longleftarrow} A$, then create the body of the funding and refund transactions:

$$\texttt{TX}_\texttt{f}^\gamma.\texttt{Input} := (tid_A)$$
$$\texttt{TX}_\texttt{f}^\gamma.\texttt{Output} := ((\gamma.\texttt{cash}, \texttt{Multi-Sig}_{\{\gamma.\texttt{endUsers}\}}), (\gamma.\texttt{fee}/2, \texttt{One-Sig}_{pk_I}))$$
$$\texttt{TX}_{\texttt{refund}}^\gamma.\texttt{Input} := (\texttt{TX}_\texttt{f}^\gamma.\texttt{txid}||2, tid_B)$$
$$\texttt{TX}_{\texttt{refund}}^\gamma.\texttt{Output} := (\gamma.\texttt{cash} + \gamma.\texttt{fee}, \texttt{One-Sig}_{pk_I}).$$

Else stop.

(6) Send $(\texttt{PRE-CREATE}, \gamma, \texttt{TX}_\texttt{f}, 1, t_{ofl}) \overset{t_1}{\hookrightarrow} \mathcal{F}_{preL}$, for $t_{ofl} = \gamma.\texttt{val} + 3\Delta$.

(7) If $(\texttt{PRE-CREATED}, \gamma.\text{id}) \xleftarrow{t_2 \le t_1+T} \mathcal{F}_{preL}$, then compute a signature on the refund transaction, i.e., $s^B_{\texttt{Ref}} \leftarrow \text{Sign}_{sk_B}([\texttt{TX}^\gamma_{\texttt{refund}}])$ and define $\Gamma^B(\gamma.id) := (\perp, [\texttt{TX}^\gamma_{\texttt{f}}], tid_B)$. Then, send $(\text{createFund}, \gamma.\text{id}, s^B_{\texttt{Ref}}, [\texttt{TX}^\gamma_{\texttt{refund}}], [\texttt{TX}^\gamma_{\texttt{f}}]) \xhookrightarrow{t_2} I$ and consider procedure successfully completed. Else stop.

Party I on input (γ, tid_A, tid_B)

(8) If $(\text{createFund}, \gamma.\text{id}, s^B_{\texttt{Ref}}, [\texttt{TX}^\gamma_{\texttt{refund}}], [\texttt{TX}^\gamma_{\texttt{f}}]) \xleftarrow{t_3 \le t_0+T+2} B$, verify the fund and refund transactions and signature of B, i.e. check:

$\text{Vrfy}_{sk_B}([\texttt{TX}^\gamma_{\texttt{refund}}]; s^B_{\texttt{Ref}}) = 1.$

$[\texttt{TX}^\gamma_{\texttt{refund}}].\text{Input} = (\texttt{TX}^\gamma_{\texttt{f}}.\text{txid}||2, tid_B),$

$[\texttt{TX}^\gamma_{\texttt{refund}}].\text{Output} = (\gamma.\text{cash} + \gamma.\text{fee}, \texttt{One-Sig}_{pk_I}),$

$[\texttt{TX}^\gamma_{\texttt{f}}].\text{Output}[2] = (\gamma.\text{fee}/2, \texttt{One-Sig}_{pk_I})$

If all checks pass, then sign the fund and refund transactions, i.e. compute

$s^I_{\texttt{Ref}} := \text{Sign}_{sk_I}([\texttt{TX}^\gamma_{\texttt{refund}}]),\ s^I_{\texttt{f}} := \text{Sign}_{sk_I}([\texttt{TX}^\gamma_{\texttt{f}}]),$

$\texttt{TX}^\gamma_{\texttt{refund}} := \{([\texttt{TX}^\gamma_{\texttt{refund}}], s^I_{\texttt{Ref}}, s^B_{\texttt{Ref}})\}.$

Else stop.

(9) Store $\Gamma^I(\gamma.\text{id}) := (\perp, [\texttt{TX}^\gamma_{\texttt{f}}], \texttt{TX}^\gamma_{\texttt{refund}}, tid_A, tid_B)$, send the message $(\text{createFund}, \gamma.\text{id}, s^I_{\texttt{f}}) \xhookrightarrow{t_3} A$, and consider procedure successfully completed.

Party A

(10) If you receive $(\text{createFund}, \gamma.\text{id}, s^I_{\texttt{f}}) \xleftarrow{t_2+2} I$, verify the signature, i.e. check $\text{Vrfy}_{pk_I}([\texttt{TX}^\gamma_{\texttt{f}}]; s^I_{\texttt{f}}) = 1$. If the check passes, compute a signature on the fund transaction:

$s^A_{\texttt{f}} := \text{Sign}_{sk_A}([\texttt{TX}^\gamma_{\texttt{f}}]),$

$\texttt{TX}^{\gamma,\texttt{A}}_{\texttt{f}} := \{([\texttt{TX}^\gamma_{\texttt{f}}], s^I_{\texttt{f}}, s^A_{\texttt{f}})\}.$

and set $\Gamma^A(\gamma.\text{id}) := (\perp, \texttt{TX}^{\gamma,\texttt{A}}_{\texttt{f}}, tid_A)$. Then consider procedure successfully completed. Else stop.

D.3 Update

As discussed in Section 4, in order to update a virtual channel, parties update the corresponding prepared channel. This is does in a black-box way via the hybrid functionality $\mathcal{F}_{preL}$. Hence, parties act as dummy parties as forward update instructions (modified by adding PRE–) to the hybrid functionality $\mathcal{F}_{preL}$ and forward the replies of the functionality (modified by removing PRE–) to the environment. In case the update fails, party offload the channel which allows to resolve disputes on-chain.

Update

Below we abbreviate $\mathcal{F}_{preL} := \mathcal{F}_{preL}(T, 1)$.

Initiating party P:

(1) Upon $(\texttt{UPDATE}, id, \vec{\theta}, t_{\text{stp}}) \xhookleftarrow{t_0} \mathcal{E}$, $(\texttt{PRE-UPDATE}, id, \vec{\theta}, t_{\text{stp}}) \xhookrightarrow{t_0} \mathcal{F}_{preL}$.

(2) If $(\texttt{PRE-SETUP}, id, tid_P) \xleftarrow{t_1 \le t_0+T} \mathcal{F}_{preL}$, $(\texttt{SETUP}, id, tid_P) \xhookrightarrow{t_1} \mathcal{E}$. Else stop.

(3) If $(\texttt{SETUP-OK}, id) \xleftarrow{t_2 \le t_1+t_{\text{stp}}} \mathcal{E}$, $(\texttt{PRE-SETUP-OK}, id) \xhookrightarrow{t_2} \mathcal{E}$. Else stop.

(4) Distinguish the following three cases:

- If $(\texttt{PRE-UPDATE-OK}, id) \xleftarrow{t_3 \le t_2+T} \mathcal{F}_{preL}$, $(\texttt{UPDATE-OK}, id) \xhookrightarrow{t_3} \mathcal{E}$.
- If $(\texttt{PRE-UPDATE-REJECT}, id) \xleftarrow{t_3 \le t_2+T} \mathcal{F}_{preL}$, then stop.
- Else execute the procedure $\texttt{Offload}^P(id)$ and stop.

(5) If $(\texttt{REVOKE}, id) \xhookleftarrow{t_3} \mathcal{E}$, $(\texttt{PRE-REVOKE}, id) \xhookrightarrow{t_3} \mathcal{F}_{preL}$. Else execute $\texttt{Offload}^P(id)$ and stop.

(6) If $(\texttt{PRE-UPDATED}, id) \xleftarrow{t_4 \le t_3+T} \mathcal{F}_{preL}$, update the channel space, i.e., let $\gamma := \Gamma^P(id)$, set $\gamma.\text{st} := \vec{\theta}$ and $\Gamma(id) := \gamma$. Then $(\texttt{UPDATED}, id) \xhookrightarrow{t_4} \mathcal{F}_{preL}$. Else execute $\texttt{Offload}^P(id)$ and stop.

Reacting party Q

(1) Upon $(\texttt{PRE-UPDATE-REQ}, id, \vec{\theta}, t_{\text{stp}}, tid) \xhookleftarrow{\tau_0} \mathcal{F}_{preL}$, $(\texttt{UPDATE-REQ}, id, \vec{\theta}, t_{\text{stp}}, tid) \xhookrightarrow{\tau_0} \mathcal{E}$.

(2) If $(\texttt{PRE-SETUP-OK}, id) \xleftarrow{\tau_1 \le \tau_0+t_{\text{stp}}+T} \mathcal{F}_{preL}$, $(\texttt{SETUP-OK}, id) \xhookrightarrow{\tau_1} \mathcal{E}$. Else stop.

(3) If $(\texttt{UPDATE-OK}, id) \xhookleftarrow{\tau_1} \mathcal{E}$, $(\texttt{PRE-UPDATE-OK}, id) \xhookrightarrow{\tau_1} \mathcal{F}_{preL}$. Else stop.

(4) If $(\texttt{PRE-REVOKE-REQ}, id) \xleftarrow{\tau_2 \le \tau_1+T} \mathcal{F}_{preL}$, $(\texttt{REVOKE-REQ}, id) \xhookrightarrow{\tau_2} \mathcal{E}$. Else execute $\texttt{Offload}^Q(id)$ and stop.

(5) If $(\texttt{REVOKE}, id) \xhookleftarrow{\tau_2} \mathcal{E}$, $(\texttt{PRE-REVOKE}, id) \xhookrightarrow{\tau_2} \mathcal{F}_{preL}$. Else execute $\texttt{Offload}^Q(id)$ and stop.

(6) Upon $(\texttt{PRE-UPDATED}, id) \xleftarrow{\tau_3 \le \tau_2+T} \mathcal{F}_{preL}$, update the channel space, i.e., let $\gamma := \Gamma^Q(id)$, set $\gamma.\text{st} := \vec{\theta}$ and $\Gamma(id) := \gamma$. Then $(\texttt{UPDATED}, id) \xhookrightarrow{\tau_3} \mathcal{E}$.

D.4 Offload

As a next step, we define the offloading process which transforms a virtual channel into a ledger channel. Let us stress that offloading can be triggered either by the environment via a message OFFLOAD or internally by parties when executing an update or close. To avoid code repetition, we define a procedure $\texttt{Offload}^P(id)$ and instruct parties upon receiving $(\texttt{OFFLOAD}, id) \xhookleftarrow{t_0} \mathcal{E}$ to simply call $\texttt{Offload}^P(id)$.

Since channels with validity are constructed in a different way than channel without validity, the procedure is defined for the two cases separately.

Subprocedure $\texttt{Offload}^P(id)$

Below we abbreviate $\mathcal{F}_{preL} := \mathcal{F}_{preL}(T, 1)$, $A := \gamma.\text{Alice}$ and $B := \gamma.\text{Bob}$ and $I = \gamma.\text{Ingrid}$. For $P \in \gamma.\text{endUsers}$, we denote $Q := \gamma.\text{otherParty}(P)$. Let t_0 be the current round.

Channels without validity

$P \in \{A, B\}$

(1) Extract γ and $\texttt{TX}^\gamma_{\texttt{f}}$ from $\Gamma^P(id)$ and tid_P, tid_Q from $\texttt{TX}^\gamma_{\texttt{f}}$. Then define $id_\alpha := \gamma.\text{subchan}(P)$ and send $(\texttt{CLOSE}, id_\alpha) \xhookrightarrow{t_0} \mathcal{F}_{preL}$.

(2) If you receive $(\texttt{CLOSED}, id_\alpha) \xleftarrow{t_1 \le t_0+T+3\Delta} \mathcal{F}_{preL}$, then continue. Else set $\Gamma^P(\gamma.\text{id}) = \perp$ and stop.

(3) Let $T_2 := t_1 + T + 3\Delta$ and distinguish:

- If in round $t_2 \leq T_2$ a transaction with tid_Q appeared on $\widehat{\mathcal{L}}$, then (post, $\mathtt{TX}_{\mathsf{f}}^{\gamma}$) $\overset{t_2}{\hookrightarrow}$ $\widehat{\mathcal{L}}$.
- Else in round T_2 create the punishment transaction $\mathtt{TX}_{\mathsf{pun}}$ as $\mathtt{TX}_{\mathsf{pun}}.\mathsf{Input} := tid_P$, $\mathtt{TX}_{\mathsf{pun}}.\mathsf{Output} := (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \mathtt{One\text{-}Sig}_{pk_P})$ and $\mathtt{TX}_{\mathsf{pun}}.\mathsf{Witness} := \mathsf{Sign}_{sk_P}([\mathtt{TX}_{\mathsf{pun}}])$. Then (post, $\mathtt{TX}_{\mathsf{pun}}$) $\overset{T_2}{\hookrightarrow}$ $\widehat{\mathcal{L}}$.

(4) Let $T_3 := t_2 + \Delta$ and distinguish the following two cases:
- The transaction $\mathtt{TX}_{\mathsf{f}}^{\gamma}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3 \leq T_3$, then update $\Gamma_L^P(id) := \Gamma^P(id)$ and set $m :=$ offloaded.
- The transaction $\mathtt{TX}_{\mathsf{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3 \leq T_3$, then set $m :=$ punished.

(5) Set $\Gamma^P(id) = \bot$ and return m in round t_3.

Party I

(1) Extract γ and $\mathtt{TX}_{\mathsf{f}}^{\gamma}$ from $\Gamma^I(id)$ and tid_A, tid_B from $\mathtt{TX}_{\mathsf{f}}^{\gamma}$. Then define $id_\alpha := \gamma.\mathsf{subchan}(A)$, $id_\beta := \gamma.\mathsf{subchan}(B)$ and send the messages $(\mathtt{CLOSE}, id_\alpha) \overset{t_0}{\hookrightarrow} \mathcal{F}_{preL}$ and $(\mathtt{CLOSE}, id_\beta) \overset{t_0}{\hookrightarrow} \mathcal{F}_{preL}$.

(2) If you receive both messages $(\mathtt{CLOSED}, id_\alpha) \overset{t_1^A \leq t_0+T+3\Delta}{\longleftarrow} \mathcal{F}_{preL}$ and $(\mathtt{CLOSED}, id_\beta) \overset{t_1^B \leq t_0+T+3\Delta}{\longleftarrow} \mathcal{F}_{preL}$, publish (post, $\mathtt{TX}_{\mathsf{f}}^{\gamma}$) $\overset{t_1}{\hookrightarrow}$ $\widehat{\mathcal{L}}$, where $t_1 := \max\{t_1^A, t_1^B\}$. Otherwise set $\Gamma^I(id) = \bot$ and stop.

(3) Once $\mathtt{TX}_{\mathsf{f}}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round $t_2 \leq t_1 + \Delta$, then $\Gamma^I(id) = \bot$ and return "offloaded".

Channels with validity

Party A

(1) Extract γ, tid_A and $\mathtt{TX}_{\mathsf{f}}^{\gamma}$ from $\Gamma^A(id)$. Let $id_\alpha := \gamma.\mathsf{subchan}(A)$ and send $(\mathtt{CLOSE}, id_\alpha) \overset{t_0}{\hookrightarrow} \mathcal{F}_{preL}$.

(2) If you receive $(\mathtt{CLOSED}, id_\alpha) \overset{t_1 \leq t_0+T+3\Delta}{\longleftarrow} \mathcal{F}_{preL}$, then post (post, $\mathtt{TX}_{\mathsf{f}}^{\gamma,\mathsf{A}}$) $\overset{t_2}{\hookrightarrow}$ $\widehat{\mathcal{L}}$. Otherwise, set $\Gamma^A(\gamma.\mathsf{id}) = \bot$ and stop.

(3) Once $\mathtt{TX}_{\mathsf{f}}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round $t_2 \leq t_1 + \Delta$, then update $\Gamma_L^A(id) := \Gamma^A(id)$, $\Gamma^A(id) := \bot$ and return "offloaded".

Party B

(1) Extract γ, tid_B and $[\mathtt{TX}_{\mathsf{f}}^{\gamma}]$ from $\Gamma^B(id)$. Let $id_\beta := \gamma.\mathsf{subchan}(B)$ and send $(\mathtt{CLOSE}, id_\beta) \overset{t_0}{\hookrightarrow} \mathcal{F}_{preL}$.

(2) If you receive $(\mathtt{CLOSED}, id_\beta) \overset{t_1 \leq t_0+T+3\Delta}{\longleftarrow} \mathcal{F}_{preL}$, then continue. Otherwise, set $\Gamma^B(\gamma.\mathsf{id}) = \bot$ and stop.

(3) Create the punishment transaction $\mathtt{TX}_{\mathsf{pun}}$ as $\mathtt{TX}_{\mathsf{pun}}.\mathsf{Input} := tid_B$, $\mathtt{TX}_{\mathsf{pun}}.\mathsf{Output} := (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \mathtt{One\text{-}Sig}_{pk_B})$ and set the value $\mathtt{TX}_{\mathsf{pun}}.\mathsf{Witness} := \mathsf{Sign}_{sk_B}([\mathtt{TX}_{\mathsf{pun}}])$. Then wait until round $t_2 := \max\{t_1, \gamma.\mathsf{val} + 2\Delta\}$ and send (post, $\mathtt{TX}_{\mathsf{pun}}$) $\overset{t_2}{\hookrightarrow}$ $\widehat{\mathcal{L}}$.

(4) Let $T_3 := t_2 + \Delta$ and distinguish the following two cases:
- A transaction with identifier $\mathtt{TX}_{\mathsf{f}}^{\gamma}.\mathsf{txid}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3 \leq T_3$, then define $\Gamma_L^B(id) := \Gamma^B(id)$ and set $m :=$ offloaded.
- The transaction $\mathtt{TX}_{\mathsf{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3 \leq T_3$, set $m :=$ punished.

(5) Set $\Gamma^B(id) := \bot$ and return m in round t_3.

Party I

(1) Extract γ, tid_A, tid_B, $\mathtt{TX}_{\mathsf{refund}}^{\gamma}$ and $[\mathtt{TX}_{\mathsf{f}}^{\gamma}]$ from $\Gamma^I(id)$. Then define $id_\alpha := \gamma.\mathsf{subchan}(A)$, $id_\beta := \gamma.\mathsf{subchan}(B)$ and send $(\mathtt{CLOSE}, id_\alpha) \overset{t_0}{\hookrightarrow} \mathcal{F}_{preL}$ and $(\mathtt{CLOSE}, id_\beta) \overset{t_0}{\hookrightarrow} \mathcal{F}_{preL}$.

(2) If you receive both messages $(\mathtt{CLOSED}, id_\alpha) \overset{t_1^A \leq t_0+T+3\Delta}{\longleftarrow} \mathcal{F}_{preL}$ and $(\mathtt{CLOSED}, id_\beta) \overset{t_1^B \leq t_0+T+3\Delta}{\longleftarrow} \mathcal{F}_{preL}$, then continue. Otherwise, set $\Gamma^I(\gamma.\mathsf{id}) = \bot$ and stop.

(3) Create the punishment transaction $\mathtt{TX}_{\mathsf{pun}}$ as $\mathtt{TX}_{\mathsf{pun}}.\mathsf{Input} := tid_A$, $\mathtt{TX}_{\mathsf{pun}}.\mathsf{Output} := (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \mathtt{One\text{-}Sig}_{pk_I})$ and set the value $\mathtt{TX}_{\mathsf{pun}}.\mathsf{Witness} := \mathsf{Sign}_{sk_I}([\mathtt{TX}_{\mathsf{pun}}])$. Then wait until round $t_2 := \max\{t_1^A, \gamma.\mathsf{val}\}$ and send (post, $\mathtt{TX}_{\mathsf{pun}}$) $\overset{t_2}{\hookrightarrow}$ $\widehat{\mathcal{L}}$.

(4) Let $T_3 := t_2 + \Delta$ and distinguish the following two cases:
- A transaction with identifier $\mathtt{TX}_{\mathsf{f}}^{\gamma}.\mathsf{txid}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3' \leq T_3$, send (post, $\mathtt{TX}_{\mathtt{refund}}^{\gamma}$) $\overset{t_4}{\hookrightarrow}$ $\widehat{\mathcal{L}}$ where $t_4 := \max\{t_1^B, t_3'\}$. Once $\mathtt{TX}_{\mathtt{refund}}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round $t_5 \leq t_4 + \Delta$, then define $m :=$ offloaded and $\Gamma^I(\gamma.\mathsf{id}) = \bot$.
- The transaction $\mathtt{TX}_{\mathsf{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3'' \leq T_3$, then define $m :=$ punished and $\Gamma^I(\gamma.\mathsf{id}) = \bot$.

(5) Return m in round t_6 where $t_6 := \max\{t_5, t_3''\}$.

D.5 Close

In order to close a virtual channel, parties first try to adjust the balances in the sunchannel according to the latest valid state of the virtual channel. This is done by updating the subchannels in a synchonous way, as was done during virtual channel creation. In case this process fails, parties close the channel forcefully. This means that parties first offload the channel and then immediately close the offloaded ledger channel.

Close a virtual channel

Below we abbreviate $\mathcal{F}_{preL} := \mathcal{F}_{preL}(T, 1)$, $A := \gamma.\mathsf{Alice}$ and $B := \gamma.\mathsf{Bob}$ and $I = \gamma.\mathsf{Ingrid}$. For $P \in \gamma.\mathsf{endUsers}$, we denote $Q := \gamma.\mathsf{otherParty}(P)$.

Party $P \in \{A, B\}$

Upon receiving $(\mathtt{CLOSE}, id) \overset{t_0^P}{\longleftarrow} \mathcal{E}$ or in round $t_0^P := \gamma.\mathsf{val} - (4\Delta + 7T)$ if $\gamma.\mathsf{val} \neq \bot$, proceed as follows:

(1) Extract γ, $\mathtt{TX}_{\mathsf{f}}^{\gamma}$ from $\Gamma^P(id)$.

(2) Parse $\gamma.\mathsf{st} = \left((c_P, \mathtt{One\text{-}Sig}_{pk_P}), (c_Q, \mathtt{One\text{-}Sig}_{pk_Q})\right)$.

(3) Compute the new state of the channel $id_\alpha := \gamma.\mathsf{subchan}(P)$ as

$$\vec{\theta}_P := \{(c_P, \mathtt{One\text{-}Sig}_{pk_P}), (c_Q + \frac{\gamma.\mathsf{fee}}{2}, \mathtt{One\text{-}Sig}_{pk_I})\}$$

Then, send $(\mathtt{UPDATE}, id_\alpha, \vec{\theta}_P, 0) \overset{t_0^P}{\hookrightarrow} \mathcal{F}_{preL}$.

(4) Upon $(\mathtt{SETUP}, id_\alpha, tid_P) \overset{t_1^P \leq t_0^P + T}{\longleftarrow} \mathcal{F}_{preL}$, send $(\mathtt{SETUP\text{-}OK}, id_\alpha)$ $\overset{t_1^P}{\hookrightarrow} \mathcal{F}_{preL}$.

Party I

Upon receiving $(\mathtt{CLOSE}, id) \overset{t_0^I}{\longleftarrow} \mathcal{E}$ or in round $t_0^I := \gamma.\mathsf{val} - (4\Delta + 7T)$, proceed as follows:

(1) Extract γ, $\mathtt{TX}_{\mathsf{f}}^{\gamma}$ from $\Gamma^I(id)$.

(2) Let $id_\alpha = \gamma.\mathsf{subchan}(A)$, $id_\beta = \gamma.\mathsf{subchan}(B)$ and $c := \gamma.\mathsf{cash}$

(3) If in round $t_1^I \le t_0^I + T$ you received both $(\texttt{UPDATE-REQ}, id_\alpha, tid_A, \vec{\theta}_A, 0) \hookrightarrow \mathcal{F}_{preL}$ and $(\texttt{UPDATE-REQ}, id_\beta, tid_B, \vec{\theta}_B, 0) \hookrightarrow \mathcal{F}_{preL}$ check that for some c_A, c_B s.t. $c_A + c_B = c$ it holds

$$\vec{\theta}_A = \{(c_A, \texttt{One-Sig}_{pk_A}), (c_B + \gamma.\texttt{fee}/2, \texttt{One-Sig}_{pk_I})\}$$

$$\vec{\theta}_B = \{(c_B, \texttt{One-Sig}_{pk_B}), (c_A + \gamma.\texttt{fee}/2, \texttt{One-Sig}_{pk_I})\}$$

If not, then stop.

(4) If in round $t_2^I \le t_1^I + T$ you receive both $(\texttt{SETUP-OK}, id_\alpha)) \hookrightarrow \mathcal{F}_{preL}$ and $(\texttt{SETUP-OK}, id_\beta)) \hookrightarrow \mathcal{F}_{preL}$, send $(\texttt{UPDATE-OK}, id_\alpha) \overset{t_2^I}{\hookrightarrow} \mathcal{F}_{preL}$ and $(\texttt{UPDATE-OK}, id_\beta) \overset{t_2^I}{\hookrightarrow} \mathcal{F}_{preL}$. If not, then stop.

Party $P \in \{A, B\}$

If you receive $(\texttt{UPDATE-OK}, id_\alpha) \overset{t_2^P \le t_1^P + 2T}{\longleftarrow} \mathcal{F}_{preL}$, reply with $(\texttt{REVOKE}, id_\alpha) \overset{t_2^P}{\hookrightarrow} \mathcal{F}_{preL}$. Otherwise execute $\texttt{Offload}^P(id)$ and stop.

Party I

If in round $t_3^I \le t_2^I + 2T$ you received both $(\texttt{REVOKE-REQ}, id_\alpha) \hookrightarrow \mathcal{F}_{preL}$ and $(\texttt{REVOKE-REQ}, id_\beta) \hookrightarrow \mathcal{F}_{preL}$, reply $(\texttt{REVOKE}, id_\alpha) \overset{t_3^I}{\hookrightarrow} \mathcal{F}_{preL}$ and $(\texttt{REVOKE}, id_\beta) \overset{t_3^I}{\hookrightarrow} \mathcal{F}_{preL}$ and set $\Gamma^I(id) := \bot$.

Party $P \in \{A, B\}$

If you receive $(\texttt{UPDATED}, id_\alpha) \overset{t_3^P \le t_2^P + 2T}{\longleftarrow} \mathcal{F}_{preL}$, set $\Gamma^P(id) := \bot$. Then output $(\texttt{CLOSED}, id) \overset{t_3^P}{\hookrightarrow} \mathcal{E}$ and stop. Else execute $\texttt{Offload}^P(id)$ and stop.

D.6 Punish

Finally, we formalize the actions taken by parties in every round. On a high level, in addition to triggering the hybrid ideal functionality to take the every-round actions for ledger channel (which include blockchain monitoring for outdated commit transactions), parties also need to make several check for virtual channel. Namely, channel users that tried to offload the virtual channel by closing their subchannel) monitor whether the other subchannel was closed as well. If yes, then they can publish the funding transaction and complete the offload and otherwise apply the punishment mechanism.

Punish virtual channel

Below we abbreviate $\mathcal{F}_{preL} := \mathcal{F}_{preL}(T, 1)$, $A := \gamma.\text{Alice}$ and $B := \gamma.\text{Bob}$ and $I = \gamma.\text{Ingrid}$. For $P \in \gamma.\text{endUsers}$, we denote $Q := \gamma.\text{otherParty}(P)$.

Upon receiving $(\texttt{PUNISH}) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$, do the following:

- Forward this message to the hybrid ideal functionality $(\texttt{PUNISH}) \overset{\tau_0}{\hookrightarrow} \mathcal{F}_{preL}$. If $(\texttt{PUNISHED}, id) \overset{\tau_1}{\hookleftarrow} \mathcal{F}_{preL}$, then $(\texttt{PUNISHED}, id) \overset{\tau_1}{\hookrightarrow} \mathcal{E}$.
- Execute both subprotocols Punish and Punish-Validity.

Punish

Party $P \in \{A, B\}$

For every $id \in \{0,1\}^*$, such that γ which $\gamma.\text{val} = \bot$ can be extracted from $\Gamma^P(id)$ do the following:

(1) Extract $\texttt{TX}_\text{f}^\gamma$ from $\Gamma^P(id)$ and tid_P, tid_Q from $\texttt{TX}_\text{f}^\gamma$. Check if tid_P appeared on $\widehat{\mathcal{L}}$. If not, then stop.

(2) Denote $T_2 := t_1 + T + 3\Delta$ and distinguish:
- If in round $t_2 \le T_2$ the transaction with tid_Q appeared on $\widehat{\mathcal{L}}$, then $(\text{post}, \texttt{TX}_\text{f}^\gamma) \overset{t_2}{\hookrightarrow} \widehat{\mathcal{L}}$.
- Else in round T_2 create the punishment transaction $\texttt{TX}_\text{pun}$ as

$$\texttt{TX}_\text{pun}.\text{Input} := tid_P$$
$$\texttt{TX}_\text{pun}.\text{Output} := (\gamma.\text{cash} + \gamma.\text{fee}/2, \texttt{One-Sig}_{pk_P})$$
$$\texttt{TX}_\text{pun}.\text{Witness} := \text{Sign}_{sk_P}([\texttt{TX}_\text{pun}]),$$

and $(\text{post}, \texttt{TX}_\text{pun}) \overset{T_2}{\hookrightarrow} \widehat{\mathcal{L}}$.

(3) Let $T_3 := t_2 + \Delta$ and distinguish the following two cases:
- The transaction $\texttt{TX}_\text{f}^\gamma$ was accepted by $\widehat{\mathcal{L}}$ in $t_3 \le T_3$, then $\Gamma_L^P(id) := \Gamma^P(id)$, $\Gamma^P(id) = \bot$ and $m := \texttt{OFFLOADED}$.
- The transaction $\texttt{TX}_\text{pun}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3 \le T_3$, then define $\Gamma^P(\gamma.\text{id}) = \bot$ and set $m := \texttt{PUNISHED}$.

(4) Output $(m, id) \overset{t_3}{\hookrightarrow} \mathcal{E}$.

Party I

For every $id \in \{0,1\}^*$, such that γ with $\gamma.\text{val} = \bot$ can be extracted from $\Gamma^I(id)$ do the following:

(1) Extract $\texttt{TX}_\text{f}^\gamma$ from $\Gamma^I(id)$ and tid_A, tid_B from $\texttt{TX}_\text{f}^\gamma$. Check if for some $P \in \{A, B\}$ a transaction with identifier tid_P appeared on $\widehat{\mathcal{L}}$. If not, then stop.

(2) Denote $id_\alpha := \gamma.\text{subchan}(Q)$ and send $(\texttt{CLOSE}, id_\alpha) \overset{t_0}{\hookrightarrow} \mathcal{F}_{preL}$.

(3) If you receive $(\texttt{CLOSED}, id_\alpha) \overset{t_1 \le t_0 + T + 3\Delta}{\longleftarrow} \mathcal{F}_{preL}$ and tid_Q appeared on $\widehat{\mathcal{L}}$, $(\text{post}, \texttt{TX}_\text{f}^\gamma) \overset{t_1}{\hookrightarrow} \widehat{\mathcal{L}}$. Otherwise set $\Gamma^I(id) = \bot$ and stop.

(4) Once $\texttt{TX}_\text{f}^\gamma$ is accepted by $\widehat{\mathcal{L}}$ in round t_2, such that $t_2 \le t_1 + \Delta$, set $\Gamma^I(id) = \bot$ and output $(\texttt{OFFLOADED}, id) \overset{t_2}{\hookrightarrow} \mathcal{E}$.

Punish-Validity

Party A

For every $id \in \{0,1\}^*$, such that γ with $\gamma.\text{val} \neq \bot$ can be extracted from $\Gamma^A(id)$ do the following:

(1) Extract $\texttt{TX}_\text{f}^\gamma$ from $\Gamma^A(id)$ and tid_A from $\texttt{TX}_\text{f}^\gamma$. If tid_A appeared on $\widehat{\mathcal{L}}$, then send $(\text{post}, \texttt{TX}_\text{f}^\gamma) \overset{t_1}{\hookrightarrow} \widehat{\mathcal{L}}$. Else stop.

(2) Once $\texttt{TX}_\text{f}^\gamma$ is accepted by $\widehat{\mathcal{L}}$ in round $t_2 \le t_1 + \Delta$, set $\Gamma_L^A(id) := \Gamma^A(id)$, $\Gamma^A(id) := \bot$ and output $(\texttt{OFFLOADED}, id) \overset{t_2}{\hookrightarrow} \mathcal{E}$.

Party B

For every $id \in \{0,1\}^*$, such that γ which $\gamma.\text{val} = \bot$ can be extracted from $\Gamma^B(id)$ do the following:

(1) Extract tid_B and $[\texttt{TX}_\text{f}^\gamma]$ from $\Gamma^B(id)$. Check if tid_B or $[\texttt{TX}_\text{f}^\gamma].\text{txid}$ appeared on $\widehat{\mathcal{L}}$. If not, then stop.

(2) If a transaction $\texttt{TX}_\text{f}^\gamma$ appeared on $\widehat{\mathcal{L}}$, update set $\Gamma_L^B(id) := \Gamma^B(id)$ and $\Gamma^B(id) := \bot$. Then output $(\texttt{OFFLOADED}, id) \overset{t_1}{\hookrightarrow} \mathcal{E}$ and stop.

(3) If tid_B appeared on $\widehat{\mathcal{L}}$, create the punishment transaction $\texttt{TX}_\text{pun}$ as

$$\texttt{TX}_\text{pun}.\text{Input} := tid_B$$
$$\texttt{TX}_\text{pun}.\text{Output} := (\gamma.\text{cash} + \gamma.\text{fee}/2, \texttt{One-Sig}_{pk_B})$$
$$\texttt{TX}_\text{pun}.\text{Witness} := \text{Sign}_{sk_B}([\texttt{TX}_\text{pun}]).$$

Then wait until round $t_2 := \max\{t_1, \gamma.\text{val} + 2\Delta\}$ and send (post, $\mathsf{TX}_{\mathsf{pun}}$) $\overset{t_2}{\hookrightarrow} \widehat{\mathcal{L}}$.

(4) If transaction $\mathsf{TX}_{\mathsf{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3 \leq t_2 + \Delta$, then define $\Gamma^B(\gamma.\text{id}) = \perp$ and output (PUNISHED, id) $\overset{t_3}{\hookrightarrow} \mathcal{E}$.

Party I

For every $id \in \{0,1\}^*$, such that γ which $\gamma.\text{val} = \perp$ can be extracted from $\Gamma^I(id)$ do the following:

(1) Extract tid_A, tid_B, $\mathsf{TX}^{\gamma}_{\mathsf{refund}}$ and $[\mathsf{TX}^{\gamma}_{\mathsf{f}}]$ from $\Gamma^I(id)$. Check if tid_A or tid_B appeared on $\widehat{\mathcal{L}}$ or $t_1 = \gamma.\text{val} - (3\Delta + T)$. If not, then stop.

(2) Distinguish the following cases:
- If $t_1 = \gamma.\text{val} - (3\Delta + T)$, define $id_\alpha := \gamma.\text{subchan}(A)$, $id_\beta := \gamma.\text{subchan}(B)$ and send (CLOSE, id_α) $\overset{t_1}{\hookrightarrow} \mathcal{F}_{preL}$ and (CLOSE, id_β) $\overset{t_1}{\hookrightarrow} \mathcal{F}_{preL}$.
- If tid_B appeared on $\widehat{\mathcal{L}}$, send (CLOSE, id_α) $\overset{t_1}{\hookrightarrow} \mathcal{F}_{preL}$.

(3) If a transaction with identifier tid_A appeared on $\widehat{\mathcal{L}}$ in round $t_2 \leq t_1 + T + 3\Delta$, create the punishment transaction $\mathsf{TX}_{\mathsf{pun}}$ as

$$\mathsf{TX}_{\mathsf{pun}}.\text{Input} := tid_A$$
$$\mathsf{TX}_{\mathsf{pun}}.\text{Output} := (\gamma.\text{cash} + \gamma.\text{fee}/2, \mathsf{One\text{-}Sig}_{pk_I})$$
$$\mathsf{TX}_{\mathsf{pun}}.\text{Witness} := \mathsf{Sign}_{sk_I}([\mathsf{TX}_{\mathsf{pun}}]).$$

Then wait until round $t_3 := \max\{t_2, \gamma.\text{val}\}$ and send (post, $\mathsf{TX}_{\mathsf{pun}}$) $\overset{t_3}{\hookrightarrow} \widehat{\mathcal{L}}$.

(4) Distinguish the following two cases:
- The transaction $\mathsf{TX}^{\gamma}_{\mathsf{f}}$.txid was accepted by $\widehat{\mathcal{L}}$ in $t_4 \leq t_3 + \Delta$, send (post, $\mathsf{TX}^{\gamma}_{\mathsf{refund}}$) $\overset{t_5}{\hookrightarrow} \widehat{\mathcal{L}}$ where $t_5 := \max\{\gamma.\text{val} + \Delta, t_4\}$. Once $\mathsf{TX}^{\gamma}_{\mathsf{refund}}$ is accepted by $\widehat{\mathcal{L}}$ in round $t_6 \leq t_5 + \Delta$, set $\Gamma^I(\gamma.\text{id}) = \perp$ and output (OFFLOADED, id) $\overset{t_6}{\hookrightarrow} \mathcal{E}$ and stop.
- The transaction $\mathsf{TX}_{\mathsf{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $t_4 \leq t_3 + \Delta$, then set $\Gamma^I(\gamma.\text{id}) = \perp$ and output (PUNISHED, id) $\overset{t_4}{\hookrightarrow} \mathcal{E}$.

E SIMPLIFYING FUNCTIONALITY DESCRIPTION

In order to simplify the exposition, the formal descriptions of the channel ideal functionalities $\mathcal{F}_L$, $\mathcal{F}_{preL}$ and $\mathcal{F}_V$ are simplified. Namely, they exclude several natural checks that one would expect an ideal functionality to make when it receives a message from a party. The purpose of the checks is to avoid the functionality from accepting malformed messages. To provide some intuition, we present several examples of such restrictions:

- A party sends a malformed message (e.g. missing or additional parameters)
- A party request creation of a virtual channel but one of the two subchannels does not exists or does not have enough funds for virtual channel creation.
- Parties try to update the same channel twice in parallel.

We now list all check formally in the wrapper below which can be seen as an extension to the wrapper provided by [4] for $\mathcal{F}_L$.

Functionality wrapper: $\mathcal{W}_{\text{checks}}(T)$

The wrapper is defined for $\mathcal{F} \in \{\mathcal{F}_V(T), \mathcal{F}_{preL}(T), \mathcal{F}_L(T)\}$. Below, we abbreviate $A := \gamma.\text{Alice}$, $B := \gamma.\text{Bob}$ and $I := \gamma.\text{Ingrid}$.

Create: Upon (CREATE, γ, tid) $\overset{\tau_0}{\hookleftarrow} P$, where $P \in \gamma.\text{users}$, check if: $\Gamma(\gamma.\text{id}) = \perp$, $\mathcal{F}.\Gamma_{pre}(\gamma.\text{id}) = \perp$ and there is no channel γ' with $\gamma.\text{id} = \gamma'.\text{id}$ being created or pre-created; γ is valid according to the definition given in Section 3.1; $\gamma.\text{st} = \{(c_P, \mathsf{One\text{-}Sig}_{pk_P}), (c_Q, \mathsf{One\text{-}Sig}_{pk_Q})\}$ for $c_P, c_Q \in \mathbb{R}^{\geq 0}$. Depending on the type of channel, make the following additional checks:

ledger channel: There exists $(t, id, i, \theta) \in \widehat{\mathcal{L}}.\text{UTXO}$ such that $\theta = (c_P, \mathsf{One\text{-}Sig}_P)$ for $(id, i) := tid$;[a]

virtual channel:
- If $P \in \gamma.\text{endUsers}$, then $\alpha := \mathcal{F}.\Gamma(id_P) \neq \perp$ for $id_P := \gamma.\text{subchan}(P)$; $\alpha.\text{endUsers} = \{P, I\}$; there is no other virtual channel being created over α and α is currently not being updated; both P and I have enough funds in α.
- If $P = I$, then $\alpha := \mathcal{F}.\Gamma(id_A) \neq \perp$ for $id_A := \gamma.\text{subchan}(A)$; $\beta := \mathcal{F}.\Gamma(id_B) \neq \perp$ for $id_B := \gamma.\text{subchan}(B)$; $\alpha.\text{endUsers} = \{A, I\}$; $\beta.\text{endUsers} = \{B, I\}$; there is no other virtual channel being created over α or β; A and I have enough funds in α and B and I have enough funds in β.
- If $\gamma.\text{val} \neq \perp$, then $\gamma.\text{val} \geq \tau_0 + 4\Delta + 15T$.

If one of the above checks fails, drop the message. Else proceed as $\mathcal{F}$.

Pre-Create: Upon (PRE−CREATE, γ, TX_{f}, i, t_{ofl}) $\overset{\tau_0}{\hookleftarrow} P$, check if: $P \in \gamma.\text{users}$, $\mathcal{F}.\Gamma_{pre}(\gamma.\text{id}) = \perp$, $\mathcal{F}.\Gamma_{pre}(\gamma.\text{id}) = \perp$ and there is no channel γ' with $\gamma.\text{id} = \gamma'.\text{id}$ being created or pre-created; γ is valid according to the definition given in Section 3.1; $\gamma.\text{st} = \{(c_P, \mathsf{One\text{-}Sig}_{pk_P}), (c_Q, \mathsf{One\text{-}Sig}_{pk_Q})\}$ for $c_P, c_Q \in \mathbb{R}^{\geq 0}$ and TX_{f} is not a published transaction on $\widehat{\mathcal{L}}$. If one of the above checks fails, drop the message. Else proceed as $\mathcal{F}$.

(Pre)-Update: Upon (m, id, $\vec{\theta}$, t_{stp}) $\overset{\tau_0}{\hookleftarrow} P$, check if: $\gamma := \Gamma(id) \neq \perp$ if $m = \text{UPDATE}$ and $\gamma := \Gamma_{pre}(id) \neq \perp$ if $m = \text{PRE−UPDATE}$. In both cases additionally check: $P \in \gamma.\text{endUsers}$; there is no other update being preformed on γ; let $\vec{\theta} = (\theta_1, \ldots \theta_\ell) = ((c_1, \varphi_1), \ldots, (c_\ell, \varphi_\ell))$, then $\sum_{j \in [\ell]} c_i = \gamma.\text{cash}$ and $\varphi_j \in \widehat{\mathcal{L}}.\mathcal{V}$ for each $j \in [\ell]$. If not, drop the message. Else proceed as $\mathcal{F}$.

Upon ((PRE−)SETUP−OK, id) $\overset{\tau_2}{\hookleftarrow} P$ check if: you accepted a message ((PRE−)UPDATE, id, $\vec{\theta}$, t_{stp}) $\overset{\tau_0}{\hookleftarrow} P$, where $t_2 - t_0 \leq t_{\text{stp}} + T$ and the message is a reply to the message ((PRE−)SETUP, id, tid) sent to P in round τ_1 such that $\tau_2 - \tau_1 \leq t_{\text{stp}}$[b]. If not, drop the message. Else proceed as $\mathcal{F}$.

Upon ((PRE−)UPDATE−OK, id) $\overset{\tau_0}{\hookleftarrow} P$, check if the message is a reply to the message ((PRE−)SETUP−OK, id) sent to P in round τ_0. If not, drop the message. Else proceed as $\mathcal{F}$.

Upon ((PRE−)REVOKE, id) $\overset{\tau_0}{\hookleftarrow} P$, check if the message is a reply to either the message ((PRE−)UPDATE−OK, id) sent to P in round τ_0 or the message ((PRE−)REVOKE−REQ, id) sent to P in round τ_0. If not, drop the message. Else proceed as $\mathcal{F}$.

Offload: Upon receiving (OFFLOAD, id) $\overset{\tau_0}{\hookleftarrow} P$ make the following checks: $\gamma := \Gamma(id) \neq \perp$ is a virtual channel and $P \in \gamma.\text{users}$. If one of the checks fails, then drop the message. Otherwise proceed as the functionality $\mathcal{F}$.

Close: Upon (CLOSE, id) $\overset{\tau_0}{\hookleftarrow} P$, check if $\gamma := \Gamma(id) \neq \perp$ and $P \in \gamma.\text{endUsers}$. If γ is a virtual channel, additionally check that $\gamma.\text{val} = \perp$. If not, drop the message. Else proceed as $\mathcal{F}$.

All other messages are dropped.

[a] In case more channels are being created at the same time, then none of the other creation requests can use of the tid.

[b] See Appendix B what we formally meant by "reply".

F SIMPLIFYING THE PROTOCOL DESCRIPTIONS

Similarly as the descriptions of our ideal functionality, the description our channel protocols, the protocol Π_{preL} presented in Appendix C.4 and the protocol Π_V presented in Appendix D, exclude many natural checks that we would want an honest party to make. Let us give a few examples of requests which an honest party drops if received from the environment:

- The environment sends a malformed message to a party P (e.g. missing or additional parameters);
- A party P receives an instruction to create a channel γ but $P \notin \gamma.\mathsf{endUsers}$;
- A party P receives an instruction to create a virtual channel on top of a ledger channel that does not exist, does not belong to part P or is not sufficiently funded.
- Parties request to create a channel with validity whose validity time already expired (or is about to expire).

We define all these check as a wrapper $\mathcal{W}_{\text{checksP}}$ that can be seen as a extension of the wrapper provided by [4] for their ledger channel protocol.

Protocol wrapper: $\mathcal{W}_{\text{checksP}}$

The wrapper is defined for $\Pi \in \{\Pi_V(T), \Pi_{preL}(T)\}$. Below, we abbreviate $A := \gamma.\mathsf{Alice}$, $B := \gamma.\mathsf{Bob}$ and $I := \gamma.\mathsf{Ingrid}$.

Party P

Create: Upon $(\mathtt{CREATE}, \gamma, tid) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$ check if: $P \in \gamma.\mathsf{endUsers}$; $\Gamma^P(\gamma.\mathsf{id}) = \bot$, $\Gamma^P_{pre}(\gamma.\mathsf{id}) = \bot$ and there is no channel γ' with $\gamma.\mathsf{id} = \gamma'.\mathsf{id}$ being created or pre-created; γ is valid according to the definition given in Section 3.1; $\gamma.\mathsf{st} = \{(c_P, \mathtt{One\text{-}Sig}_{pk_P}), (c_Q, \mathtt{One\text{-}Sig}_{pk_Q})\}$ for $c_P, c_Q \in \mathbb{R}^{\geq 0}$. Depending on the type of channel, make the following additional checks:

ledger channel: There exists $(t, id, i, \theta) \in \widehat{\mathcal{L}}.\mathtt{UTXO}$ such that $\theta = (c_P, \mathtt{One\text{-}Sig}_P)$ for $(id, i) := tid$;[a]

virtual channel:

- If $P \in \gamma.\mathsf{endUsers}$, then $\alpha := \Gamma^P(id_P) \neq \bot$ for $id_P := \gamma.\mathsf{subchan}(P)$; $\alpha.\mathsf{endUsers} = \{P, I\}$; there is no other virtual channel being created over α and α is currently not being updated; both P and I have enough funds in α.
- If $P = I$, then $\alpha := \Gamma^P(id_A) \neq \bot$ for $id_A := \gamma.\mathsf{subchan}(A)$; $\beta := \Gamma^P(id_B) \neq \bot$ for $id_B := \gamma.\mathsf{subchan}(B)$; $\alpha.\mathsf{endUsers} = \{A, I\}$; $\beta.\mathsf{endUsers} = \{B, I\}$; there is no other virtual channel being created over α or β; A and I have enough funds in α and B and I have enough funds in β.
- If $\gamma.\mathsf{val} \neq \bot$, then $\gamma.\mathsf{val} \geq \tau_0 + 4\Delta + 15T$.

If one of the checks fails, drop the message. Else proceed as in Π.

Pre-Create: Upon $(\mathtt{PRE\text{-}CREATE}, \gamma, \mathtt{TX_f}, i, t_{ofl}) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$, check if: $P \in \gamma.\mathsf{users}$, $\Gamma^P_{pre}(\gamma.\mathsf{id}) = \bot$, $\Gamma^P_{pre}(\gamma.\mathsf{id}) = \bot$ and there is no channel γ' with $\gamma.\mathsf{id} = \gamma'.\mathsf{id}$ being created or pre-created; γ is valid according to the definition given in Section 3.1; $\gamma.\mathsf{st} = \{(c_P, \mathtt{One\text{-}Sig}_{pk_P}), (c_Q, \mathtt{One\text{-}Sig}_{pk_Q})\}$ for $c_P, c_Q \in \mathbb{R}^{\geq 0}$ and $\mathtt{TX_f}$ is not a published transaction on $\widehat{\mathcal{L}}$. If one of the above checks fails, drop the message. Else proceed as in Π.

(Pre)-Update: Upon $(m, id, \vec{\theta}, t_{\text{stp}}) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$ check if: $\gamma := \Gamma^P(id) \neq \bot$ if $m = \mathtt{UPDATE}$ and $\gamma := \Gamma^P_{pre}(id) \neq \bot$ if $m = \mathtt{PRE\text{-}UPDATE}$. In both cases, check that there is no other update being preformed on γ; let $\vec{\theta} = (\theta_1, \ldots \theta_\ell) = ((c_1, \varphi_1), \ldots, (c_\ell, \varphi_\ell))$, then $\sum_{j\in[\ell]} c_i = \gamma.\mathsf{cash}$ and $\varphi_j \in \widehat{\mathcal{L}}.\mathcal{V}$ for each $j \in [\ell]$. If on of the checks fails, drop the message. Else proceed as in Π.

Upon $((\mathtt{PRE\text{-}})\mathtt{SETUP\text{-}OK}, id) \overset{\tau_2}{\hookleftarrow} \mathcal{E}$ check if: you accepted a message $((\mathtt{PRE\text{-}})\mathtt{UPDATE}, id, \vec{\theta}, t_{\text{stp}}) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$, where $t_2 - t_0 \leq t_{\text{stp}} + T$ and the message is a reply to the message $((\mathtt{PRE\text{-}})\mathtt{SETUP}, id, \vec{tid})$ you sent in round τ_1 such that $\tau_2 - \tau_1 \leq t_{\text{stp}}$[b]. If not, drop the message. Else proceed as in Π.

Upon $((\mathtt{PRE\text{-}})\mathtt{UPDATE\text{-}OK}, id) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$, check if the message is a reply to the message $((\mathtt{PRE\text{-}})\mathtt{SETUP\text{-}OK}, id)$ you sent in round τ_0. If not, drop the message. Else proceed as in Π.

Upon $((\mathtt{PRE\text{-}})\mathtt{REVOKE}, id) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$, check if the message is a reply to either $((\mathtt{PRE\text{-}})\mathtt{UPDATE\text{-}OK}, id)$ or $((\mathtt{PRE\text{-}})\mathtt{REVOKE\text{-}REQ}, id)$ you sent in round τ_0. If not, drop the message. Else proceed as in Π.

Offload: Upon receiving $(\mathtt{OFFLOAD}, id) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$ make the following checks: $\gamma := \Gamma(id) \neq \bot$ is a virtual channel and $P \in \gamma.\mathsf{users}$. If one of the checks fails, then drop the message. Else proceed as in Π.

Close: Upon $(\mathtt{CLOSE}, id) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$, check if $\gamma := \Gamma^P(id) \neq \bot$, $P \in \gamma.\mathsf{endUsers}$. If γ is a virtual channel, additionally check that $\gamma.\mathsf{val} = \bot$. If not, drop the message. Else proceed as in Π.

All other messages are dropped.

[a] In case more channels are being created at the same time, then none of the other creation requests can use of the *tid*.

[b] See Appendix B what we formally meant by "reply".

G SIMULATION WRAPPER

In this section we provide a proof for Theorem 2. In our proof, we provide the code for a simulator, that simulates the protocol Π_{preL} in the ideal world, having access to the functionalities $\widehat{\mathcal{L}}$ and $\mathcal{F}_{preL}$. In UC proofs it is required to provide a simulation of the real protocol in the ideal world even without knowledge of the secret inputs of the honest protocol participants. The main challenge is that this transcript of the simulation has to be indistinguishable to the environment $\mathcal{E}$ from the transcript of the real protocol execution. Yet, in our protocols, parties do not receive secret inputs, but are only instructed by the environment to take certain protocol actions, e.g. updating a channel. Hence the only challenge that arises during simulation is handling different behavior of malicious parties. Due to this, we only provide the simulator code for the protocol without arguing about indistinguishability of simulation and real protocol execution, since it naturally holds due to the reasons given above. In our simulation, we omit the case where all parties are honest, since the simulator simply has to follow the protocol description. In case of three protocol participants, we provide a simulation for all cases where two parties are corrupted and one party is honest, because these cases cover also all cases where just one party is corrupted. In other words, the case where two parties are honest is a combination of cases where each of these parties are honest individually.

Since the functionality $\mathcal{F}_{preL}$ incorporates, $\mathcal{F}_L$, we refer at some point of our simulation to the simulator code for ledger channels.

We note that the indistinguishability of the simulated transcript and the transcript of the real protocol can only hold if the security properties of the underlying adaptor signature scheme holds.

Namely, we require the adaptor signature scheme to fulfill the unforgeability, witness extractability and adaptability properties.

Simulator for Wrapper protocol

Pre-Creat

Case A honest and B corrupted

(1) Upon A sending $(\texttt{PRE-CREATE}, \gamma, \texttt{TX}_\texttt{f}, i, t_{ofl}) \xhookrightarrow{\tau_0} \mathcal{F}_{preL}$ set $T_1 = 2$ and do the following:
(2) If $\texttt{TX}_\texttt{f}.\text{Output}[i].\text{cash} \neq \gamma.\text{cash}$, then ignore the message.
(3) Set $id := \gamma.\text{id}$, generate $(R_A, r_A) \leftarrow \text{GenR}$, $(Y_A, y_A) \leftarrow \text{GenR}$ and send $(\text{createInfo}, id, \texttt{TX}_\texttt{f}, i, t_{ofl}, R_A, Y_A) \xhookrightarrow{\tau_0} B$.
(4) If $(\text{createInfo}, id, \texttt{TX}_\texttt{f}, i, t_{ofl}, R_B, Y_B) \xhookleftarrow{\tau_0+1} B$, create:

$$[\texttt{TX}_\texttt{c}] := \texttt{GenCommit}([\texttt{TX}_\texttt{f}], I_A, I_B, 0)$$
$$[\texttt{TX}_\texttt{s}] := \texttt{GenSplit}([\texttt{TX}_\texttt{c}].\text{txid}\|1, \gamma.\text{st})$$

for $I_A := (pk_A, R_B, Y_A)$, $I_B := (pk_B, R_B, Y_B)$. Else stop.
(5) Compute $s_\text{c}^A \leftarrow \text{pSign}_{sk_A}([\texttt{TX}_\texttt{c}], Y_B)$, $s_\text{s}^A \leftarrow \text{Sign}_{sk_A}([\texttt{TX}_\texttt{s}])$ and send $(\text{createCom}, id, s_\text{c}^A, s_\text{s}^A) \xhookrightarrow{\tau_0+1} B$.
(6) If $(\text{createCom}, id, s_\text{c}^B, s_\text{s}^B) \xhookleftarrow{\tau_0+2} B$, s.t. $\text{pVrfy}_{pk_B}([\texttt{TX}_\texttt{c}], Y_A; s_\text{c}^B) = 1$ and $\text{Vrfy}_{pk_B}([\texttt{TX}_\texttt{s}]; s_\text{s}^B) = 1$, set

$$\texttt{TX}_\texttt{c} := ([\texttt{TX}_\texttt{c}], \{\text{Sign}_{sk_A}([\texttt{TX}_\texttt{c}]), \text{Adapt}(s_\text{c}^B, y_A)\})$$
$$\texttt{TX}_\texttt{s} := ([\texttt{TX}_\texttt{s}], \{s_\text{s}^A, s_\text{s}^B\})$$
$$\Gamma_{pre}^A(\gamma.\text{id}) := (\gamma, \texttt{TX}_\texttt{f}, (\texttt{TX}_\texttt{c}, r_A, R_B, Y_B, s_\text{c}^A), \texttt{TX}_\texttt{s}, t_{ofl}).$$

and if B has not sent $(\texttt{PRE-CREATE}, \gamma, \texttt{TX}_\texttt{f}, i, t_{ofl})$ to $\mathcal{F}_{preL}$ send this message on behalf of B.

Pre-Update

Let $T_1 = 2$ and $T_2 = 1$ and let $|\vec{tid}| = 1$.

Case A is honest and B is corrupted

Upon A sending $(\texttt{PRE-UPDATE}, id, \vec{\theta}, t_\text{stp}) \xhookrightarrow{\tau_0} \mathcal{F}_{preL}$, proceed as follows:

(1) Generate new revocation public/secret pair $(R_P, r_P) \leftarrow \text{GenR}$ and a new publishing public/secret pair $(Y_P, y_P) \leftarrow \text{GenR}$ and send $(\text{updateReq}, id, \vec{\theta}, t_\text{stp}, R_A, Y_A) \xhookrightarrow{\tau_0^A} B$.
(2) Upon $(\text{updateInfo}, id, h_B, Y_B, s_\text{s}^B) \xhookleftarrow{\tau_0^A+2} B$, set $t_\text{lock} := \tau_0^A + t_\text{stp} + 5 + \Delta + t_{ofl}$, extract $\texttt{TX}_\texttt{f}$ from $\Gamma_{pre}^B(id)$ and

$$[\texttt{TX}_\texttt{c}] := \texttt{GenCommit}([\texttt{TX}_\texttt{f}], I_A, I_B, t_\text{lock})$$
$$[\texttt{TX}_\texttt{s}] := \texttt{GenSplit}([\texttt{TX}_\texttt{c}].\text{txid}\|1, \vec{\theta}),$$

for $I_A := (pk_A, R_A, Y_A)$ and $I_B := (pk_B, R_B, Y_B)$. If it holds that $\text{Vrfy}_{pk_B}([\texttt{TX}_\texttt{s}]; s_\text{s}^B) = 1$ continue. Else mark this execution as "failed" and stop.
(3) If A sends $(\texttt{PRE-SETUP-OK}, id) \xhookrightarrow{\tau_1^A \leq \tau_0^A+2+t_\text{stp}} \mathcal{F}_{preL}$, compute $s_\text{c}^A \leftarrow \text{pSign}_{sk_A}([\texttt{TX}_\texttt{c}], Y_B)$, $s_\text{s}^A \leftarrow \text{Sign}_{sk_A}([\texttt{TX}_\texttt{s}])$ and send the message $(\text{update-commitA}, id, s_\text{c}^A, s_\text{s}^A) \xhookrightarrow{\tau_1^A} B$.
(4) In round $\tau_1^A + 2$ distinguish the following cases:

- If A receives $(\text{update-commitB}, id, s_\text{c}^B) \xhookleftarrow{\tau_1^A+2} B$ check if B has not sent $(\texttt{PRE-UPDATE-OK}, id) \xhookrightarrow{\tau_1^A+1} \mathcal{F}_{preL}$. If so send the message $(\texttt{PRE-UPDATE-OK}, id) \xhookrightarrow{\tau_1^A+1} \mathcal{F}_{preL}$ on behalf of B. If $\text{pVrfy}_{pk_B}([\texttt{TX}_\texttt{c}], Y_A; s_\text{c}^B) = 0$, then mark this execution as "failed" and stop.
- If A receives $(\text{updateNotOk}, id, r_B) \xhookleftarrow{\tau_1^A+2} B$, where $(R_B, r_B) \in R$, add $\Theta^A(id) := \Theta^A(id) \cup ([\texttt{TX}_\texttt{c}], r_B, Y_B, s_\text{c}^A)$, instruct $\mathcal{F}_{preL}$ to send $(\texttt{PRE-UPDATE-REJECT}, id) \hookrightarrow A$ and to stop and mark this execution as "failed" and stop.
- Else, execute the simulator code for the procedure $\texttt{Wait-if-Register}^A(id)$ and stop.

(5) If A sends $(\texttt{PRE-REVOKE}, id) \xhookrightarrow{\tau_1^A+2} \mathcal{F}_{preL}$, then parse $\Gamma_{pre}^A(id)$ as $(\gamma, \texttt{TX}_\texttt{f}, (\overline{\texttt{TX}}_\texttt{c}, \bar{r}_A, \bar{R}_B, \bar{Y}_B, \bar{s}_{\text{Com}}^A), \overline{\texttt{TX}}_\texttt{s})$ and update the channel space as $\Gamma_{pre}^A(id) := (\gamma, \texttt{TX}_\texttt{f}, (\texttt{TX}_\texttt{c}, r_A, R_B, Y_B, s_\text{c}^A), \texttt{TX}_\texttt{s})$, for $\texttt{TX}_\texttt{s} := ([\texttt{TX}_\texttt{s}], \{s_\text{s}^A, s_\text{s}^B\})$ and $\texttt{TX}_\texttt{c} := ([\texttt{TX}_\texttt{c}], \{\text{Sign}_{sk_A}([\texttt{TX}_\texttt{c}]), \text{Adapt}(s_\text{c}^B, y_A)\})$. Then send $(\text{revokeP}, id, \bar{r}_A) \xhookrightarrow{\tau_1^A+2} B$. Else, execute the simulator code for the procedure $\texttt{Wait-if-Register}^A(id)$ and stop.
(6) If A receives $(\text{revokeB}, id, \bar{r}_B) \xhookleftarrow{\tau_1^A+4} B$, check if B has not sent $(\texttt{PRE-REVOKE}, id) \xhookrightarrow{\tau_1^B+2} \mathcal{F}_{preL}$. If so send $(\texttt{PRE-REVOKE}, id) \xhookrightarrow{\tau_1^B+2} \mathcal{F}_L$ on behalf of B. Check if $(\bar{R}_B, \bar{r}_B) \in R$, then set

$$\Theta^B(id) := \Theta^A(id) \cup ([\overline{\texttt{TX}}_\texttt{c}], \bar{r}_B, \bar{Y}_B, \bar{s}_{\text{Com}}^A)$$

Else execute the simulator code for the procedure $\texttt{Wait-if-Register}^A(id)$ and stop.

Case B is honest and A is corrupted

Upon A sending $(\text{updateReq}, id, \vec{\theta}, t_\text{stp}, h_A) \xhookrightarrow{\tau_0} B$, send the message $(\texttt{PRE-UPDATE}, id, \vec{\theta}, t_\text{stp}) \xhookrightarrow{\tau_0} \mathcal{F}_{preL}$ on behalf of A, if A has not already sent this message. Proceed as follows:

(1) Upon $(\text{updateReq}, id, \vec{\theta}, t_\text{stp}, R_A, Y_A) \xhookleftarrow{\tau_0^B} A$, generate $(R_B, r_B) \leftarrow \text{GenR}$ and $(Y_B, y_B) \leftarrow \text{GenR}$.
(2) Set $t_\text{lock} := \tau_0^B + t_\text{stp} + 4 + \Delta + t_{ofl}$, extract $\texttt{TX}_\texttt{f}$ from $\Gamma_{pre}^A(id)$ and

$$[\texttt{TX}_\texttt{c}] := \texttt{GenCommit}([\texttt{TX}_\texttt{f}], I_A, I_B, t_\text{lock})$$
$$[\texttt{TX}_\texttt{s}] := \texttt{GenSplit}([\texttt{TX}_\texttt{c}].\text{txid}\|1, \vec{\theta})$$

where $I_A := (pk_A, R_A, Y_A)$, $I_B := (pk_B, R_B, Y_B)$.
(3) Compute $s_\text{s}^B \leftarrow \text{Sign}_{sk_B}([\texttt{TX}_\texttt{s}])$, send $(\text{updateInfo}, id, R_B, Y_B, s_\text{s}^B) \xhookrightarrow{\tau_0^B} A$.
(4) If B receives $(\text{update-commitA}, id, s_\text{c}^A, s_\text{s}^A) \xhookleftarrow{\tau_1^B \leq \tau_0^B+2+t_\text{stp}} A$ then send $(\texttt{PRE-SETUP-OK}, id) \xhookrightarrow{\tau_1^B} \mathcal{F}_{preL}$ on behalf of A, if A has not sent this message.
(5) Check if $\text{pVrfy}_{pk_P}([\texttt{TX}_\texttt{c}], Y_Q; s_\text{c}^P) = 1$ and $\text{Vrfy}_{pk_P}([\texttt{TX}_\texttt{s}]; s_\text{s}^P) = 1$. Else mark this execution as "failed" and stop.
(6) If B sends $(\texttt{PRE-UPDATE-OK}, id) \xhookrightarrow{\tau_1^B} \mathcal{F}_{preL}$, then compute $s_\text{c}^B \leftarrow \text{pSign}([\texttt{TX}_\texttt{c}], Y_A)$ and send $(\text{update-commitB}, id, s_\text{c}^B) \xhookrightarrow{\tau_1^B} A$. Else send $(\text{updateNotOk}, id, r_B) \xhookrightarrow{\tau_1^B} A$, mark this execution as "failed" and stop.

(7) Parse $\Gamma^B_{pre}(id)$ as $(\gamma, \mathtt{TX_f}, (\overline{\mathtt{TX}}_\mathtt{c}, \bar{r}_B, \bar{R}_A, \bar{Y}_A, \bar{s}^B_{\mathtt{Com}}), \overline{\mathtt{TX}}_\mathtt{s})$. If B receives (revokeA, id, $\bar{r}_A$) $\xleftarrow{\tau_1^B+2}$ A, send (PRE−REVOKE, id) $\xleftarrow{\tau_1^B+2}$ $\mathcal{F}_{preL}$ on behalf of A, if A has not sent this message.

Else if you do not receive (revokeA, id, $\bar{r}_A$) $\xleftarrow{\tau_1^B+2}$ A or if $(\bar{R}_A, \bar{r}_A) \notin R$, execute the simulator code of the procedure $\mathtt{Wait\text{−}if\text{−}Register}^B(id)$ and stop.

(8) If B sends (PRE−REVOKE, id) $\xleftarrow{\tau_1^B+2}$ $\mathcal{F}_{preL}$, then set

$$\Theta^B(id) := \Theta^B(id) \cup ([\overline{\mathtt{TX}}_\mathtt{c}], \bar{r}_A, \bar{Y}_A, \bar{s}^B_{\mathtt{Com}})$$

$$\Gamma^B_{pre}(id) := (\gamma, \mathtt{TX_f}, (\mathtt{TX_c}, r_B, R_A, Y_A, s^B_\mathtt{c}), \mathtt{TX_s}),$$

for $\mathtt{TX_s} := ([\mathtt{TX_s}], \{s^A_\mathtt{s}, s^B_\mathtt{s}\})$ and $\mathtt{TX_c} := ([\mathtt{TX_c}], \{\mathrm{Sign}_{sk_B}([\mathtt{TX_c}]), \mathrm{Adapt}(s^A_\mathtt{c}, y_B)\})$. Then (revokeB, id, $\bar{r}_B$) $\xleftarrow{\tau_1^B+2}$ A and stop. Else, in round $\tau_1^B + 2$, execute the simulator code of the procedure $\mathtt{Wait\text{−}if\text{−}Register}^B(id)$ and stop.

Register

Case A honest and B corrupted

For party A in every round τ_0 do the following:

(1) For each $id \in \{0,1\}^*$ s.t. $\Gamma^A_{pre}(id) \neq \perp$:
(2) Parse $\Gamma^A_{pre}(id) := (\gamma, \mathtt{TX_f}, (\mathtt{TX_c}, r_A, R_B, Y_B, s^A_\mathtt{c}), \mathtt{TX_s}, t_{ofl}, x)$
(3) If $\mathtt{TX_f}$ appeared on-chain in this round, then
  (a) Set $\Gamma(id) := (\gamma, \mathtt{TX_f}, (\mathtt{TX_c}, r_A, R_B, Y_B, s^A_\mathtt{c}), \mathtt{TX_s})$.
  (b) Set $\Gamma^A_{pre}(id) := \perp$
  (c) If $x = \mathtt{in\text{−}dispute}$, then execute the simulator code for $\mathtt{L\text{−}ForceClose}^A(id)$.

$\mathtt{Wait\text{−}if\text{−}Register}(id)$

Case A honest and B corrupted

Let τ_0 be the current round. Let $X := \Gamma^A_{pre}(id)$. Then set $\Gamma^A_{pre}(id) := (X, \mathtt{in\text{−}dispute})$.

H SIMULATION VIRTUAL CHANNELS

In this section we provide a proof for Theorem 1. In our proof, we provide the code for a simulator, that simulates the protocol Π_V in the ideal world having access to the functionalities $\widehat{\mathcal{L}}$ and $\mathcal{F}_V$.

We note that since during our simulation, no ERROR messages are produced by the functionality, the protocol satisfies the security properties of the functionality $\mathcal{F}_V$ as mentioned in Section 3.2.

Simulator for creating virtual channels

creating virtual channels

Case A is honest and I are B are corrupt

Upon A sending (CREATE, γ) $\xleftarrow{\tau_0^A}$ $\mathcal{F}_V$ set $T_1 = 6T + t_{\mathrm{stp}}$ proceed as follows:

(1) Let $id_\alpha := \gamma.\mathrm{subchan}(A)$ and compute

$$\theta_A := \mathtt{GenVChannelOutput}(\gamma, A).$$

(2) Upon A sending (UPDATE, id_α, θ_A, t_{stp}) $\xleftarrow{\tau_0^A}$ $\mathcal{F}_L$ execute the simulator code of the update procedure for the generalized channels until the message (SETUP, id_α, tid_A) is sent by $\mathcal{F}_L$. If the execution stops send (peaceful−reject, id_α) $\hookrightarrow$ $\mathcal{F}_V$.
(3) Upon A receiving (SETUP, id_α, tid_A) $\xleftarrow{\tau_1^A \leq \tau_0^A + T}$ $\mathcal{F}_L$, execute the simulator code for SetupVChannel with input (γ, tid_A).
(4) If this execution of SetupVChannel is recorded "failed" stop. Otherwise execute the simulator code of the update procedure for the generalized channels until the end. If the execution failed (I does not revoke) instruct $\mathcal{F}_V$ to $\mathtt{L\text{−}ForceClose}(id_\alpha)$.
(5) If B or I have not sent (CREATE, γ) $\hookrightarrow$ $\mathcal{F}_V$ send this message on their behalf.
(6) Upon A receiving (CREATED, γ) $\xleftarrow{\tau_2^A \leq \tau_1^A + 5T}$ $\mathcal{F}_V$, mark γ as created, i.e. update $\Gamma^A(\gamma.\mathrm{id})$ from $(\perp, x)$ to (γ, x).

Case I is honest and A, B are corrupted

Upon I sending (CREATE, γ) $\xleftarrow{\tau_0^I}$ $\mathcal{F}_V$ proceed as follows:

(1) Set $id_\alpha = \gamma.\mathrm{subchan}(A)$, $id_\beta = \gamma.\mathrm{subchan}(B)$ and generate

$$\theta_A := \mathtt{GenVChannelOutput}(\gamma, A)$$

$$\theta_B := \mathtt{GenVChannelOutput}(\gamma, B)$$

(2) Upon A and B sending (UPDATE, id_α, θ_A, t_{stp}) $\xleftarrow{\tau_0^A}$ $\mathcal{F}_L$ and (UPDATE, id_α, θ_B, t_{stp}) $\xleftarrow{\tau_0^B}$ $\mathcal{F}_L$, execute the simulator code for the update procedure of the generalized channel functionality until the message (UPDATE−REQ, id_α, θ_A, t_{stp}, tid_A) and (UPDATE−REQ, id_α, θ_B, t_{stp}, tid_B) are sent by $\mathcal{F}_L$.
(3) If in round $\tau_1^I \leq \tau_0^I + T$, I has received both (UPDATE−REQ, id_α, θ_A, t_{stp}, tid_A) $\hookleftarrow$ $\mathcal{F}_L$ and (UPDATE−REQ, id_β, θ_B, t_{stp}, tid_B) $\hookleftarrow$ $\mathcal{F}_L$, then execute the simulator code of SetupVChannel with inputs (γ, tid_A, tid_B). Or send (peaceful−reject, id_α) $\hookrightarrow$ $\mathcal{F}_V$ and (peaceful−reject, id_β) $\hookrightarrow$ $\mathcal{F}_V$ if instructed by $\mathcal{E}$. Else stop.
(4) If in round $\tau_2^I \leq \tau_1^I + t_{\mathrm{stp}} + T$, I receives both (SETUP−OK, id_α)) $\hookleftarrow$ $\mathcal{F}_L$ and (SETUP−OK, id_β)) $\hookleftarrow$ $\mathcal{F}_L$, continue executing the simulator code of the update procedure of generalized channels until the messages (REVOKE−REQ, id_α) and (REVOKE−REQ, id_α) are sent by $\mathcal{F}_L$. Otherwise stop.
(5) If in round $\tau_3^I \leq \tau_2^I + 4T$ you have received both (REVOKE−REQ, id_α) $\hookleftarrow$ $\mathcal{F}_L$ and (REVOKE−REQ, id_β) $\hookleftarrow$ $\mathcal{F}_L$, continue executing the simulator code of the update procedure of generalized channels until the end. Otherwise stop.
(6) If A or B have not sent (CREATE, γ) $\hookrightarrow$ $\mathcal{F}_V$ send this message on their behalf. Update $\Gamma^I(\gamma.\mathrm{id})$ from $(\perp, x)$ to (γ, x).

SetupVChannel **for channels without validity**

Case A is honest and I, B are corrupted

(1) Create the body of the funding transaction:

$$\mathtt{TX}^\gamma_\mathtt{f}.\mathrm{Input} := (tid_A, tid_B)$$

$$\mathtt{TX}^\gamma_\mathtt{f}.\mathrm{Output} := ((\gamma.\mathrm{cash}, \mathtt{Multi\text{−}Sig}_{\{\gamma.\mathrm{endUsers}\}}), (\gamma.\mathrm{cash} + \gamma.\mathrm{fee}, \mathtt{One\text{−}Sig}_{pk_I}))$$

(2) Upon A sending (PRE−CREATE, γ, $\mathtt{TX_f}$, 1, t_{ofl}) $\xleftarrow{t_0}$ $\mathcal{F}_{preL}$ where $t_{ofl} = 2T + 8\Delta$, execute the simulator code for the Pre-Create procedure of the $\mathcal{F}_{preL}$ functionality.

(3) Upon A receiving $(\texttt{PRE-CREATED}, \gamma.id) \xleftarrow{\tau_1 \leq \tau_0 + T} \mathcal{F}_{preL}$ then sign the funding transaction, i.e. $s_{\mathsf{f}}^B \leftarrow \mathsf{Sign}_{sk_B}([\mathsf{TX}_{\mathsf{f}}^\gamma])$ and send $(\text{createFund}, \gamma.\text{id}, s_{\mathsf{f}}^A, [\mathsf{TX}_{\mathsf{f}}^\gamma]) \xhookrightarrow{\tau_1} I$. Else record this execution as "failed" and stop.

(4) Upon receiving $(\text{createFund}, \gamma.\text{id}, s_{\mathsf{f}}^B, s_{\mathsf{f}}^I) \xleftrightarrow{\tau_1+1} I$, verify all signatures, i.e. check:

$$\mathsf{Vrfy}_{pk_B}([\mathsf{TX}_{\mathsf{f}}^\gamma]; s_{\mathsf{f}}^B) = 1$$
$$\mathsf{Vrfy}_{pk_I}([\mathsf{TX}_{\mathsf{f}}^\gamma], s_{\mathsf{f}}^I) = 1.$$

If all checks pass define

$$\mathsf{TX}_{\mathsf{f}}^\gamma := \{([\mathsf{TX}_{\mathsf{f}}^\gamma], s_{\mathsf{f}}^A, s_{\mathsf{f}}^B, s_{\mathsf{f}}^I)\},$$

and set

$$\Gamma^A(\gamma.\text{id}) := (\bot, \mathsf{TX}_{\mathsf{f}}^\gamma, tid_A)$$

and consider procedure successfully completed. Else record this execution as "failed" and stop.

Case I is honest and A, B are corrupted

(5) If I receives $(\text{createFund}, \gamma.\text{id}, s_{\mathsf{f}}^A, [\mathsf{TX}_{\mathsf{f}}^\gamma]) \xleftarrow{\tau_2 \leq \tau_0 + T + 1} A$ and $(\text{createFund}, \gamma.\text{id}, s_{\mathsf{f}}^B, [\mathsf{TX}_{\mathsf{f}}^\gamma]) \xleftarrow{\tau_2} B$, verify the funding transaction and signatures of A and B, i.e. check:

$$\mathsf{Vrfy}_{pk_A}([\mathsf{TX}_{\mathsf{f}}^\gamma]; s_{\mathsf{f}}^A) = 1$$
$$\mathsf{Vrfy}_{pk_B}([\mathsf{TX}_{\mathsf{f}}^\gamma], s_{\mathsf{f}}^B) = 1$$
$$(tid_A, tid_B) = \mathsf{TX}_{\mathsf{f}}^\gamma.\mathsf{Input}$$
$$(\gamma.\mathsf{cash} + \gamma.\mathsf{fee}, \mathsf{One\text{-}Sig}_{pk_I}) \in \mathsf{TX}_{\mathsf{f}}^\gamma.\mathsf{Output}.$$

(6) If all checks pass, sign the funding transaction, i.e. compute

$$s_{\mathsf{f}}^I := \mathsf{Sign}_{sk_I}([\mathsf{TX}_{\mathsf{f}}^\gamma]),$$
$$\mathsf{TX}_{\mathsf{f}}^\gamma := \{([\mathsf{TX}_{\mathsf{f}}^\gamma], s_{\mathsf{f}}^A, s_{\mathsf{f}}^B, s_{\mathsf{f}}^I)\}.$$

Store $\Gamma^I(\gamma.\text{id}) := (\bot, \mathsf{TX}_{\mathsf{f}}^\gamma)$. Then send $(\text{createFund}, \gamma.\text{id}, s_{\mathsf{f}}^B, s_{\mathsf{f}}^I) \xhookrightarrow{\tau_2} A$ and $(\text{createFund}, \gamma.\text{id}, s_{\mathsf{f}}^A, s_{\mathsf{f}}^I) \xhookrightarrow{\tau_2} B$, and consider procedure successfully completed. Else record this execution as "failed" and stop.

SetupVChannel for channels With validity

Case A is honest and B, I are corrupted

(1) Send $(\text{createInfo}, \gamma.\text{id}, tid_A) \xhookrightarrow{\tau_0} B$.

(2) In round $\tau_1 = \tau_0 + 1$, create the body of the funding transaction:

$$\mathsf{TX}_{\mathsf{f}}^\gamma.\mathsf{Input} := (tid_A)$$
$$\mathsf{TX}_{\mathsf{f}}^\gamma.\mathsf{Output} := ((\gamma.\mathsf{cash}, \mathtt{Multi\text{-}Sig}_{\{\gamma.\mathsf{endUsers}\}}), (\gamma.\mathsf{fee}/2, \mathsf{One\text{-}Sig}_{pk_I}))$$

(3) Upon A sending $(\texttt{PRE-CREATE}, \gamma, \mathsf{TX_f}, 1, t_{ofl}) \xhookrightarrow{\tau_1} \mathcal{F}_{preL}$ where $t_{ofl} = \gamma.\text{val} + 3\Delta$, execute the simulator code for the Pre-Create procedure of the $\mathcal{F}_{preL}$ functionality. If A does not receive the message $(\texttt{PRE-CREATED}, \gamma.id) \xleftarrow{\tau_2 \leq \tau_1 + T} \mathcal{F}_{preL}$ then mark this execution as "failed" and stop.

(4) If A receives $(\text{createFund}, \gamma.\text{id}, s_{\mathsf{f}}^I) \xleftrightarrow{\tau_2+2} I$, verify the signature, i.e. check:

$$\mathsf{Vrfy}_{sk_I}([\mathsf{TX}_{\mathsf{f}}^\gamma]; s_{\mathsf{f}}^I) = 1.$$

If the check passes, compute a signature on the fund transaction:

$$s_{\mathsf{f}}^A := \mathsf{Sign}_{sk_A}([\mathsf{TX}_{\mathsf{f}}^\gamma]),$$
$$\mathsf{TX}_{\mathsf{f}}^{\gamma,\mathsf{A}} := \{([\mathsf{TX}_{\mathsf{f}}^\gamma], s_{\mathsf{f}}^I, s_{\mathsf{f}}^A)\}.$$

Else record this execution as "failed" and stop.

(5) Set

$$\Gamma^A(\gamma.\text{id}) := (\bot, \mathsf{TX}_{\mathsf{f}}^{\gamma,\mathsf{A}}, tid_A)$$

and consider procedure successfully completed.

Case B is honest and A, I are corrupted

(1) If $(\text{createInfo}, \gamma.\text{id}, tid_A) \xleftrightarrow{\tau_0+1} A$, create the body of the funding and the first commit and split transactions:

$$\mathsf{TX}_{\mathsf{f}}^\gamma.\mathsf{Input} := (tid_A)$$
$$\mathsf{TX}_{\mathsf{f}}^\gamma.\mathsf{Output} := ((\gamma.\mathsf{cash}, \mathtt{Multi\text{-}Sig}_{\{\gamma.\mathsf{endUsers}\}}), (\gamma.\mathsf{fee}/2, \mathsf{One\text{-}Sig}_{pk_I}))$$
$$\mathsf{TX}_{\mathtt{refund}}^\gamma.\mathsf{Input} := (\mathsf{TX}_{\mathsf{f}}^\gamma.\mathsf{txid}||2, tid_B)$$
$$\mathsf{TX}_{\mathtt{refund}}^\gamma.\mathsf{Output} := (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}, \mathsf{One\text{-}Sig}_{pk_I}).$$

Else record this execution as "failed" and stop.

(2) Upon B sending $(\texttt{PRE-CREATE}, \gamma, \mathsf{TX_f}, 1, t_{ofl}) \xhookrightarrow{\tau_1 = \tau_0 + 1} \mathcal{F}_{preL}$ where $t_{ofl} = \gamma.\text{val} + 3\Delta$, execute the simulator code for the Pre-Create procedure of the $\mathcal{F}_{preL}$ functionality. If B does not receive $(\texttt{PRE-CREATED}, \gamma.id) \xleftarrow{\tau_2 \leq \tau_1 + T} \mathcal{F}_{preL}$ then mark this execution as "failed" and stop.

(3) Compute a signature on the refund transaction, i.e., $s_{\mathsf{Ref}}^B \leftarrow \mathsf{Sign}_{sk_B}([\mathsf{TX}_{\mathtt{refund}}^\gamma])$ and define $\mathsf{TX}_{\mathsf{f}}^{\gamma,\mathsf{B}} := \{([\mathsf{TX}_{\mathsf{f}}^\gamma])\}$. Then, send $(\text{createFund}, \gamma.\text{id}, s_{\mathsf{Ref}}^B, [\mathsf{TX}_{\mathtt{refund}}^\gamma], [\mathsf{TX}_{\mathsf{f}}^\gamma]) \xhookrightarrow{\tau_2} I$, set

$$\Gamma^B(\gamma.\text{id}) := (\bot, \mathsf{TX}_{\mathsf{f}}^{\gamma,\mathsf{B}}, tid_B)$$

and consider procedure successfully completed. Else record this execution as "failed" and stop.

Case I is honest and A, B are corrupted

(4) If I receives the message $(\text{createFund}, \gamma.\text{id}, s_{\mathsf{Ref}}^B, [\mathsf{TX}_{\mathtt{refund}}^\gamma], [\mathsf{TX}_{\mathsf{f}}^\gamma]) \xleftrightarrow{\tau_3 \leq \tau_0 + T + 2} B$, verify the fund and refund transactions and signature of B, i.e. check:

$$\mathsf{Vrfy}_{sk_B}([\mathsf{TX}_{\mathtt{refund}}^\gamma]; s_{\mathsf{Ref}}^B) = 1.$$
$$[\mathsf{TX}_{\mathtt{refund}}^\gamma].\mathsf{Input} = (\mathsf{TX}_{\mathsf{f}}^\gamma.\mathsf{txid}||2, tid_B),$$
$$[\mathsf{TX}_{\mathtt{refund}}^\gamma].\mathsf{Output} = (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}, \mathsf{One\text{-}Sig}_{pk_I}),$$
$$[\mathsf{TX}_{\mathsf{f}}^\gamma].\mathsf{Output}[2] = (\gamma.\mathsf{fee}/2, \mathsf{One\text{-}Sig}_{pk_I})$$

If all checks pass, sign the fund and refund transactions, i.e. compute

$$s_{\mathsf{Ref}}^I := \mathsf{Sign}_{sk_I}([\mathsf{TX}_{\mathtt{refund}}^\gamma]),\ s_{\mathsf{f}}^I := \mathsf{Sign}_{sk_I}([\mathsf{TX}_{\mathsf{f}}^\gamma]),$$
$$\mathsf{TX}_{\mathtt{refund}}^\gamma := \{([\mathsf{TX}_{\mathtt{refund}}^\gamma], s_{\mathsf{Ref}}^I, s_{\mathsf{Ref}}^B)\}.$$

Store $\Gamma^I(\gamma.\text{id}) := (\bot, [\mathsf{TX}_{\mathsf{f}}^\gamma], \mathsf{TX}_{\mathtt{refund}}^\gamma, tid_A, tid_B)$. Then send $(\text{createFund}, \gamma.\text{id}, s_{\mathsf{f}}^I) \xhookrightarrow{\tau_3} A$, and consider procedure successfully completed. Else record this execution as "failed" and stop.

Function $\mathtt{GenVChannelOutput}(\gamma, P)$

Return θ, where $\theta.\mathsf{cash} = \gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2$ and $\theta.\varphi$ is defined as follows

$$\theta.\varphi = \begin{cases} \mathtt{Multi\text{-}Sig}_{\gamma.\mathsf{users}} \vee (\mathtt{One\text{-}Sig}_P \wedge \mathtt{CheckRelative}_{(T+4\Delta)}), \\ \qquad\qquad \text{if } \gamma.\mathsf{val} = \bot \\ \mathtt{Multi\text{-}Sig}_{A,I} \vee (\mathtt{One\text{-}Sig}_I \wedge \mathtt{CheckLockTime}_{\gamma.\mathsf{val}}), \\ \qquad\qquad \text{if } \gamma.\mathsf{val} \neq \bot \wedge P = A \\ \mathtt{Multi\text{-}Sig}_{B,I} \vee (\mathtt{One\text{-}Sig}_B \wedge \mathtt{CheckLockTime}_{\gamma.\mathsf{val}+2\Delta}), \\ \qquad\qquad \text{if } \gamma.\mathsf{val} \neq \bot \wedge P = B \end{cases}$$

Simulator for updating virtual channels

Update virtual channels

Case A is honest and B is corrupt

Below we abbreviate $\mathcal{F}_{preL} := \mathcal{F}_{preL}(T, 1)$ and assume A is the initiating party.

(1) Upon A sending $(\mathtt{UPDATE}, id, \vec{\theta}, t_{\mathsf{stp}}) \overset{\tau_0^A}{\hookrightarrow} \mathcal{F}_{preL}$, execute the simulator for the pre-update procedure of the $\mathcal{F}_{preL}$ functionality from beginning until PRE–SETUP is sent. If this execution is marked "failed" stop.

(2) Upon A sending $(\mathtt{SETUP\text{–}OK}, id) \overset{\tau_2^A \leq \tau_1^A + t_{\mathsf{stp}}}{\longleftarrow\!\shortmid} \mathcal{F}_{preL}$, continue executing the simulator code until step 4. If this execution is marked "failed" stop.

(3) If A does not receive $(\mathtt{PRE\text{–}UPDATE\text{–}OK}, id) \overset{\tau_3^A \leq \tau_2^A + T}{\longleftarrow\!\shortmid} \mathcal{F}_{preL}$ or $(\mathtt{PRE\text{–}UPDATE\text{–}REJECT}, id) \overset{\tau_3^A \leq \tau_2^A + T}{\longleftarrow\!\shortmid} \mathcal{F}_{preL}$, execute the simulator code for the procedure $\mathtt{Offload}^A(id)$ and stop.

(4) Upon A sending $(\mathtt{PRE\text{–}REVOKE}, id) \overset{\tau_3^A}{\hookrightarrow} \mathcal{F}_{preL}$ continue executing the simulator code until the end. If this execution is marked as "failed" execute the simulator code for the procedure $\mathtt{Offload}^A(id)$ and stop.

(5) Upon A receiving $(\mathtt{PRE\text{–}UPDATED}, id) \overset{\tau_4^A \leq \tau_3^A + T}{\longleftarrow\!\shortmid} \mathcal{F}_{preL}$, update the channel space, i.e., let $\gamma := \Gamma^A(id)$, set $\gamma.\mathsf{st} := \vec{\theta}$ and $\Gamma(id) := \gamma$. Else if this execution is marked as "failed" execute the simulator code for the procedure $\mathtt{Offload}^A(id)$ and stop.

Case B is honest and A is corrupt

(1) Let τ_0^B be the round in which B receives $(\mathtt{PRE\text{–}UPDATE\text{–}REQ}, id, \vec{\theta}, t_{\mathsf{stp}}, tid) \overset{\tau_0^B}{\longleftarrow\!\shortmid} \mathcal{F}_{preL}$.

(2) Let $\tau_1^B \leq \tau_0 + t_{\mathsf{stp}} + T$ be the round in which B receives the message $(\mathtt{PRE\text{–}SETUP\text{–}OK}, id) \overset{\tau_1^B \leq \tau_0 + t_{\mathsf{stp}} + T}{\longleftarrow\!\shortmid} \mathcal{F}_{preL}$.

(3) Upon B sending $(\mathtt{UPDATE\text{–}OK}, id) \overset{\tau_1^B}{\longleftarrow\!\shortmid} \mathcal{F}_{preL}$ execute the simulator code of the pre-update procedure for the $\mathcal{F}_{preL}$ functionality until the message PRE–REVOKE–REQ is sent by the functionality and let the this round be $\tau_2^B \leq \tau_1^B + T$. If this execution is marked as "failed" execute the simulator code of the procedure $\mathtt{Offload}^B(id)$ and stop.

(4) Upon B sending $(\mathtt{PRE\text{–}REVOKE}, id) \overset{\tau_2^B}{\hookrightarrow} \mathcal{F}_{preL}$ continue executing the simulator code until the end. If this execution is marked as "failed" execute the simulator code for the procedure $\mathtt{Offload}^B(id)$ and stop.

(5) Upon B receiving $(\mathtt{PRE\text{–}UPDATED}, id) \overset{\tau_3^B \leq \tau_2^B + T}{\longleftarrow\!\shortmid} \mathcal{F}_{preL}$, update the channel space, i.e., let $\gamma := \Gamma^B(id)$, set $\gamma.\mathsf{st} := \vec{\theta}$ and $\Gamma(id) := \gamma$.

Simulator for offloading virtual channels

Offloading virtual channels without validity

Case A honest and I, B corrupted

(1) Extract γ and $\mathsf{TX}_{\mathsf{f}}^{\gamma}$ from $\Gamma^A(id)$ and tid_A, tid_B from $\mathsf{TX}_{\mathsf{f}}^{\gamma}$. Then define $id_\alpha := \gamma.\mathsf{subchan}(A)$. Upon A sending $(\mathtt{CLOSE}, id_\alpha) \overset{\tau_0}{\hookrightarrow} \mathcal{F}_L$ execute the simulator code for the close procedure of generalized ledger channels.

(2) If A receives $(\mathtt{CLOSED}, id_\alpha) \overset{\tau_1 \leq \tau_0 + T + 3\Delta}{\longleftarrow\!\shortmid} \mathcal{F}_L$, check that a transaction with tid_A appeared on $\widehat{\mathcal{L}}$. Else stop.

(3) Let $T_2 := \tau_1 + T + 3\Delta$ and distinguish:
- If in round $\tau_2 \leq T_2$ a transaction with tid_B appeared on $\widehat{\mathcal{L}}$, then $(\mathsf{post}, \mathsf{TX}_{\mathsf{f}}^{\gamma}) \overset{\tau_2}{\hookrightarrow} \widehat{\mathcal{L}}$.
- Else in round T_2 create the punishment transaction $\mathsf{TX}_{\mathsf{pun}}$ as $\mathsf{TX}_{\mathsf{pun}}.\mathsf{Input} := tid_A$, $\mathsf{TX}_{\mathsf{pun}}.\mathsf{Output} := (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \mathtt{One\text{-}Sig}_{pk_A})$ and $\mathsf{TX}_{\mathsf{pun}}.\mathsf{Witness} := \mathsf{Sign}_{sk_A}([\mathsf{TX}_{\mathsf{pun}}])$. Then $(\mathsf{post}, \mathsf{TX}_{\mathsf{pun}}) \overset{T_2}{\hookrightarrow} \widehat{\mathcal{L}}$.

(4) Let $T_3 := \tau_2 + \Delta$ and distinguish the following two cases:
- The transaction $\mathsf{TX}_{\mathsf{f}}^{\gamma}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3 \leq T_3$, then update $\Gamma^A := \mathtt{ToLedgerChannel}(\Gamma^A, \gamma.\mathsf{id})$ and set $m :=$ offloaded.
- The transaction $\mathsf{TX}_{\mathsf{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3 \leq T_3$, then define $\Gamma^A(\gamma.\mathsf{id}) = \bot$ and set $m :=$ punished.

(5) Return m in round τ_3.

Case I honest and A, B corrupted

(1) Extract γ and $\mathsf{TX}_{\mathsf{f}}^{\gamma}$ from $\Gamma^I(id)$ and tid_A, tid_B from $\mathsf{TX}_{\mathsf{f}}^{\gamma}$. Then define $id_\alpha := \gamma.\mathsf{subchan}(A)$, $id_\beta := \gamma.\mathsf{subchan}(B)$ Upon I sending the messages $(\mathtt{CLOSE}, id_\alpha) \overset{\tau_0}{\hookrightarrow} \mathcal{F}_L$ and $(\mathtt{CLOSE}, id_\beta) \overset{\tau_0}{\hookrightarrow} \mathcal{F}_L$ execute the simulator code of the close procedure for the generalized channels.

(2) If I receives both $(m, id_\alpha) \overset{\tau_1^A \leq \tau_0 + T + 3\Delta}{\longleftarrow\!\shortmid} \mathcal{F}_L$ and $(\mathtt{CLOSED}, id_\beta) \overset{\tau_1^B \leq \tau_0 + T + 3\Delta}{\longleftarrow\!\shortmid} \mathcal{F}_L$, check that a transaction with tid_A and a transaction with tid_B appeared on $\widehat{\mathcal{L}}$. Then publish $(\mathsf{post}, \mathsf{TX}_{\mathsf{f}}^{\gamma}) \overset{\tau_1}{\hookrightarrow} \widehat{\mathcal{L}}$, where $\tau_1 := \max\{\tau_1^A, \tau_1^B\}$. Otherwise set $\Gamma^I(id) = \bot$ and stop.

(3) Once $\mathsf{TX}_{\mathsf{f}}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round $\tau_2 \leq \tau_1 + \Delta$, then $\Gamma^I(\gamma.\mathsf{id}) = \bot$ and return "offloaded".

Offloading virtual channels with validity

Case A honest and B, I corrupted

(1) Extract γ, tid_A and $\mathsf{TX}_{\mathsf{f}}^{\gamma}$ from $\Gamma^A(id)$. Then define $id_\alpha := \gamma.\mathsf{subchan}(A)$ and Upon A sending $(\mathtt{CLOSE}, id_\alpha) \overset{\tau_0}{\hookrightarrow} \mathcal{F}_L$ execute the simulator code of the close procedure of the generalized ledger channel.

(2) If A receives $(\texttt{CLOSED}, id_\alpha) \xleftarrow{\tau_1 \le \tau_0+T+3\Delta} \mathcal{F}_L$, then post $(\text{post}, \texttt{TX}_\text{f}^{\gamma,\text{A}}) \overset{\tau_2}{\hookrightarrow} \widehat{\mathcal{L}}$. Otherwise, set $\Gamma^A(\gamma.\text{id}) = \bot$ and stop.

(3) Once $\texttt{TX}_\text{f}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round $\tau_2 \le \tau_1 + \Delta$, then set $\Gamma_L^A(id) := \Gamma^A(id)$, $\Gamma^A(id) := \bot$ and return "offloaded".

Case B honest and A, I corrupted

(1) Extract γ, tid_B and $[\texttt{TX}_\text{f}^{\gamma}]$ from $\Gamma^B(id)$. Then define $id_\beta := \gamma.\text{subchan}(B)$ and Upon B sending $(\texttt{CLOSE}, id_\beta) \overset{\tau_0}{\hookrightarrow} \mathcal{F}_L$ execute the simulator code of the close procedure of the generalized ledger channel.

(2) If B receives $(\texttt{CLOSED}, id_\beta) \xleftarrow{\tau_1 \le \tau_0+T+3\Delta} \mathcal{F}_L$, then continue. Otherwise, set $\Gamma^B(\gamma.\text{id}) = \bot$ and stop.

(3) Create the punishment transaction $\texttt{TX}_\text{pun}$ as $\texttt{TX}_\text{pun}.\text{Input} := tid_B$, $\texttt{TX}_\text{pun}.\text{Output} := (\gamma.\text{cash} + \gamma.\text{fee}/2, \texttt{One-Sig}_{pk_B})$ and $\texttt{TX}_\text{pun}.\text{Witness} := \text{Sign}_{sk_B}([\texttt{TX}_\text{pun}])$. Then wait until round $\tau_2 := \max\{\tau_1, \gamma.\text{val} + 2\Delta\}$ and send $(\text{post}, \texttt{TX}_\text{pun}) \overset{\tau_2}{\hookrightarrow} \widehat{\mathcal{L}}$.

(4) Let $T_3 := \tau_2 + \Delta$ and distinguish the following two cases:
- A transaction with identifier $\texttt{TX}_\text{f}^{\gamma}.\text{txid}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3 \le T_3$, then update $\Gamma_L^B(id) := \Gamma^B(id)$ and set $m :=$ offloaded.
- The transaction $\texttt{TX}_\text{pun}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3 \le T_3$, then define $\Gamma^B(\gamma.\text{id}) = \bot$, and set $m :=$ punished.

(5) Return m in round τ_3.

Case I honest and A, B corrupted

(1) Extract γ, tid_A, tid_B, $\texttt{TX}_{\texttt{refund}}^{\gamma}$ and $[\texttt{TX}_\text{f}^{\gamma}]$ from $\Gamma^I(id)$. Then define $id_\alpha := \gamma.\text{subchan}(A)$, $id_\beta := \gamma.\text{subchan}(B)$ and upon I sending $(\texttt{CLOSE}, id_\alpha) \overset{\tau_0}{\hookrightarrow} \mathcal{F}_L$ and $(\texttt{CLOSE}, id_\beta) \overset{\tau_0}{\hookrightarrow} \mathcal{F}_L$ execute the simulator code of the close procedure of the generalized ledger channel for both id_α and id_β.

(2) If I receives both $(\texttt{CLOSED}, id_\alpha) \xleftarrow{\tau_1^A \le \tau_0+T+3\Delta} \mathcal{F}_L$ and $(\texttt{CLOSED}, id_\beta) \xleftarrow{\tau_1^B \le \tau_0+T+3\Delta} \mathcal{F}_L$, then continue. Otherwise, set $\Gamma^I(\gamma.\text{id}) = \bot$ and stop.

(3) Create the punishment transaction $\texttt{TX}_\text{pun}$ as $\texttt{TX}_\text{pun}.\text{Input} := tid_A$, $\texttt{TX}_\text{pun}.\text{Output} := (\gamma.\text{cash} + \gamma.\text{fee}/2, \texttt{One-Sig}_{pk_I})$ and $\texttt{TX}_\text{pun}.\text{Witness} := \text{Sign}_{sk_I}([\texttt{TX}_\text{pun}])$. Then wait until round $\tau_2 := \max\{\tau_1^A, \gamma.\text{val}\}$ and send $(\text{post}, \texttt{TX}_\text{pun}) \overset{\tau_2}{\hookrightarrow} \widehat{\mathcal{L}}$.

(4) Let $T_3 := \tau_2 + \Delta$ and distinguish the following two cases:
- A transaction with identifier $\texttt{TX}_\text{f}^{\gamma}.\text{txid}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3' \le T_3$, send $(\text{post}, \texttt{TX}_{\texttt{refund}}^{\gamma}) \overset{\tau_4}{\hookrightarrow} \widehat{\mathcal{L}}$ where $\tau_4 := \max\{\tau_1^B, \tau_3'\}$. Once $\texttt{TX}_{\texttt{refund}}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round $\tau_5 \le \tau_4 + \Delta$, set $\Gamma^I(\gamma.\text{id}) = \bot$ and $m :=$ offloaded.
- The transaction $\texttt{TX}_\text{pun}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3'' \le T_3$, then set $\Gamma^I(\gamma.\text{id}) = \bot$ and $m :=$ punished.

(5) Return m in round τ_6 where $\tau_6 := \max\{\tau_5, \tau_3''\}$.

Simulator for punishing in a virtual channel

- Upon a party sending $(\texttt{PUNISH}) \overset{\tau_0}{\hookrightarrow} \mathcal{F}_L$, execute the simulator code for the punish procedure of the generalized channels.
- Execute the simulator code for both Punish and Punish-Validity.

Punish

Case A honest and I, B corrupted

For every $id \in \{0,1\}^*$, such that γ which $\gamma.\text{val} = \bot$ can be extracted from $\Gamma^A(id)$ do the following:

(1) Extract $\texttt{TX}_\text{f}^{\gamma}$ from $\Gamma^A(id)$ and tid_A, tid_B from $\texttt{TX}_\text{f}^{\gamma}$. Check if tid_A appeared on $\widehat{\mathcal{L}}$. If not, then stop.

(2) Denote $T_2 := \tau_1 + T + 3\Delta$ and distinguish:
- If in round $\tau_2 \le T_2$ the transaction with tid_B appeared on $\widehat{\mathcal{L}}$, then $(\text{post}, \texttt{TX}_\text{f}^{\gamma}) \overset{\tau_2}{\hookrightarrow} \widehat{\mathcal{L}}$.
- Else in round T_2 create the punishment transaction $\texttt{TX}_\text{pun}$ as $\texttt{TX}_\text{pun}.\text{Input} := tid_A$, $\texttt{TX}_\text{pun}.\text{Output} := (\gamma.\text{cash} + \gamma.\text{fee}/2, \texttt{One-Sig}_{pk_A})$ and $\texttt{TX}_\text{pun}.\text{Witness} := \text{Sign}_{sk_A}([\texttt{TX}_\text{pun}])$ and $(\text{post}, \texttt{TX}_\text{pun}) \overset{T_2}{\hookrightarrow} \widehat{\mathcal{L}}$.

(3) Let $T_3 := \tau_2 + \Delta$ and distinguish the following two cases:
- The transaction $\texttt{TX}_\text{f}^{\gamma}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3 \le T_3$, then update $\Gamma^A := \texttt{ToLedgerChannel}(\Gamma^A, \gamma.\text{id})$.
- The transaction $\texttt{TX}_\text{pun}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3 \le T_3$, then define $\Gamma^A(\gamma.\text{id}) = \bot$.

Case I honest and A, B corrupted

For every $id \in \{0,1\}^*$, such that γ with $\gamma.\text{val} = \bot$ can be extracted from $\Gamma^I(id)$ do the following:

(1) Extract $\texttt{TX}_\text{f}^{\gamma}$ from $\Gamma^I(id)$ and tid_A, tid_B from $\texttt{TX}_\text{f}^{\gamma}$. Check if for some $P \in \{A, B\}$ a transaction with identifier tid_P appeared on $\widehat{\mathcal{L}}$. If not, then stop.

(2) Denote $id_\alpha := \gamma.\text{subchan}(Q)$ where $Q = \gamma.\text{otherParty}(P)$ and upon I sending $(\texttt{CLOSE}, id_\alpha) \overset{\tau_0}{\hookrightarrow} \mathcal{F}_L$ execute simulator the code for the punish procedure of the generalized virtual channels.

(3) If I receives $(\texttt{CLOSED}, id_\alpha) \xleftarrow{\tau_1 \le \tau_0+T+3\Delta} \mathcal{F}_L$ and tid_Q appeared on $\widehat{\mathcal{L}}$, $(\text{post}, \texttt{TX}_\text{f}^{\gamma}) \overset{\tau_1}{\hookrightarrow} \widehat{\mathcal{L}}$. Otherwise set $\Gamma^I(id) = \bot$ and stop.

(4) Once $\texttt{TX}_\text{f}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round τ_2, such that $\tau_2 \le \tau_1 + \Delta$, set $\Gamma^I(id) = \bot$.

Punish-Validity

Case A honest and I, B corrupted

For every $id \in \{0,1\}^*$, such that γ which $\gamma.\text{val} \ne \bot$ can be extracted from $\Gamma^A(id)$ do the following:

(1) Extract tid_A and $\texttt{TX}_\text{f}^{\gamma}$ from $\Gamma^A(id)$. Check if tid_A appeared on $\widehat{\mathcal{L}}$. If not, then stop.

(2) Send $(\text{post}, \texttt{TX}_\text{f}^{\gamma}) \overset{\tau_1}{\hookrightarrow} \widehat{\mathcal{L}}$.

(3) Once $\texttt{TX}_\text{f}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round $\tau_2 \le \tau_1 + \Delta$, set $\Gamma_L^A(id) := \Gamma^A(id)$, $\Gamma^A(id) := \bot$.

Case B honest and A, I corrupted

For every $id \in \{0,1\}^*$, such that γ which $\gamma.\text{val} \ne \bot$ can be extracted from $\Gamma^B(id)$ do the following:

(1) Extract tid_B and $[\texttt{TX}_\text{f}^{\gamma}]$ from $\Gamma^B(id)$. Check if tid_B or $[\texttt{TX}_\text{f}^{\gamma}].\text{txid}$ appeared on $\widehat{\mathcal{L}}$. If not, then stop.

(2) If $\texttt{TX}_\text{f}^{\gamma}$ appeared on $\widehat{\mathcal{L}}$, set $\Gamma_L^B(id) := \Gamma^B(id)$, $\Gamma^B(id) := \bot$ and stop.

(3) If tid_B appeared on $\widehat{\mathcal{L}}$, create the punishment transaction $\texttt{TX}_\text{pun}$ as $\texttt{TX}_\text{pun}.\text{Input} := tid_B$, $\texttt{TX}_\text{pun}.\text{Output} := (\gamma.\text{cash} + \gamma.\text{fee}/2, \texttt{One-Sig}_{pk_B})$ and $\texttt{TX}_\text{pun}.\text{Witness} := \text{Sign}_{sk_B}([\texttt{TX}_\text{pun}])$. Then wait until round $\tau_2 := \max\{\tau_1, \gamma.\text{val} + 2\Delta\}$ and send $(\text{post}, \texttt{TX}_\text{pun}) \overset{\tau_2}{\hookrightarrow} \widehat{\mathcal{L}}$.

(4) If transaction $\mathsf{TX}_{\mathsf{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3 \leq \tau_2 + \Delta$, then define $\Gamma^B(\gamma.\mathrm{id}) = \bot$.

Case I honest and A, B corrupted

For every $id \in \{0,1\}^*$, such that γ which $\gamma.\mathrm{val} \neq \bot$ can be extracted from $\Gamma^I(id)$ do the following:

(1) Extract tid_A, tid_B, $\mathsf{TX}^{\gamma}_{\mathsf{refund}}$ and $[\mathsf{TX}^{\gamma}_{\mathsf{f}}]$ from $\Gamma^I(id)$. Check if tid_A or tid_B appeared on $\widehat{\mathcal{L}}$ or $\tau_1 = \gamma.\mathrm{val} - (3\Delta + T)$. If not, then stop.

(2) Distinguish the following cases:
- If $\tau_1 = \gamma.\mathrm{val} - (3\Delta + T)$, define $id_\alpha := \gamma.\mathrm{subchan}(A)$, $id_\beta := \gamma.\mathrm{subchan}(B)$ and upon I sending the messages $(\mathtt{CLOSE}, id_\alpha) \overset{\tau_1}{\hookrightarrow} \mathcal{F}_L$ and $(\mathtt{CLOSE}, id_\beta) \overset{\tau_1}{\hookrightarrow} \mathcal{F}_L$ execute the simulator code for the close procedure of the generalized channels for both channels id_α and id_β.
- If tid_B appeared on $\widehat{\mathcal{L}}$, send $(\mathtt{CLOSE}, id_\alpha) \overset{\tau_1}{\hookrightarrow} \mathcal{F}_L$.

(3) If a transaction with identifier tid_A appeared on $\widehat{\mathcal{L}}$ in round $\tau_2 \leq \tau_1 + T + 3\Delta$, create the punishment transaction $\mathsf{TX}_{\mathsf{pun}}$ as $\mathsf{TX}_{\mathsf{pun}}.\mathsf{Input} := tid_A$, $\mathsf{TX}_{\mathsf{pun}}.\mathsf{Output} := (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \mathtt{One\text{-}Sig}_{pk_I})$ and $\mathsf{TX}_{\mathsf{pun}}.\mathsf{Witness} := \mathrm{Sign}_{sk_I}([\mathsf{TX}_{\mathsf{pun}}])$. Then wait until round $\tau_3 := \max\{\tau_2, \gamma.\mathrm{val}\}$ and send $(\mathrm{post}, \mathsf{TX}_{\mathsf{pun}}) \overset{\tau_3}{\hookrightarrow} \widehat{\mathcal{L}}$.

(4) Distinguish the following two cases:
- The transaction $\mathsf{TX}^{\gamma}_{\mathsf{f}}.\mathrm{txid}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_4 \leq \tau_3 + \Delta$, send $(\mathrm{post}, \mathsf{TX}^{\gamma}_{\mathsf{refund}}) \overset{\tau_5}{\hookrightarrow} \widehat{\mathcal{L}}$ where $\tau_5 := \max\{\gamma.\mathrm{val} + \Delta, \tau_4\}$. Once $\mathsf{TX}^{\gamma}_{\mathsf{refund}}$ is accepted by $\widehat{\mathcal{L}}$ in round $\tau_6 \leq \tau_5 + \Delta$, set $\Gamma^I(\gamma.\mathrm{id}) = \bot$ and stop.
- The transaction $\mathsf{TX}_{\mathsf{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_4 \leq \tau_3 + \Delta$, then set $\Gamma^I(\gamma.\mathrm{id}) = \bot$.

Simulator for Close in a virtual channel

Closing virtual channels

Case A honest and I, B corrupted

(1) Upon A sending $(\mathtt{CLOSE}, id) \overset{\tau_0^A}{\hookrightarrow} \mathcal{F}_V$ or in round $\tau_0^A = \gamma.\mathrm{val} - (4\Delta + 7T)$ if $\gamma.\mathrm{val} \neq \bot$, proceed as follows:

(2) Extract γ, $\mathsf{TX}^{\gamma}_{\mathsf{f}}$, tid_A from $\Gamma^A(id)$. Parse

$$\mathsf{TX}^{\gamma}_{\mathsf{s}}.\mathsf{Output} = \left((c_A, \mathtt{One\text{-}Sig}_{pk_A}), (c_B, \mathtt{One\text{-}Sig}_{pk_B})\right).$$

(3) Compute the new state of the channel $id_\alpha := \gamma.\mathrm{subchan}(A)$ as

$$\vec{\theta}_A := \{(c_A, \mathtt{One\text{-}Sig}_{pk_A}), (\gamma.\mathsf{cash} - c_A + \frac{\gamma.\mathsf{fee}}{2}, \mathtt{One\text{-}Sig}_{pk_I})\}$$

Then, upon A sending $(\mathtt{UPDATE}, id_\alpha, \vec{\theta}_A, 0) \overset{\tau_0^A}{\hookrightarrow} \mathcal{F}_L$ execute the simulator code for the update procedure of generalized channels until $(\mathtt{SETUP}, id_\alpha, \vec{tid}'_A)$ is sent by $\mathcal{F}_L$. If this execution fails instruct $\mathcal{F}_V$ to execute $\mathtt{V\text{-}ForceClose}(id)$ and stop.

(4) Upon A sending $(\mathtt{SETUP\text{-}OK}, id_\alpha) \overset{\tau_1^A \leq \tau_0^A + T}{\longrightarrow} \mathcal{F}_L$ continue executing the simulator code for the update procedure of generalized channels until A receives $(\mathtt{UPDATE\text{-}OK}, id_\alpha) \overset{\tau_2^A \leq \tau_1^A + 2T}{\longleftarrow} \mathcal{F}_L$. If this execution fails instruct $\mathcal{F}_V$ to execute $\mathtt{V\text{-}ForceClose}(id)$ and stop.

(5) Upon A sending $(\mathtt{REVOKE}, id_\alpha) \overset{\tau_2^A}{\hookrightarrow} \mathcal{F}_L$ continue executing the simulator code for the update procedure of generalized channels until A receives $(\mathtt{UPDATED}, id_\alpha) \overset{\tau_3^A \leq \tau_2^A + 2T}{\longleftarrow} \mathcal{F}_L$, then set $\Gamma^A(id) := \bot$ and stop. If this execution fails instruct $\mathcal{F}_V$ to execute $\mathtt{V\text{-}ForceClose}(id)$ and stop.

Case I honest and A, B corrupted

(1) Upon I sending $(\mathtt{CLOSE}, id) \overset{\tau_0^I}{\hookrightarrow} \mathcal{F}_V$ or in round $\tau_0^A = \gamma.\mathrm{val} - (4\Delta + 7T)$ if $\gamma.\mathrm{val} \neq \bot$, proceed as follows:

(2) Extract γ, $\mathsf{TX}^{\gamma}_{\mathsf{f}}$, tid_A and tid_B from $\Gamma^I(id)$.

(3) Let $id_\alpha = \gamma.\mathrm{subchan}(A)$, $id_\beta = \gamma.\mathrm{subchan}(B)$ and $c := \gamma.\mathsf{cash}$, execute the simulator code for the update procedure of generalized channels until $(\mathtt{UPDATE\text{-}REQ}, id_\alpha, \vec{tid}'_A, \vec{\theta}_A, 0) \leftrightarrow \mathcal{F}_L$ and $(\mathtt{UPDATE\text{-}REQ}, id_\beta, \vec{tid}'_B, \vec{\theta}_B, 0) \leftrightarrow \mathcal{F}_L$ are sent by $\mathcal{F}_L$ until round $\tau_1^I \leq \tau_0^I + T$.

(4) Check that for some c_A, c_B s.t. $c_A + c_B + \gamma.\mathsf{fee} = c$ it holds

$$\vec{\theta}_A = \{(c_A, \mathtt{One\text{-}Sig}_{pk_A}), (c - c_A + \gamma.\mathsf{fee}/2, \mathtt{One\text{-}Sig}_{pk_I})\}$$

$$\vec{\theta}_B = \{(c_B, \mathtt{One\text{-}Sig}_{pk_B}), (c - c_B + \gamma.\mathsf{fee}/2, \mathtt{One\text{-}Sig}_{pk_I})\}$$

If not, then stop.

(5) continue executing the simulator code for the update procedure of generalized channels until I receives $(\mathtt{SETUP\text{-}OK}, id_\alpha) \leftrightarrow \mathcal{F}_L$ and $(\mathtt{SETUP\text{-}OK}, id_\beta)) \leftrightarrow \mathcal{F}_L$ until round $\tau_2^I \leq \tau_1^I + T$. Otherwise stop.

(6) Upon I sending $(\mathtt{UPDATE\text{-}OK}, id_\beta) \overset{\tau_2^I}{\hookrightarrow} \mathcal{F}_L$ continue executing the simulator code for the update procedure of generalized channels until I receives the messages $(\mathtt{REVOKE\text{-}REQ}, id_\alpha) \leftrightarrow \mathcal{F}_L$ and $(\mathtt{REVOKE\text{-}REQ}, id_\beta) \leftrightarrow \mathcal{F}_L$ until round $\tau_3^I \leq \tau_2^I + 2T$.

(7) Upon I sending $(\mathtt{REVOKE}, id_\alpha) \overset{\tau_3^I}{\hookrightarrow} \mathcal{F}_L$ and $(\mathtt{REVOKE}, id_\beta) \overset{\tau_3^I}{\hookrightarrow} \mathcal{F}_L$ executing the simulator code for the update procedure of generalized channels until the end and set $\Gamma^I(id) := \bot$.