Bitcoin-Compatible Virtual Channels

Lukas Aumayr*, Oğuzhan Ersoy†, Andreas Erwig‡, Sebastian Faust‡,
Kristina Hostáková§, Matteo Maffei*, Pedro Moreno-Sanchez¶, Siavash Riahi‡
* TU Wien, {lukas.aumayr, matteo.maffei}@tuwien.ac.at
†TU Delft, o.ersoy@tudelft.nl
‡TU Darmstadt, {firstname.surname}@tu-darmstadt.de
§ETH Zürich, kristina.hostakova@inf.ethz.ch
¶IMDEA Software Institute, pedro.moreno@imdea.org

***Abstract*—Current permissionless cryptocurrencies such as Bitcoin suffer from a limited transaction rate and slow confirmation time, which hinders further adoption. Payment channels are one of the most promising solutions to address these problems, as they allow the parties 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 direct 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 (IEEE S&P'19), 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 so far considered an open challenge.**

In this work, we present the first virtual channel protocols that are built on the UTXO-model and require a scripting language supporting only a digital signature scheme and a timelock functionality, being thus backward 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 9+2n transactions or a total of 3524+695n bytes, with no on-chain footprint in the optimistic case. This is a substantial improvement compared to routing payments in a payment channel network, which requires 8n transactions with a total of 3026n bytes to be exchanged.

I. Introduction

Permissionless cryptocurrencies such as Bitcoin [24] 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 on its core on a decentralized consensus protocol for these tasks. The consensus protocol establishes and maintains a distributed ledger that tracks every 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 [16, 31, 4], payment channels [7] have emerged as one of the most promising scalability solutions. The most prominent example that is currently deployed over Bitcoin is the so-called Lightning network [26], which at the time of writing hosts deposits worth more than 60M 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) [26, 22]. 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 Bob. The standard technique for constructing PCNs requires the intermediary (e.g., Ingrid in the example from above) to be actively involved in each payment. This has multiple disadvantages, including (i) making the system less reliable (e.g., Ingrid might have to go offline), (ii) increasing the latency of each payment,

(iii) augmenting its costs since each intermediary charges a fee per transaction, and (iv) revealing possibly sensitive payment information to the intermediaries [25, 29, 19].

An alternative approach for connecting multiple payment channels was introduced by Dziembowski et al. [13]. 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 is so-called payment hubs [13]. Since establishing a payment channel requires a deposit and active monitoring, the number of channels a user can establish is limited. With payment hubs [13], 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 very challenging since, as previously mentioned, it has to account for all possible compromise and collusion scenarios. For this purpose, existing virtual channel constructions [13] require smart contracts programmed over an expressive scripting 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 programmatically be updated. On the downside, however, these requirements currently limit 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. We believe that answering this question is important for several reasons. First, by limiting the trusted computing base (i.e., the scripting functionality supported by the underlying blockchain), we reduce the on-chain complexity of the virtual channel protocol. As bugs in smart contracts are manifold and notoriously hard to fix, our construction eliminates an additional attack vector by moving the complexity to the protocol level (rather than on-chain as in the construction from [13]). Second, investigating the minimal functionality that is required by the underlying ledger to support complex protocols is scientifically interesting. One may view this as a more general research direction of building a lambda calculus for off-chain protocols. Concretely, our construction shows that virtual channels can be built with stateless scripts, while earlier constructions required stateful on-chain computation. Finally, from a practical perspective, our construction can be integrated into the Lightning Network (the by far most prominent PCN), and thus our solution can offer the benefits of virtual payment channels/hubs to a broad user base.

A. Our contributions

In this work, we develop the *first* protocols for building virtual channel hubs over cryptocurrencies that support limited scripting functionality. Our construction requires only digital signatures and timelocks, which are ubiquitously available in cryptocurrencies and well characterized. We also provide a comprehensive formal analysis of our constructions and benchmarks of a prototype implementation. Concretely, our contributions are summarized below.

- We present the first protocols for virtual channel hubs that are built for the UTXO-model and require a scripting language supporting only digital signature verification and timelock functionality, being thus 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), 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 [8], and prove that our protocols constitute a secure realization thereof. Since our virtual channels are built in the UTXO-model, our ideal functionality and formalization significantly differs from earlier work [13].
- We evaluate our protocol over two different PCN constructions, the Lightning Network (LN) [26] and Generalized channels (GC) [2], which extend LN channels to support functionality other than one-to-one payments. We show that for virtual channels on top of GC, n sequential payment operations require an off-chain exchange of $9 + 2 \cdot n$ transactions or a total of $3524 + 695 \cdot n$ bytes, as compared to $8 \cdot n$ transactions or $3026 \cdot n$ bytes when Ingrid routes the payment actively through the PCN. This means a virtual channel is already cheaper if two or more sequential payments are performed. For virtual channels over LN, n transactions require an off-chain exchange of $6292 + 2824 \cdot n$ bytes, compared to $4776 \cdot n$ bytes when routed through an intermediary. We have interacted with the Bitcoin blockchain to store the required transactions, demonstrating the compatibility of our protocol.

To summarize, for the first time in Bitcoin, we enable off-chain payments between users connected by payment channels via a hub without requiring the continuous presence of any intermediary. Hence, our solution increases the reliability and, at the same time, reduces the latency and costs of Bitcoin PCNs.

II. Background

In this section, we first introduce notation and preliminaries on UTXO-based blockchains. We then overview the basics of payment and virtual channels, referring the reader to [1, 22, 21, 13] for further details. We finally discuss the main technical challenges one needs to overcome when constructing Bitcoin-compatible virtual channels.

A. UTXO-based blockchains

We adopt the notation for UTXO-based blockchains from [2], which we shortly review below.

a) Attribute tuples: Let T be a tuple of values, which we call in the following *attributes*. Each attribute in T is identified by a unique keyword, e.g., attr and referred to as T.attr.

b) 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 with 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 set of operations (also called scripts) 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 $\mathsf{tx.txid} := \mathcal{H}([\mathsf{tx}])$, where $\mathcal{H}$ is a hash function which is modeled as a random oracle and $[\mathsf{tx}]$ is the *body of the transaction* defined as $[\mathsf{tx}] := (\mathsf{tx.Input}, \mathsf{tx.Output}, \mathsf{tx.TimeLock})$. 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 $\mathsf{tx.TimeLock} \in \mathbb{N} \cup \{0\}$ 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, $\mathsf{tx.Witness} \in \{0,1\}^*$ 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. 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.

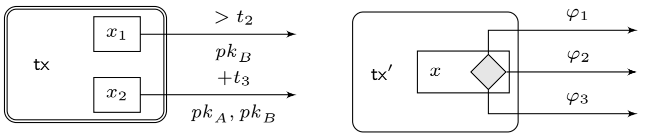

Fig. 1: (Left) Transaction tx is published on the blockchain. The output of value x_1 can be spent by a transaction signed w.r.t. pk_B after round t_2, and the output of value x_2 can be spent by a transaction signed w.r.t. pk_A and pk_B but only if at least t_3 rounds passed since tx was accepted by 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 $\varphi_1 \vee \varphi_2 \vee \varphi_3$.

In order to be concise, we use the following abbreviations for the frequently used conditions. Most outputs can only be spent by a transaction that 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. We use the command `One−Sig` and `Multi−Sig` in the pseudocode. Other additional spending conditions are written *above* the arrow. 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 (and `CheckRelative` in the pseudocode). 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.

B. Payment channels

A payment channel enables arbitrarily many transactions between users while requiring only two on-chain transactions. The first step when creating a payment channel is to deposit coins into an output controlled by two users. Once the money is deposited, the users can authorize new balance updates 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 [16, 2] 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 $\mathtt{TX_f}$) 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 $\mathtt{TX_f}$ 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* $\mathtt{TX_c}$ representing the commitment from both users to the new balance of the channel. The commit transaction spends the output of $\mathtt{TX_f}$ 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 signatures on the commit transaction, which serve as valid witnesses for $\mathtt{TX_f}$. At this point, Alice (resp. Bob) could add $\mathtt{TX_c}$ to the blockchain. Instead, they keep it locally in their memory and overwrite it when they agree on another commitment transaction $\overline{\mathtt{TX}}_\mathtt{c}$ representing a newer balance of the channel. This, however, leads to the problem that there exist 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 outdated commit transaction, payment channels require a mechanism that punishes such malicious behavior. This 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 outdated 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 $\overline{\mathtt{TX}}_\mathtt{c}$ that they have agreed on to the blockchain. After it is accepted, the coins initially locked at the channel creation via $\mathtt{TX_f}$ are redistributed to both users according to the last agreed balance. As aforementioned, if one of the users submits an outdated commitment transaction instead, the counterparty can punish the former through the revocation mechanism.

The Lightning Network [26] defines the state-of-the-art payment channel construction for Bitcoin.

C. Generalized channels

The recent work of Aumayr et al. [2] proposes the concept of *generalized channels*. Generalized channels improve and extend payment channels (see Figure 2 for details) in two ways. First, they extend the functionality of payment channels by offering off-chain execution of any script that is supported by the underlying ledger. Hence, one may view generalized channels as state channels for blockchains with restricted scripting functionality. Second, and more important for our work, generalized channels significantly improve the on-chain and off-chain communication complexity. More concretely, this efficiency improvement is achieved by introducing a so-called *split transaction* (that we denote as $\mathtt{TX_s}$) along with a *punish-then-split* paradigm. In contrast to regular payment channels that require one revocation process per output in the commit transaction, the punish-then-split approach decouples the revocation process from the number of outputs in the

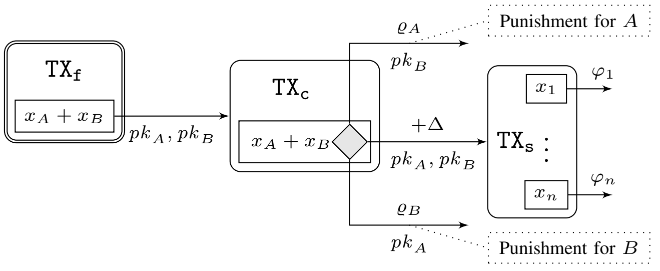

Fig. 2: A generalized channel in the state $((x_1, \varphi_1), \ldots, (x_n, \varphi_n))$. The value of Δ upper bounds the time needed to publish a transaction on a blockchain. The condition ϱ_A represents the verification of A' revocation secret and ϱ_B represents the verification of B' revocation secret.

commit transaction. This allows moving from revocation for each output to a single revocation for the entire channel. As shown in Figure 2, the commit transaction ($\mathtt{TX_c}$) is only responsible for the punishment, while the split transaction ($\mathtt{TX_s}$) holds the actual outputs of the channel.

The efficiency of generalized channels is further improved since they only require a single commit transaction per channel. This is in contrast to the payment channels used by Lightning, which require two distinct commit transactions for each channel user. We will discuss in Section III-D3 why the punish-then-split paradigm (and requiring only one commit transaction) is useful in order to improve the efficiency of our virtual channels for Bitcoin.

To simplify terminology, we will use the term *ledger channel* for all channels that are funded directly over the blockchain.

D. Channel Networks

The aforementioned payment and generalized channels allow two parties to issue transactions between each other while having to communicate with the blockchain only during the creation and closure of the channel. This on-chain communication can further be reduced by using *channel networks*.

a) Payment Channel Networks (PCNs): A PCN is a protocol that allows parties to connect multiple ledger channels to form a payment channel network. In this network, a sender can route a payment to a receiver as long as both parties are connected by a path in the network. Suppose that Alice and Bob are not directly connected via a ledger channel, but instead both maintain a channel with an intermediary party (Ingrid). In a nutshell, Alice can pay Bob by sending her coins to Ingrid who then forwards them in her ledger channel to Bob. Importantly, the protocol must achieve atomicity, i.e., either both transfers from Alice to Ingrid and from Ingrid to Bob happen, or neither of them goes through. Current PCNs such as the Lightning network use the HTLC-technique (hash-time-lock transaction), which comes with several drawbacks as mentioned in the introduction: (i) *low reliability* because the success of payments relies on Ingrid being online; (ii) *high latency* as each payment must be routed through Ingrid;

$$\begin{tikzcd}
\text{Alice} \arrow[r, Leftrightarrow, "\alpha"'] \arrow[rr, leftrightarrow, bend left, "\gamma"] & \text{Ingrid} \arrow[r, Leftrightarrow, "\beta"'] & \text{Bob}
\end{tikzcd}$$

Fig. 3: A virtual channel γ built over ledger channels α, β.

(iii) *high-cost* as Ingrid may charge a fee for each payment between Alice and Bob; and (iv) *low privacy* as Ingrid can observe each payment that happens between Alice and Bob. To mitigate these issues, virtual channels have been proposed.

b) Virtual Channels: An alternative solution to connect two payment channels with each other is offered by the concept of *virtual channels* [13]. Virtual channels allow Alice and Bob to send payments between each other without the involvement of the intermediary Ingrid. In some sense, they thus mimic the functionality offered by ledger channels, with the difference that they are not created directly over the blockchain but instead over two ledger channels. More concretely, as shown in Figure 3, a virtual channel γ between Alice and Bob with intermediary Ingrid is constructed on top of two ledger channels α and β. Ingrid is required to participate in the initial creation and final closing of the virtual channel. But importantly, Ingrid is not involved in any balance updates that occur in the virtual channel. This overcomes the four drawbacks mentioned above. While these advantages over PCNs make virtual channels an attractive off-chain solution, their design is far from trivial. Previous work showed how to construct virtual channels over a ledger that supports Turing complete smart contracts [13, 11, 12]. The smart contract acts in the protocol as a trust anchor that parties can fall back to in case of malicious behavior. Through a rather complex protocol and careful smart contract design, existing virtual channel constructions guarantee that honest parties in the virtual channel will always get the coins they rightfully own. Unfortunately, most cryptocurrencies (including Bitcoin) do not offer Turing complete smart contracts, and hence the constructions from prior work cannot be used. In this work, we present a novel construction of virtual channels that makes only minimal assumptions on the underlying scripting functionality offered by the ledger.

III. Virtual Channels

In this section, we first give some notation before presenting the necessary properties for virtual channels and discussing design challenges. Finally, we present our protocol.

A. Definitions

We briefly recall some notation and definition for generalized channels [2] and extend the definition to generalized virtual channels. In order to make the distinction between the two types of channels clearer, we call the former generalized *ledger* channel (or ledger channels for short).

A *generalized ledger channel* as defined in [2] is 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 for short virtual channel) is defined as a tuple $\gamma := (\gamma.\mathsf{id}, \gamma.\mathsf{Alice}, \gamma.\mathsf{Bob}, \gamma.\mathsf{cash}, \gamma.\mathsf{st}, \gamma.\mathsf{Ingrid}, \gamma.\mathsf{subchan}, \gamma.\mathsf{fee}, \gamma.\mathsf{val})$. The attributes $\gamma.\mathsf{id}, \gamma.\mathsf{Alice}, \gamma.\mathsf{Bob}, \gamma.\mathsf{cash}, \gamma.\mathsf{st}$ are defined as in the case of ledger channels. The additional attribute $\gamma.\mathsf{Ingrid} \in \mathcal{P}$ denotes the identity of the *intermediary* of the virtual channel γ. The set $\gamma.\mathsf{endUsers}$ and the function $\gamma.\mathsf{otherParty}$ are defined as before. Additionally, we also define the set $\gamma.\mathsf{users} := \{\gamma.\mathsf{Alice}, \gamma.\mathsf{Bob}, \gamma.\mathsf{Ingrid}\}$. The attribute $\gamma.\mathsf{subchan}$ is a function mapping $\gamma.\mathsf{endUsers}$ to a channel identifier; namely, the value $\gamma.\mathsf{subchan}(\gamma.\mathsf{Alice})$ refers to the identifier of the channel between $\gamma.\mathsf{Alice}$ and $\gamma.\mathsf{Ingrid}$ (i.e., the id of α from the description above); similarly, the value $\gamma.\mathsf{subchan}(\gamma.\mathsf{Bob})$ refers to the identifier of the channel between $\gamma.\mathsf{Bob}$ and $\gamma.\mathsf{Ingrid}$ (i.e., β from the description above). The value $\gamma.\mathsf{fee} \in \mathbb{R}^{\geq 0}$ represents the fee charged by $\gamma.\mathsf{Ingrid}$ for her service of being an intermediary of γ. Finally, we introduce the attribute $\gamma.\mathsf{val} \in \mathbb{N} \cup \{\bot\}$. If $\gamma.\mathsf{val} \neq \bot$, then we call γ a *virtual channel with validity* and the value of $\gamma.\mathsf{val}$ represents the round number until which γ remains open. Channels with $\gamma.\mathsf{val} = \bot$ are called *virtual channels without validity*.

B. Security and efficiency goals

We briefly recall the properties of generalized channels as defined in [2] and state the additional properties that we require from virtual channels.

a) 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 $\gamma.\mathsf{st}$ and $\gamma.\mathsf{st}$ is the last state that the channel is updated to, then an honest party $P \in \gamma.\mathsf{endUsers}$ can either enforce this state on the ledger or P can enforce a state where she gets all the coins locked in the channel. We say that a state st is *enforced* when 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 two subtle but important differences: (i) 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; (ii) the finality (i.e., offloading) of the virtual channel depends on whether Alice is expected to offload the virtual channel within a predetermined validity period (virtual channel with validity VC-V) or the offload task is delegated to the intermediary Ingrid without having a predefined validity period (virtual channel without validity VC-NV). In order to account for these two differences, 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 can transform γ to a ledger channel. Party $P \in \gamma$.endUsers can initiate the transformation which either completes or P can get financially compensated.

(V3) Validity with punish: If γ is a virtual channel *with* validity (VC-V), then γ.Alice can transform γ to a ledger channel. If γ is not transformed into a ledger channel or closed before time γ.val, γ.Ingrid and γ.Bob can get financially compensated.

We first note that the instant finality with punish property (S3) does not provide any guarantees for Ingrid $\notin \gamma$.endUsers, which is why we need to define (V1) for virtual channels. Properties (V2) and (V3) point out the main difference between VC-NV and VC-V. 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. Furthermore, in case Alice and Bob transform the virtual channel to a ledger channel or even misbehave, honest Ingrid is guaranteed that she will receive the collateral back. In a VC-V γ, Ingrid cannot transform a virtual channel into a ledger channel at any time she wants. Instead, there is a pre-agreed point in time, defined by γ.val, until when γ.endUsers have to close the virtual channel or transform it into a ledger channel (Ingrid's collateral is freed in both cases). If γ.endUsers fail to do so, Ingrid can get her collateral back through a punishment mechanism. Hence, γ.endUsers have a guarantee that their VC-V will remain a virtual channel until a certain round, after which they must ensure its closure or transformation to avoid punishments.

b) Efficiency goals: Lastly, we define the following efficiency goals, which describe the number of rounds certain protocol steps require:

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

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

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

(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 a 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 a 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 I. We formalize these properties as a UC ideal functionality in Appendix D1.

C. Design Challenges for Constructing Virtual Channels

The main challenges that arise when constructing Bitcoin-compatible virtual channels stem from the need to ensure the security properties (V1) - (V3) as presented in the previous section. Namely, to guarantee balance security to the intermediary, we need to ensure that the virtual channel creation and closure is reflected symmetrically and synchronously on both underlying ledger channels. We identify this as a challenge (C1). As we discuss in more detail below, this can be solved by giving the intermediary the right of a "last say" in the virtual channel creation and closure procedures. However, a malicious intermediary could abuse such power and block virtual channel closure indefinitely. Therefore, the second challenge (C2) is to design a punishment mechanism that allows virtual channel users to either enforce closure or claim financial compensation. We provide some further details below.

a) Synchronous create and close (C1): The creation and closure of a virtual channel are done by updating the underlying ledger channels. In order to guarantee balance security for the intermediary, we must ensure that updates on both ledger channels are symmetric and either both of them succeed or both of them fail. That is, if the intermediary Ingrid loses coins in one ledger channel as a result of the virtual channel construction, then she has the guarantee of gaining the same amount of coins from the other ledger channel. Such an atomicity property can be achieved by allowing Ingrid to be the reacting party in both ledger channel update procedures. Namely, Ingrid has to receive symmetric update requests from both Alice and Bob before she confirms either of them.

As a result, Ingrid has the power to block a virtual channel creation and closure. For a virtual channel creation, this is not a problem. It simply represents the fact that Ingrid does not want to be an intermediary, and hence Alice and Bob have to find a different party. However, for virtual channel closing, this power of the intermediary results in a violation of the instant finality property for Alice and Bob, and requires a more involved mechanism.

b) Enforcing virtual channel state (C2): In contrast to standard ledger channels that rely on funding transactions that are published on the ledger, the funding transactions of a virtual channel are, in the optimistic case (i.e., when parties are honest), kept off-chain. In case of misbehavior (e.g., when malicious Ingrid refuses to close the virtual channel), however, honest parties must be able to publish the virtual channel funding transaction to the blockchain in order to enforce the latest state of the virtual channel. Unfortunately, the funding transactions can only be published if *both* of the underlying channels are closed in a state which funds the virtual channel. The fact that the virtual channel participants, Alice and Bob, respectively have control over just one of the underlying ledger channels further complicates this situation. For instance, one of the underlying ledger channels may be updated or closed maliciously at any time which would prevent the publishing of the funding transaction on the ledger.

D. Virtual Channel Protocol

We now show how to build virtual channels on top of generalized channels. We later discuss in Section III-D3 how our construction can be built over other channels such as Lightning and why generalized channels offer better efficiency.

As mentioned in the previous section, virtual channels are created and closed through an update of the underlying ledger channels. Hence, let us recall the update process of ledger channels, depicted as UpdateChan in Figures 4 and 5, before explaining our construction in more detail. The update procedure consists of 4 steps, namely (1) the *Initialization* step, during which parties agree on the new state of the channel, (2) the *Preparation* step, where parties generate the transactions with the given state, (3) the *Setup* during which parties exchange their application-dependent data (e.g., for building virtual channels), and finally (4) the *Completion* step where parties commit to the new state and revoke the old one. We refer the reader to [2] for more details.

1) High level protocol description: We are now prepared to present a high-level description of our modular virtual channel protocol and explain how to solve the main technical challenges when designing virtual channels. In a nutshell, this modular protocol gives a generic framework on how to design virtual channels. Afterwards, we show how to instantiate this modular protocol with our virtual channel construction without validity. For the description of the instantiation with our construction with validity, we refer the reader to Appendix D2. We present the formal pseudocode for the modular protocol as well as the instantiations with and without validity in Appendix D3.

a) Create: Let γ be a virtual channel that $A := \gamma.\mathsf{Alice}$ and $B := \gamma.\mathsf{Bob}$ want to create, using their generalized ledger channels with $I := \gamma.\mathsf{Ingrid}$. At a high level, the creation procedure of a virtual channel is a synchronous update of the underlying ledger channels. Given the ledger channels, 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 parties use the identifiers

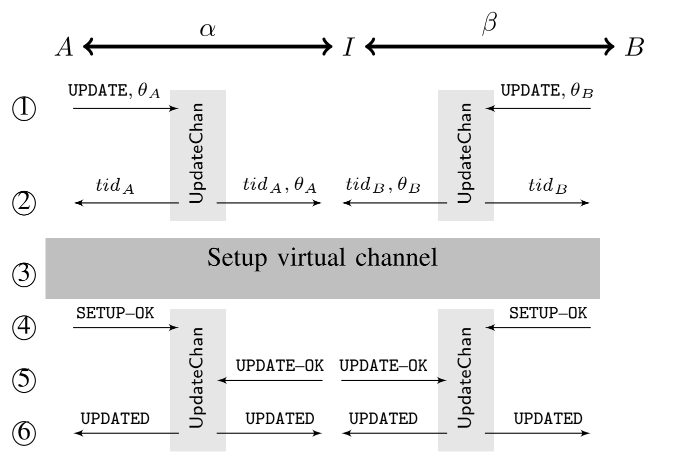

Fig. 4: Modular creation procedure of a virtual channel on top of two ledger channels α and β.

tid_A and tid_B of their subchannels in order to build the virtual channel (step ②). Next, all three parties engage in a setup phase, in which the structure of the virtual channel is built (step ③). More concretely, all three parties agree on a funding transaction of the virtual channel which when published on the blockchain transforms the virtual channel to a ledger channel. 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 crucial 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 the final update agreement (step ⑤). Upon a successful execution, parties consider the channels as updated (step ⑥), which implies that the virtual channel γ was successfully created.

b) Update: Updating the virtual channel essentially works in the same way as the update procedure of a ledger channel. As long as the update is successful or peacefully rejected (meaning that the reacting party rejects the update), the parties act as instructed in the ledger channel protocol. The situation is more delicate when the update fails because one of the parties misbehaved and aborted the procedure.

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 the channel but the other one did not. In a standard ledger channel, these disputes are resolved by a force close procedure, meaning that the honest party publishes the latest valid state on the blockchain, thereby forcefully closing the channel. 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 to the ledger. In other words, we first need to transform a virtual channel into a ledger channel by publishing its funding transaction on-chain. This process is discussed later in this section. Once the funding transaction is published, the dispute is handled in the same way as for ledger channels.

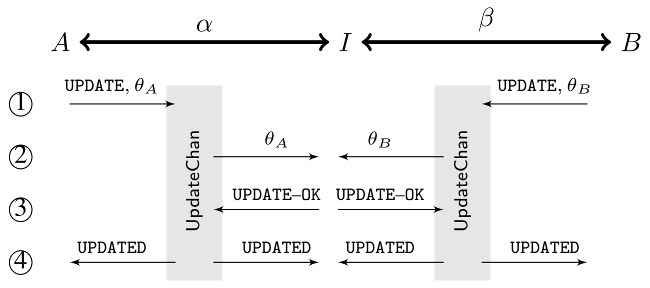

Fig. 5: Modular close procedure of a virtual channel on top of two ledger channels α and β. For $P \in \{A, B\}$, $\vec{\theta}_P := \{(c_P, \texttt{One–Sig}_{pk_P}), (c_Q + \frac{\gamma.\mathsf{fee}}{2}, \texttt{One–Sig}_{pk_I})\}$ where $\gamma.\mathsf{st} = \Big((c_P, \texttt{One–Sig}_{pk_P}), (c_Q, \texttt{One–Sig}_{pk_Q})\Big)$.

c) Close: The closure of a virtual channel is done by updating the underlying ledger channels α and β according to the latest state of the virtual channel $\gamma.\mathsf{st}$. To this end, each party $P \in \{A, B\}$ computes the new state for the ledger channel $\vec{\theta}_P := \{(c_P, \texttt{One–Sig}_{pk_P}), (\gamma.\mathsf{cash} - c_P, \texttt{One–Sig}_{pk_I})\}$ where c_P is the latest balance of P in γ. All parties update their ledger channels according to this state.

In a bit more detail, the closing procedure of a virtual channel proceeds as follows (see Figure 5). Each party P initiates an update of the underlying ledger channel with state $\vec{\theta}_P$ (step ①). Since both ledger channels must be updated synchronously, I waits for both parties to initiate the update procedure. Upon receiving the states from both parties (step ②), I checks that the states are consistent and if so, she agrees to the update of both ledger channels (step ③). Finally, after all parties have successfully revoked the previous ledger channel state, the virtual channel is considered to be closed.

In the pessimistic case (if the states $\vec{\theta}_A$ and $\vec{\theta}_B$ are inconsistent, revocation fails or I remains idle), parties must forcefully close their virtual channel by publishing the funding transaction (offloading) and closing the resulting ledger channel. This, together with the fact that I plays the role of the reacting party in its interactions with A and B, addresses the challenge (C1) as mentioned in Section III-C.

d) Offload: During the offload procedure, parties try to publish the funding transaction of the virtual channel γ which effectively transforms the virtual channel into a ledger channel. In a nutshell, during this procedure, parties try to publish the commit and split transactions of both underlying ledger channels and afterward the funding transaction of the virtual channel. In case offloading is prevented by some form of malicious behavior, parties can engage in the punishment procedure to ensure that they do not lose any funds.

e) Punish: The concept of punishment in virtual channels is similar to that in ledger channels; namely in case that the latest state of a 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, the virtual channel is transformed into a ledger channel and parties can execute the regular punishment protocol for ledger channels. In addition to the ledger channel's punishment procedure, parties can punish if the funding transaction of γ cannot be published. Since this punishment, however, differs for each concrete instantiation, we will explain it in more detail for our protocol without validity in the following section (and in Appendix D2 for the case with validity).

The offloading and punishment procedure together tackles challenge (C2) from Section III-C.

2) Concrete Instantiation Without Validity: We now describe how the modular protocol explained above can be concretely instantiated with our construction for virtual channels without validity.

a) Create: In our construction without validity, A and B must "prepare" the virtual channel during the setup procedure (step ③ in create of the modular protocol). This is done by executing the creation procedure of a regular ledger channel, i.e., they create a funding transaction with inputs tid_A and tid_B, as well as a commit and split transactions that spend the funding transaction. Once all three transactions are created, A and B sign them and exchange their signatures. Note that this corresponds to a normal channel opening, with the mere difference that the funding transaction is not published to the blockchain. In order to complete the virtual channel setup, A and B send the signed funding transaction to I who, upon receiving both signatures, sends her own signature on the transaction back to A and B. At this stage, the virtual channel is prepared, however, the creation is not completed yet. In order to finish the creation procedure, A, I, and B have to finish the update of their respective ledger channels. Once this is done, the virtual channel has been successfully created.

We illustrate the transaction structure prepared during the creation process in Figure 6. The funding transaction of the virtual channel $\texttt{TX}_\texttt{f}$, which is generated during the create procedure, takes as input coins from both, the ledger channel α (represented by $\texttt{TX}^\texttt{A}_\texttt{s}$) and the ledger channel β (represented by $\texttt{TX}^\texttt{B}_\texttt{s}$). Both ledger channels jointly contribute a total of $2c + f$ coins so that c coins are later used to setup the virtual channel and the remaining $c + f$ coins are I's collateral and the fees paid to I for providing the service for A and B.[1] I's collateral and fees in the funding transaction $\texttt{TX}_\texttt{f}$ are the reason why I has to proactively monitor the virtual channel as she has an incentive to publish $\texttt{TX}_\texttt{f}$ in case any party misbehaves.

b) Offload: I is always able to offload the virtual channel by herself (i.e., without having to cooperate with another party) which guarantees that I can redeem her collateral at any 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

[1]For simplicity we assume each of the parties contributes $f/2$ coins to I's total fees in addition to $c/2$ coins for funding the virtual channel.

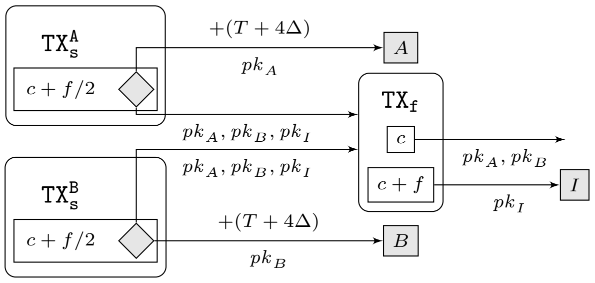

Fig. 6: Funding of a virtual channel γ without validity. T upper bounds the number of off-chain communication rounds between two parties for any operation in the ledger channel.

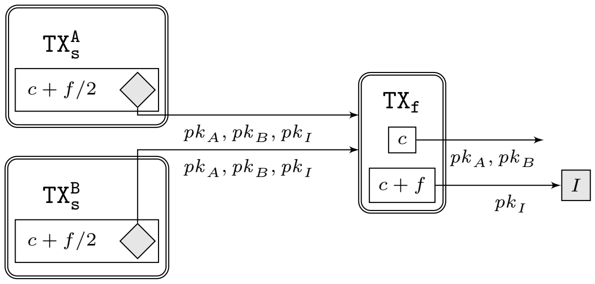

Fig. 7: Transactions published after a successful offload.

transactions of the respective other ledger channel, since I loses her collateral to P otherwise.

More precisely, if I wishes to offload the virtual channel γ and retrieve her collateral and fees, she can close both of her ledger channels with A and B (i.e., α and β) and publish the funding transaction of the virtual channel i.e., $\mathtt{TX_f}$. This is possible as I is part of both ledger channels. A or B, on the other hand, are respectively part of only one ledger channel and hence they cannot offload the virtual channel individually. However, they can force I to offload by publishing the commit and split transactions of their respective channel with I (we will elaborate on this in the description of the punishment mechanism). Figure 7 illustrates the transactions that are posted on the blockchain in case of a successful offload. The figure shows that the split transactions of both underlying ledger channels have to be published such that eventually the funding transaction of the virtual channel can be published which completes the offloading procedure.

c) Punish: 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. 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. Furthermore, upon one of the subchannels being closed, I must close the other subchannel in order to guarantee that both parties can post $\mathtt{TX_f}$.

Let us now get into more details. Assume that A's ledger channel with I is closed, but the funding transaction $\mathtt{TX_f}$ cannot be published on the blockchain. This means that I's channel with B (i.e., β) is still open or has been closed in a different state such that $\mathtt{TX_f}$ cannot be published. In other words, Ingrid acted maliciously by wrongfully closing β in a different state or by not closing β at all. In this case, A must be able to get all the coins from her channel with Ingrid. This punishment works as follows: After A publishing the split transaction of α, I is given a certain time period to close her channel with B and publish the virtual channel's funding transaction $\mathtt{TX_f}$. If I fails to do so in the prescribed time period, A receives all coins in her channel with I.

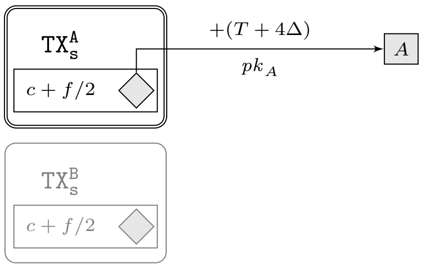

Fig. 8: Transactions published after A successfully executed the punishment procedure. The grayed transaction $\mathtt{TX_s^B}$ indicates that this transaction has not been published.

We note that in this scenario, B (instead of I) might have been the malicious party by closing β in an outdated state, thereby leaving I no option to publish $\mathtt{TX_f}$. However, in this case, I can punish B via the punishment mechanism of the underlying ledger channel and earn all the coins in β. Therefore, I will remain financially neutral as she gets punished by A but simultaneously compensated by B. Figure 8 illustrates the transactions that are posted on the blockchain in the case of A successfully executing the punishment mechanism. The case where B executes the punishment mechanism is analogous.

3) Further discussion regarding our constructions: In the following, we present further considerations regarding our protocol, including remarks on concurrency, a discussion on how the protocol can be built on top of Lightning channels, and a brief description of our virtual channel construction with validity that we detail in Appendix D2.

a) Concurrency: When creating a virtual channel, we need to lock the underlying ledger channels α and β (i.e., no further updates can be made on the ledger channels as long as the virtual channel is open). This, however, is undesirable, because in most cases the ledger channels will have more coins available than what is needed for funding the virtual channel. We emphasize that this issue can be easily addressed (and hence supporting full concurrency) by using the channel splitting technique discussed in [2]. 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 that can be used in the underlying ledger channel.

b) Virtual channels over Lightning: We will now discuss how our virtual channel constructions can be built on top of any ledger channel infrastructure that uses a *revocation/punishment* mechanism such as the Lightning Network

[26]. The main complication arises from the fact that ledger channel constructions other than generalized channels require two commit transactions per channel state (one for each party). As depicted in Figure 9 (and unlike generalized channels in Figure 2), Alice and Bob each have a commit transaction $\mathtt{TX}_\mathtt{c}^\mathtt{A}$ and $\mathtt{TX}_\mathtt{c}^\mathtt{B}$ which spends the funding transaction $\mathtt{TX}_\mathtt{f}$ and distributes the coins. Therefore, in such channel constructions, it is a priori unclear which of these commit transactions will be posted and accepted on the blockchain (note that only one of them can be successfully published) and hence building applications (e.g., virtual channels) on top of such ledger channels becomes complex.

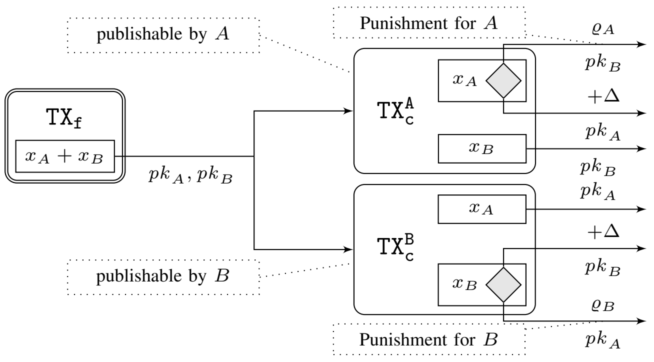

Fig. 9: A Lightning style payment channel where A has x_A coins and B has x_B coins. Δ upper bounds the time needed to publish a transaction on a blockchain. condition ϱ_A represents the verification of A' revocation secret and h represents the verification of B' revocation secret.

In more detail, assume Alice and Bob want to build a virtual channel γ on top of their respective Lightning ledger channels with Ingrid, where both ledger channels consist of two commit transactions respectively (i.e., $(\mathtt{TX}_\mathtt{c}^\mathtt{A}, \mathtt{TX}_\mathtt{c}^\mathtt{IA})$ for the channel between Alice and Ingrid and $(\mathtt{TX}_\mathtt{c}^\mathtt{B}, \mathtt{TX}_\mathtt{c}^\mathtt{IB})$ for the channel between Bob and Ingrid). All three parties now have to make sure that the virtual channel can be funded (i.e., that the funding transaction of γ can be published to the blockchain) even in case of malicious behavior. To ensure this, parties have to prepare the funding transaction of γ with respect to all possible combinations of the commit transactions of the respective underlying ledger channels. Since there are four such combinations ($(\mathtt{TX}_\mathtt{c}^\mathtt{A}, \mathtt{TX}_\mathtt{c}^\mathtt{B})$, $(\mathtt{TX}_\mathtt{c}^\mathtt{A}, \mathtt{TX}_\mathtt{c}^\mathtt{IB})$, $(\mathtt{TX}_\mathtt{c}^\mathtt{IA}, \mathtt{TX}_\mathtt{c}^\mathtt{B})$ and $(\mathtt{TX}_\mathtt{c}^\mathtt{IA}, \mathtt{TX}_\mathtt{c}^\mathtt{IB})$), parties have to prepare four funding transactions for γ. Hence, updating such a virtual channel requires repeating the update procedure for all four funding transactions.

As generalized channels require only a single commit transaction per channel state building virtual channels on top of generalized channels offers a significant efficiency improvement in terms of off-chain communication complexity (see Section V for the detailed comparison).

c) Virtual Channels With Validity: Note that so far we described our protocol without validity where the virtual channel can be offloaded by the intermediary whenever she wants. The drawback of this construction is that Ingrid needs to be proactive during the lifetime of the virtual channel, i.e., she has to constantly monitor the channel for potential misbehavior of Alice or Bob. This might be undesirable in scenarios where Ingrid plays the role of the intermediary in not just one but many different virtual channels at the same time (e.g., if Ingrid is a channel hub). For this reason, we developed an alternative solution which we call virtual channels with validity. In this solution, each virtual channel has a predetermined time (which we call validity) which indicates until when the channel has to be closed again. If the channel is still open after this time, Ingrid has to become proactive in order to receive her collateral back. The obvious advantage of this approach is that Ingrid can remain inactive until the validity of a channel expires. The details of this protocol can be found in Appendix D2.

IV. Security Model and Analysis

In order to model and prove the security of our virtual channel protocols, we use the global UC framework (GUC) [9] as in [2]. This framework allows for a global setup which we utilize to model a public blockchain. More precisely, our protocol uses a global ledger functionality $\widehat{\mathcal{L}}(\Delta, \Sigma)$, where Δ upper bounds the blockchain delay, i.e., the maximum number of rounds required to publish a transaction, and Σ is the signature scheme used by the blockchain. In this section, we only give a high-level idea behind our security analysis in the UC framework and refer the reader to Appendices A, C, C2 and D1 for more details.

As a first step, we define the expected behavior of a virtual channel protocol in the form of an *ideal functionality* $\mathcal{F}_V$. The functionality defines the input/output behavior of a protocol, its impact on the global setup (e.g., ledger) and the possible ways an adversary can influence its execution (e.g., delaying messages). In order to prove that a concrete protocol is a secure virtual channel protocol, one must show that the protocol *emulates* the ideal functionality $\mathcal{F}_V$. This means that any attack that can be mounted on the protocol can also be mounted on the ideal functionality, hence the protocol is at least as secure as the ideal specification given by $\mathcal{F}_V$.

The proof of emulation consists of two steps. First, one must design a *simulator*, which simulates the actions of an adversary on the real-world protocol by interacting with the ideal functionality. Second, it must be shown that the execution of the real-world protocol being attacked by a real-world adversary is indistinguishable from the execution of the ideal functionality communicating with the constructed simulator. In UC, the ppt distinguisher who tries to distinguish these two executions is called the *environment*.

The main challenge when designing a simulator is to make sure that the environment sees transactions being posted on the ledger in the same round in both worlds. In addition, our simulator needs to ensure that the ideal functionality outputs the same set of messages in the same round as the protocol. We reduce the indistinguishability of the two executions to the security of the cryptographic primitives used in our protocol.

One of the advantages of using UC is its composability. In other words, one can use an ideal functionality in a black-box way in other protocols. This simplifies the process of designing new protocols as it allows to reuse existing results and enables modular protocol designs. We utilize this nice property of the UC framework and use the ideal functionality of the generalized channel from [2] when designing our virtual channel protocol.

We only mention the main security theorem here and provide a high-level proof sketch here. We refer the reader to Appendix F for the full proof.

Theorem 1: Let Σ be a signature scheme that is strongly unforgeable against chosen message attacks. Then for any ledger delay $\Delta \in \mathbb{N}$, the virtual channel protocol without validity as described in Section III-D working in $\mathcal{F}_{preL}(3, 1)$-hybrid, UC-realizes the ideal functionality $\mathcal{F}_V(3)$.

We now give a proof sketch to show that the two properties (V1) Balance security and (V2) Offload with punish hold for honest parties. To this end, we analyze all possible cases in which the underlying ledger channels are maliciously closed, i.e., the cases when the virtual channel cannot be offloaded anymore. Note that if the virtual channel is offloaded, it is effectively transformed into a generalized ledger channel and satisfies the security properties of generalized channels.

If all parties behave honestly (V1) and (V2) hold trivially as I is always able to offload the virtual channel by publishing all transactions $\mathtt{TX}_\mathtt{s}^\mathtt{A}$, $\mathtt{TX}_\mathtt{s}^\mathtt{B}$ and $\mathtt{TX}_\mathtt{f}$. Furthermore, neither A nor B would ever lose their coins. Now consider the case where one of the underlying channels, e.g., the channel between B and I is closed in a different state such that $\mathtt{TX}_\mathtt{f}$ cannot be posted on the blockchain anymore (the case for the channel between A and I is analogous). As an honest A would not update her channel with I as long as the virtual channel is open, there are only two possible situations: (i) A is able to post $\mathtt{TX}_\mathtt{s}^\mathtt{A}$ which allows her to punish I (see Figure 8), or (ii) I has maliciously closed her channel with A in an outdated and revoked state. In this case, A is able to punish I according to property (S3), i.e., instant finality with punish, of the underlying ledger channel (see Section II and Figure 2 for more details on the punishment of the underlying channel). Therefore, (V2) is satisfied for A, since she can punish I and get financially compensated. Now let us analyze the maliciously closed channel between B and I, let us denote it β. If both parties are malicious, we do not need to prove anything as (V1) and (V2) should only hold for honest parties. In case B is honest, I must have closed β in an old state which would allow B to punish I. Hence (V2) holds and we do not need to prove (V1) as I is malicious. Analogously, if I is honest, malicious B must have closed β in an old state and hence I can punish B. Hence (V1) holds and we do not need to prove (V2) for malicious B). Hence, (V1) and (V2) hold for all honest parties.

V. 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) built on top of both generalized channels as well as Lightning channels and compare them. Finally, we evaluate the advantages of virtual channels over ledger channels in terms of routing communication overhead and fee costs. As testbed [6], 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.

A. 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 18803 USD per Bitcoin, and the average transaction fee of 104 satoshis per byte all of them at the time of writing. We detail in Table II the aforementioned costs measured for both virtual channel constructions building on top of generalized channels and on top of Lightning channels.

Perhaps the most relevant difference to ledger channels in practice is, in the CREATE and the optimistic CLOSE case, we do not have any on-chain transactions. This implies no on-chain fees for the opening and closing of virtual channels.

a) Virtual channels over generalized channels: For the creation of a virtual channel (CREATE operation) on top of generalized channels, 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 640 bytes for $\mathtt{TX}_\mathtt{f}$ (VC-NV) or 309 + 377 bytes for $\mathtt{TX}_\mathtt{f}$ and $\mathtt{TX}_\mathtt{refund}$ (VC-V). Finally, for both VC-NV and VC-V, we need the transactions representing the state of the the virtual channel itself which requires 431 bytes for $\mathtt{TX}_\mathtt{c}$ and 264 bytes for $\mathtt{TX}_\mathtt{s}$. 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, both on-chain. 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.

b) Virtual channels over Lightning channels: Building virtual channels on top of Lightning channels yields the following results. Instead of one commitment and one split transaction per ledger channel, we now need two commitment transactions per ledger channel, each of size 580 (VC-NV) or 546 (VC-V) bytes. Due to the fact that in both ledger channels, either commitment transaction can be published, we now need four $\mathtt{TX}_\mathtt{f}$ of 640 bytes each (VC-NV) or two $\mathtt{TX}_\mathtt{f}$ of 309 and four $\mathtt{TX}_\mathtt{refund}$ of 377 bytes (VC-V). For every $\mathtt{TX}_\mathtt{f}$, we need two commitment transactions of 353 bytes (in total, $8 \cdot 353$ in VC-NV or $4 \cdot 353$ in VC-V). For OFFLOAD, only one commitment transaction per ledger channel needs to be published, along

Operations	Generalized Channels										Lightning Channels									
	VC-NV					VC-V					VC-NV					VC-V				
	on-chain			off-chain		on-chain			off-chain		on-chain			off-chain		on-chain			off-chain	
	# txs	size	cost	# txs	size	# txs	size	cost	# txs	size	# txs	size	cost	# txs	size	# txs	size	cost	# txs	size
`CREATE`	0	0	0	7	2829	0	0	0	8	2803	0	0	0	16	7704	0	0	0	14	5722
`UPDATE`	0	0	0	2	695	0	0	0	2	695	0	0	0	8	2824	0	0	0	4	1412
`OFFLOAD`	5	2134	41.73	0	0	6	2108	41.22	0	0	3	1800	35.20	0	0	4	1778	34.77	0	0
`CLOSE` (opt)	0	0	0	4	1390	0	0	0	4	1390	0	0	0	4	1412	0	0	0	4	1412
`CLOSE` (pess)	7	2829	55.32	0	0	8	2803	54.81	0	0	4	2153	42.10	0	0	5	2131	41.67	0	0

TABLE II: 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 estimates cost of publish them on the ledger.

with one $\mathtt{TX_f}$ (for VC-NV) and $\mathtt{TX_f}$ plus $\mathtt{TX_{refund}}$ (for VC-V). `CLOSE`(pess), needs to publish a commitment transaction in addition to `OFFLOAD`, resulting in 2153 (VC-NV) or 2131 (VC-V) bytes.

B. *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 [26]. We now evaluate the difference in communication overhead and fee costs compared to virtual channels, summarize them in Table III and illustrate them in Figure 10.

a) Routing communication overhead: When performing a payment between Alice and Bob via an intermediary Ingrid in a multi-hop payment over generalized channels, 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 II, 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 $9+2 \cdot n$ transactions or $3524+695 \cdot n$ bytes, which supposes a reduction of $2331 \cdot n - 3524$ bytes with respect to relying on generalized channels only. This means that a virtual channel is already cheaper if only two (or more) sequential transactions are performed. We obtain similar results if we consider virtual channels with validity instead. For Lightning channels, the overhead is larger for both the multi-hop payment and the VC setting (Table III).

b) Fee costs: In a multi-hop payment tx in a PCN, the intermediary user Ingrid charges a base fee (BF) for being

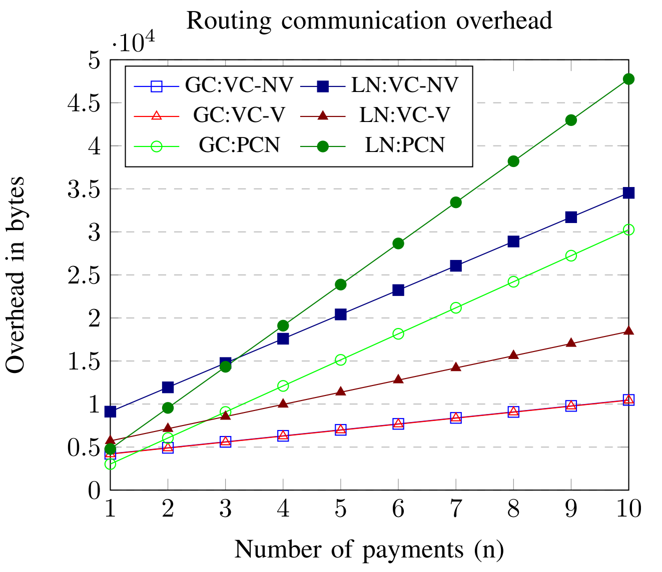

Fig. 10: Pictorial illustration of Table III.

	Overhead in bytes			fees
	1 paym.	2 paym.	n payments	tx.cash in n payments
GC: PCN	3026	6052	$3026 \cdot n$	$\mathsf{BF} \cdot n + \mathsf{FR} \cdot \mathsf{tx.cash}$
GC: VC-NV	4219	4914	$3524 + 695 \cdot n$	$\mathsf{BF} + \mathsf{FR} \cdot \mathsf{tx.cash}$
GC: VC-V	4193	4888	$3498 + 695 \cdot n$	
LN: PCN	4776	9552	$4776 \cdot n$	$\mathsf{BF} \cdot n + \mathsf{FR} \cdot \mathsf{tx.cash}$
LN: VC-NV	9116	11940	$6292 + 2824 \cdot n$	$\mathsf{BF} + \mathsf{FR} \cdot \mathsf{tx.cash}$
LN: VC-V	5722	7134	$4310 + 1412 \cdot n$	

TABLE III: Comparison of virtual channels (VC) to multi-hop payments (PCN) showing the overhead in bytes for a different number of payments and the difference in fees.

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 (and even does not know how many end-users performed)1. Let us now investigate the case of paying tx.cash in n micropayments of equal value. In PCN case, the total cost would be $\sum_{i=1}^{n} \mathsf{BF} + \mathsf{FR} \cdot \frac{\mathsf{tx.cash}}{n} = \mathsf{BF} \cdot n + \mathsf{FR} \cdot \mathsf{tx.cash}$. 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., $n > 1$, it is beneficial to use virtual channels for reducing the fee costs by $\mathsf{BF} \cdot (n-1)$.

c) Summary: We find that the best construction in practice is the combination of virtual channels on top of generalized channels, as this yields the least overhead after only two or more sequential payments. However, building virtual channels over LN channels also yields less overhead than multi-hop PCN payments over LN.

VI. Related Work

In this section, we position this work in the landscape of the literature for off-chain payments protocols.

a) Payment Channels: Started from the Lightning Channels construction [26], the idea of 2-party payment channels has been largely used in academia and industry as a building block for more complex off-chain payment protocols. More recently, Aumayr et al. [2] have proposed a novel construction for 2-party payment channels that overcome some of the drawbacks of the original Lightning channels. While their benefit in terms of scalability is out of any doubt by now, payment channels are limited to payments between two users and consequently its overall utility.

A concurrent work [18] has also proposed a virtual channel construction over Bitcoin. However, their construction uses decreasing time-locks instead of a punishment mechanism in order to guarantee that only the latest state can be posted on the blockchain. As a consequence, their construction only allows a fixed number of transactions to be made during the lifetime of the virtual channel. This is quite restrictive as it requires users to close and open new virtual channels more frequently which goes against the purpose of virtual channels. Note that one cannot simply increase the time-lock as this would essentially lock the coins of the users for a longer period of time. Furthermore, our constructions are *generalized* virtual channels, i.e., they are not limited to just payments, but rather allow to run any Bitcoin script off-chain. In addition, we propose a modular approach compared to the monolithic construction in [18]. Finally, our work proposes two protocols, which each have their advantages in different use cases.

b) Payment Channel Networks (PCN) and Payment Channel Hub (PCH): A PCN allows a payment between two users that do not share a payment channel but are however connected through a path of payment channels. The notion of PCN started with the deployment of Lightning Network [26] for Bitcoin and Raiden Network [30] for Ethereum and has been widely studied in academia to research into different aspects such as privacy [22, 21], routing of payments [27], collateral management [14] and others. Similar to PCN, different constructions for PCH exist [28, 17, 5] that allow a payment between two users through a single intermediary, the payment hub. PCNs and PCHs, however, share the drawback that each payment between two users require the active involvement of the intermediary (or several intermediaries in the case of PCH), which reduces the reliability (e.g., the intermediary can go offline) and increases the cost of the payment (e.g., each intermediary charges a fee for the payment).

c) State Channels: Several works [12, 11, 23, 10] have shown how to leverage the highly expressive scripting language available at Ethereum to construct (multi-party) state channels. A state channel allows the involved parties to carry out off-chain computations, possibly other than payments. Closer to our work, Dziembowski et al. [13] showed how to construct a virtual channel leveraging two payment channels defined in Ethereum. These approaches are, however, highly tight to the functionality provided by the Ethereum scripting language and their constructions cannot be reused in other cryptocurrencies. In this work, we instead show that virtual channels can be constructed from digital signatures and time-lock mechanism only, which makes virtual channels accessible for virtually any cryptocurrency system available today.

VII. Conclusion

Current PCNs route payments between two users through intermediate nodes, making the system less reliable (intermediaries might be offline), expensive (intermediaries 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 scripting language supported by virtually every cryptocurrency, including Bitcoin. Our two protocols provide a tradeoff on who can offload the virtual channel, similar to the preemptible vs. non-preemptible virtual machines in the cloud setting. In other words, our virtual channel construction without validity is more suitable for intermediaries who can monitor the blockchain regularly, such as payment channel hubs, but can also close the virtual channel at anytime if desired. Our virtual channel protocol with validity however, is more suitable for light intermediaries who do not wish to be active during the lifetime of the virtual channel but cannot close the virtual channel before its validity has expired. 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 $9 + 2 \cdot n$ off-chain transactions for a total of $3524 + 695 \cdot n$ bytes, with no on-chain footprint.

As mentioned in the introduction of this work, the task of designing secure virtual channels has been proven to be challenging even on a cryptocurrency like Ethereum [13] which supports smart contract execution. Unsurprisingly, this task becomes even more complex when building virtual channels for blockchains that support only a limited scripting language as it is not possible to take advantage of the full computation power of Turing complete smart contracts. Due to these significantly differing underlying assumptions (smart contracts vs. limited scripting languages), the virtual channel protocols based on Ethereum [13] and the protocols presented in this work are incomparable. We emphasize that we view our virtual channel constructions as complementary to the one presented in [13], as we do not aim to improve the construction of [13] but rather extend the concept of virtual channels to a broader class of blockchains.

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 a 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 interesting future work, as it involves a quantitative analysis that falls off 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, by the Vienna Business Agency through the project Vienna Cybersecurity and Privacy Research Center (VISP), by CoBloX Labs and by the ERC Project PREP-CRYPTO 724307.

References

[1] A. M. Antonopoulos. *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. 1st. OReilly Media, Inc., 2014. ISBN: 1449374042.

[2] L. Aumayr et al. *Generalized Bitcoin-Compatible Channels*. Cryptology ePrint Archive, Report 2020/476. https://eprint.iacr.org/2020/476. 2020.

[3] C. Badertscher et al. "Bitcoin as a Transaction Ledger: A Composable Treatment". In: *CRYPTO 2017, Part I*. Ed. by J. Katz and H. Shacham. Vol. 10401. LNCS. Springer, Heidelberg, Aug. 2017, pp. 324–356.

[4] S. Bano et al. "SoK: Consensus in the Age of Blockchains". In: *AFT 2019*, pp. 183–198.

[5] E. Ben-Sasson et al. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *IEEE SP*. 2014, pp. 459–474.

[6] *Bitcoin-Compatible Virtual Channels: Github repository*. https://github.com/utxo-virtual-channels/vc. 2020.

[7] *Bitcoin Wiki: Payment Channels*. https://en.bitcoin.it/wiki/Payment_channels. 2018.

[8] R. Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *42nd FOCS*. IEEE Computer Society Press, Oct. 2001, pp. 136–145.

[9] R. Canetti et al. "Universally Composable Security with Global Setup". In: *TCC 2007*. Ed. by S. P. Vadhan. Vol. 4392. LNCS. Springer, Heidelberg, Feb. 2007, pp. 61–85.

[10] M. M. T. Chakravarty et al. "Hydra: Fast Isomorphic State Channels". In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 299. URL: https://eprint.iacr.org/2020/299.

[11] S. Dziembowski et al. "General State Channel Networks". In: *ACM CCS*. 2018, pp. 949–966.

[12] S. Dziembowski et al. "Multi-party Virtual State Channels". In: *EUROCRYPT 2019, Part I*. 2019, pp. 625–656.

[13] S. Dziembowski et al. "Perun: Virtual Payment Hubs over Cryptocurrencies". In: *IEEE SP*. 2019, pp. 106–123.

[14] C. Egger et al. "Atomic Multi-Channel Updates with Constant Collateral in Bitcoin-Compatible Payment-Channel Networks". In: *ACM CCS*. 2019, pp. 801–815.

[15] O. Goldreich. *Foundations of Cryptography: Volume 1*. New York, NY, USA: Cambridge University Press, 2006. ISBN: 0521035368.

[16] L. Gudgeon et al. "SoK: Layer-Two Blockchain Protocols". In: *FC 2020*. 2020, pp. 201–226.

[17] E. Heilman et al. "TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub". In: *NDSS 2017*. The Internet Society, 2017.

[18] M. Jourenko et al. "Lightweight Virtual Payment Channels". In: *CANS 2020*. 2020, pp. 365–384.

[19] G. Kappos et al. "An Empirical Analysis of Privacy in the Lightning Network". In: *CoRR* abs/2003.12470 (2020). arXiv: 2003.12470. URL: https://arxiv.org/abs/2003.12470.

[20] J. Katz et al. "Universally Composable Synchronous Computation". In: *TCC 2013*. Ed. by A. Sahai. Vol. 7785. LNCS. Springer, Heidelberg, Mar. 2013, pp. 477–498. DOI: 10.1007/978-3-642-36594-2_27.

[21] G. Malavolta et al. "Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability". In: *NDSS*. 2019.

[22] G. Malavolta et al. “Concurrency and Privacy with Payment-Channel Networks”. In: *ACM CCS 17*. Ed. by B. M. Thuraisingham et al. ACM Press, 2017, pp. 455–471.

[23] A. Miller et al. “Sprites and State Channels: Payment Networks that Go Faster Than Lightning”. In: *FC 2019*. 2019, pp. 508–526.

[24] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. http://bitcoin.org/bitcoin.pdf. 2009.

[25] U. Nisslmueller et al. “Toward Active and Passive Confidentiality Attacks on Cryptocurrency Off-chain Networks”. In: *ICISSP*. Ed. by S. Furnell et al. 2020, pp. 7–14.

[26] J. Poon and T. Dryja. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. Draft version 0.5.9.2, available at https://lightning.network/lightning-network-paper.pdf. Jan. 2016.

[27] S. Roos et al. “Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions”. In: *NDSS*. 2018.

[28] E. Tairi et al. *A^2L: Anonymous Atomic Locks for Scalability and Interoperability in Payment Channel Hubs*. In press.

[29] S. Tikhomirov et al. “A Quantitative Analysis of Security, Anonymity and Scalability for the Lightning Network”. In: *IEEE EuroS&P*. 2020, pp. 387–396.

[30] *Update from the Raiden team on development progress, announcement of raidEX*. https://tinyurl.com/z2snp9e. Feb. 2017.

[31] A. Zamyatin et al. *SoK: Communication Across Distributed Ledgers*. In press.

Appendix

A. 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) [9] which extends the standard UC framework [8] by allowing for a global setup. Since our model is essentially the same as in [2], which in turn follows [11, 12], parts of this section are taken verbatim from there.

a) Protocols and adversarial model: We consider a protocol π that runs between parties from the set $\mathcal{P} = \{P_1, \dots, P_n\}$. 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 $\mathcal{H}_1, \dots, \mathcal{H}_m$. In this case we say that the protocol π *works in the* $(\mathcal{H}_1, \dots, \mathcal{H}_m)$*-hybrid model* and write $\pi^{\mathcal{H}_1,\dots,\mathcal{H}_m}$.

b) 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 [12], which in turn follows [20], 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 [12] 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.

c) 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 $(\text{register}, pk_P)$; (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 $\texttt{PKI}$ for those parties. The functionality maintains the set of all accepted transactions $\texttt{TX}$ and all unspent transaction outputs $\texttt{UTXO}$. The set $\mathcal{V}$ defines valid output conditions.

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

Post transaction: Upon $(\text{post}, \mathsf{tx}) \overset{\tau_0}{\hookleftarrow} P$, check that $|\texttt{PKI}| = |\mathcal{P}|$. 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 $(t, \mathsf{tx}') \in \texttt{TX}$, $\mathsf{tx}'.\mathsf{txid} \neq \mathsf{tx}.\mathsf{txid}$.
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 $(t, tid, i, \theta) \in \texttt{UTXO}$ 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 $I := \{\texttt{utxo} := (t, tid, i, \theta) \mid \texttt{utxo} \in \texttt{UTXO} \wedge (tid, i) \in \mathsf{tx}.\mathsf{Input}\}$, then it must hold that $\sum_{\theta' \in \mathsf{tx}.\mathsf{Output}} \theta'.\mathsf{cash} \leq \sum_{\texttt{utxo} \in I} \texttt{utxo}.\theta.\mathsf{cash}$
5) The absolute time-lock of the transaction has expired, i.e. it must hold that $\mathsf{tx}.\mathsf{TimeLock} \leq \mathsf{now}$.

If all the above checks return true, add (τ_1, tx) to $\texttt{TX}$, remove the spent outputs from $\texttt{UTXO}$, i.e., $\texttt{UTXO} := \texttt{UTXO} \setminus I$ and add the outputs of tx to $\texttt{UTXO}$, i.e., $\texttt{UTXO} := \texttt{UTXO} \cup \{(\tau_1, \mathsf{tx}.\mathsf{txid}, i, \theta_i)\}_{i \in [n]}$ for $(\theta_1, \ldots, \theta_n) := \mathsf{tx}.\mathsf{Output}$. 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 [3], that provide a more accurate formalization of the Bitcoin ledger in the UC framework [8]. 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.

d) 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

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

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

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

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

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

(where "$\overset{c}{\approx}$" denotes computational indistinguishability of distribution ensembles, see, e.g., [15]).

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.

B. Adaptor Signatures

Adaptor signatures have been introduced and used in the cryptocurrencies community for some time, but have been formalized for the first time in [2]. 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 [2]. 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 [2].

Definition 2 (Adaptor Signature Scheme): An adaptor signature scheme wrt. a hard relation R and a signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ consists of four algorithms $\Xi_{R.\Sigma} = (\mathsf{pSign}, \mathsf{Adapt}, \mathsf{pVrfy}, \mathsf{Ext})$ 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 [2] for the formal definitions.

a) 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.

b) 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 $Y \in L_R$.

c) 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 $(Y, y) \in R$.

d) 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. Ledger channels

For completeness, we recall the ledger channel ideal functionality $\mathcal{F}_L$ from [2]. We then show that we cannot use this ideal functionality in a black-box way and instead we introduce a *wrapped ledger channel functionality* $\mathcal{F}_{preL}$. Finally, we present a protocol Π_{preL} that realizes $\mathcal{F}_{preL}$.

1) Ledger Channel Functionality: We now recall the ideal functionality for ledger channels $\mathcal{F}_L(T, k)$ from [2]. This functionality is parameterized by $T \in \mathbb{N}$ that upper bounds the number of consecutive off-chain communication rounds between parties and a parameter $k \in \mathbb{N}$ that defined the number of ways a channel can be closed (i.e., number of commit transactions per update).

Following [2], the pseudocode presented below excludes several checks that one would expect the functionality to make. We formalize all the missing checks in form of a functionality wrapper in Appendix E. Moreover, in order to simplify the notation in the functionality description, 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 $\mathcal{F}_L(T, k)$

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

$\boxed{\text{Create}}$

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

Both agreed: If already received $(\texttt{CREATE}, \gamma, tid_Q) \overset{\tau}{\hookleftarrow} Q$, where $\tau_0 - \tau \le T_1$, wait if in round $\tau_1 \le \tau + \Delta + T_1$ 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 $(\texttt{CREATED}, \gamma.\mathsf{id}) \overset{\tau_1}{\hookrightarrow} \gamma.\mathsf{endUsers}$. Else stop.

Wait for Q: Else store the message and stop.

$\boxed{\text{Update}}$

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

1) In round $\tau_1 \le \tau_0 + T$, let $\mathcal{S}$ set $|\vec{tid}| = k$. Then $(\texttt{UPDATE–REQ}, id, \vec{\theta}, t_{\mathsf{stp}}, \vec{tid}) \overset{\tau_1}{\hookrightarrow} Q$ and $(\texttt{SETUP}, id, \vec{tid}) \overset{\tau_1}{\hookrightarrow} P$.
2) If $(\texttt{SETUP–OK}, id) \xhookleftarrow{\tau_2 \le \tau_1 + t_{\mathsf{stp}}} P$, then $(\texttt{SETUP–OK}, id) \xhookrightarrow{\tau_2 + T_1} Q$. Else stop.
3) If $(\texttt{UPDATE–OK}, id) \xhookleftarrow{\tau_2 + T_1} Q$, then $(\texttt{UPDATE–OK}, id) \xhookrightarrow{\tau_2 + 2T_1} P$. Else distinguish:
   - If Q honest or if instructed by $\mathcal{S}$, stop (update rejected).
   - Else execute $\texttt{L–ForceClose}(id)$ and stop.
4) If $(\texttt{REVOKE}, id) \xhookleftarrow{\tau_2 + 2T_1} P$, $(\texttt{REVOKE–REQ}, id) \xhookrightarrow{\tau_2 + 2T_1 + T_2} Q$. Else execute $\texttt{L–ForceClose}(id)$ and stop.
5) If $(\texttt{REVOKE}, id) \xhookleftarrow{\tau_2 + 2T_1 + T_2} Q$, set $\gamma.\mathsf{st} = \vec{\theta}$ and $\Gamma(id) := (\gamma, \mathsf{tx})$. Then $(\texttt{UPDATED}, id, \vec{\theta}) \xhookrightarrow{\tau_2 + 2T_1 + 2T_2} \gamma.\mathsf{endUsers}$ and stop. Else distinguish:
   - If Q honest, execute $\texttt{L–ForceClose}(id)$ 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 $\texttt{L–ForceClose}(id)$ and stop.

$\boxed{\text{Close}}$

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

Both agreed: If you received $(\texttt{CLOSE}, id) \overset{\tau}{\hookleftarrow} Q$, where $\tau_0 - \tau \le T_1$, let $(\gamma, \mathsf{tx}) := \Gamma(id)$ and distinguish:
- If in round $\tau_1 \le \tau + T_1 + \Delta$ 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 $\Gamma(id) := (\perp, \mathsf{tx})$, $(\texttt{CLOSED}, id) \overset{\tau_1}{\hookrightarrow} \gamma.\mathsf{endUsers}$ and stop.
- If tx is still unspent in round $\tau + T_1 + \Delta$, output $(\texttt{ERROR}) \xhookrightarrow{\tau + T_1 + \Delta} \gamma.\mathsf{endUsers}$ and stop.

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

$\boxed{\text{Punish (executed at the end of every round } \tau_0)}$

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

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

Close: Either $\Gamma(id) = (\perp, \mathsf{tx})$ before round $\tau_0 + \Delta$ (channels was peacefully closed) or in round $\tau_1 \le \tau_0 + 2\Delta$ 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 $\Gamma(id) := (\perp, \mathsf{tx})$ and $(\texttt{CLOSED}, id) \overset{\tau_1}{\hookrightarrow} \gamma.\mathsf{endUsers}$.

Error: Otherwise $(\texttt{ERROR}) \xhookrightarrow{\tau_0 + 2\Delta} \gamma.\mathsf{endUsers}$.

Subprocedure $\texttt{L–ForceClose}(id)$

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

2) Wrapped ledger channel functionality: 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 channels. In order to overcome these issues, we present $\mathcal{F}_{preL}(T, k)$, an ideal functionality that extends $\mathcal{F}_L(T, k)$ to support the preparation of 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.

a) Generalized channels: The functionality treats messages about standard generalized channels exactly as the functionality $\mathcal{F}_L(T, k)$ presented earlier in this section.

b) Creation: In order to pre-create a generalized channel γ, both end-users of the channel must send the message $(\texttt{PRE–CREATE}, \gamma, \texttt{TX}_\texttt{f}, i, t_{ofl})$ to the ideal functionality. Here $\texttt{TX}_\texttt{f}||i$ 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.

c) 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 $\texttt{L–ForceClose}$ since there is no ledger channel to be forcefully closed. Instead, it calls a subprocedure called $\texttt{Wait–if–Register}$ which add a flag "$\texttt{in–dispute}$" 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.

d) 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 "$\texttt{in–dispute}$", then it immediately calls $\texttt{L–ForceClose}$.

The formal functionality description on the functionality wrapper $\mathcal{F}_{preL}(T, k)$ follows. Again, for the sake of readability, we exclude several natural checks from the functionality description. These checks are formalized in Appendix E.

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 $\texttt{CREATE}$, $\texttt{UPDATE}$, $\texttt{SETUP–OK}$, $\texttt{UPDATE–OK}$, $\texttt{REVOKE}$ or $\texttt{CLOSE}$ message, then behave exactly as the functionality $\mathcal{F}_L(T, k)$.

Pre-Create

Upon $(\texttt{PRE–CREATE}, \gamma, \texttt{TX}_\texttt{f}, i, t_{ofl}) \xhookleftarrow{\tau_0} P$, let $\mathcal{S}$ define $T_1 \leq T$ and:

Both agreed: If already received $(\texttt{PRE–CREATE}, \gamma, \texttt{TX}_\texttt{f}, i, t_{ofl}) \xhookleftarrow{\tau} Q$, where $\tau_0 - \tau \leq T_1$, check that $\texttt{TX}_\texttt{f}.\mathsf{Output}[i].\mathsf{cash} = \gamma.\mathsf{cash}$. If yes, set $\Gamma_{pre}(\gamma.\mathsf{id}) := (\gamma, \texttt{TX}_\texttt{f}, t_{ofl})$ and $(\texttt{PRE–CREATED}, \gamma.\mathsf{id}) \xhookrightarrow{\tau_0} \gamma.\mathsf{endUsers}$. Else stop.
Wait for Q: Else store the message and stop.

Pre-Update

Upon $(\texttt{PRE–UPDATE}, id, \vec{\theta}, t_{\mathsf{stp}}) \xhookleftarrow{\tau_0} P$, let $\mathcal{S}$ define $T_1, T_2 \leq T$, parse $(\gamma, \texttt{TX}_\texttt{f}, t) := \Gamma_{pre}(id)$ and proceed as follows:

1) In round $\tau_1 \leq \tau_0 + T$, let $\mathcal{S}$ set $|\vec{tid}| = k$. Then $(\texttt{PRE–UPDATE–REQ}, id, \vec{\theta}, t_{\mathsf{stp}}, \vec{tid}) \xhookrightarrow{\tau_1} Q$ and $(\texttt{PRE–SETUP}, id, \vec{tid}) \xhookrightarrow{\tau_1} P$.
2) If $(\texttt{PRE–SETUP–OK}, id) \xhookleftarrow{\tau_2 \leq \tau_1 + t_{\mathsf{stp}}} P$, then $(\texttt{PRE–SETUP–OK}, id) \xhookrightarrow{\tau_2 + T_1} Q$. Else stop.
3) In round $\tau_2 + T_1$ distinguish:
   - If $(\texttt{PRE–UPDATE–OK}, id) \xhookleftarrow{\tau_2 + T_1} Q$, then $(\texttt{PRE–UPDATE–OK}, id) \xhookrightarrow{\tau_2 + 2T_1} P$.
   - If not and Q honest or if instructed by $\mathcal{S}$, $(\texttt{PRE–UPDATE–REJECT}, id) \xhookrightarrow{\tau_2 + 2T_1} P$.
   - Else execute $\texttt{Wait–if–Register}(id)$ and stop.
4) If $(\texttt{PRE–REVOKE}, id) \xhookleftarrow{\tau_2 + 2T_1} P$, $(\texttt{PRE–REVOKE–REQ}, id) \xhookrightarrow{\tau_2 + 2T_1 + T_2} Q$. Else execute $\texttt{Wait–if–Register}(id)$ and stop.
5) If $(\texttt{PRE–REVOKE}, id) \xhookleftarrow{\tau_2 + 2T_1 + T_2} Q$, set $\gamma.\mathsf{st} = \vec{\theta}$ and $\Gamma_V(id) := (\gamma, \texttt{TX}_\texttt{f})$. Then $(\texttt{PRE–UPDATED}, id, \vec{\theta}) \xhookrightarrow{\tau_2 + 2T_1 + 2T_2} \gamma.\mathsf{endUsers}$ and stop. Else $\texttt{Wait–if–Register}(id)$ and stop.

Register – executed in every round

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

Subprocedure $\texttt{Wait–if–Register}(id)$

Let τ_0 be the current round and $(\gamma, \texttt{TX}_\texttt{f}, t_{ofl}) := \Gamma_{pre}(id)$.
1) Set $\Gamma_{pre}(id) := (\gamma, \texttt{TX}_\texttt{f}, t_{ofl}, \texttt{in–dispute})$.
2) Wait for t_{ofl} rounds. If after this time, $\Gamma_{pre}(id) \neq \perp$, then set $\vec{\theta}_{old} := \gamma.\mathsf{st}$, $\gamma.\mathsf{st} := \{\vec{\theta}_{old}, \vec{\theta}\}$ and $\Gamma_{pre}(id) := (\gamma, \texttt{TX}_\texttt{f}, t_{ofl}, \texttt{in–dispute})$.

3) Realizing the wrapped functionality: While [2] 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 $\mathcal{F}_{preL}$. Note that, just like Π_L, Π_{preL} uses the adaptor signature primitive that we recalled in Appendix B.

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

a) Pre-Create: The only difference 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.

b) 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.

c) 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 $\mathtt{L–ForceClose}^P$ by calls to $\mathtt{Wait–if–Register}^P$ 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.

We are now prepared to present the formal description of the protocol. As for the ideal functionalities, we exclude several natural checks that parties have to make. We present all these checks in form of a protocol wrapper in Appendix E.

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 $(\mathtt{PRE–CREATE}, \gamma, \mathtt{TX_f}, i, t_{ofl}) \overset{t_0}{\hookleftarrow} \mathcal{E}$:

1) If $\mathtt{TX_f}.\mathsf{Output}[i].\mathsf{cash} \neq \gamma.\mathsf{cash}$, then ignore the message.
2) Set $id := \gamma.\mathsf{id}$, generate $(R_P, r_P) \leftarrow \mathsf{GenR}$, $(Y_P, y_P) \leftarrow \mathsf{GenR}$ and send $(\text{createInfo}, id, \mathtt{TX_f}, i, t_{ofl}, R_P, Y_P) \overset{t_0}{\hookrightarrow} Q$.
3) If $(\text{createInfo}, id, \mathtt{TX_f}, i, t_{ofl}, R_Q, Y_Q) \overset{t_0+1}{\hookleftarrow} Q$, create:
$$[\mathtt{TX_c}] := \mathtt{GenCommit}([\mathtt{TX_f}], I_P, I_Q, 0)$$
$$[\mathtt{TX_s}] := \mathtt{GenSplit}([\mathtt{TX_c}].\mathsf{txid}\|1, \gamma.\mathsf{st})$$
for $I_P := (pk_P, R_P, Y_P)$, $I_Q := (pk_Q, R_Q, Y_Q)$. Else stop.
4) Compute $s^P_\mathsf{c} \leftarrow \mathsf{pSign}_{sk_P}([\mathtt{TX_c}], Y_Q)$, $s^P_\mathsf{s} \leftarrow \mathsf{Sign}_{sk_P}([\mathtt{TX_s}])$ and send $(\text{createCom}, id, s^P_\mathsf{c}, s^P_\mathsf{s}) \overset{t_0+1}{\hookrightarrow} Q$.
5) If $(\text{createCom}, id, s^Q_\mathsf{c}, s^Q_\mathsf{s}) \overset{t_0+2}{\hookleftarrow} Q$, s.t. $\mathsf{pVrfy}_{pk_Q}([\mathtt{TX_c}], Y_P; s^Q_\mathsf{c}) = 1$ and $\mathsf{Vrfy}_{pk_Q}([\mathtt{TX_s}]; s^Q_\mathsf{s}) = 1$, set
$$\mathtt{TX_c} := ([\mathtt{TX_c}], \{\mathsf{Sign}_{sk_P}([\mathtt{TX_c}]), \mathsf{Adapt}(s^Q_\mathsf{c}, y_P)\})$$
$$\mathtt{TX_s} := ([\mathtt{TX_s}], \{s^P_\mathsf{s}, s^Q_\mathsf{s}\})$$
$$\Gamma^P_{pre}(\gamma.\mathsf{id}) := (\gamma, \mathtt{TX_f}, (\mathtt{TX_c}, r_P, R_Q, Y_Q, s^P_\mathsf{c}), \mathtt{TX_s}, t_{ofl}).$$
and send $(\mathtt{PRE–CREATED}, id) \overset{t_0+2}{\hookrightarrow} \mathcal{E}$.

Pre-Update

Party P upon $(\mathtt{PRE–UPDATE}, id, \vec{\theta}, t_\mathsf{stp}) \overset{t^P_0}{\hookleftarrow} \mathcal{E}$

1) Generate $(R_P, r_P) \leftarrow \mathsf{GenR}$, $(Y_P, y_P) \leftarrow \mathsf{GenR}$ and send the message $(\text{updateReq}, id, \vec{\theta}, t_\mathsf{stp}, R_P, Y_P) \overset{t^P_0}{\hookrightarrow} Q$.

$$\boxed{\text{Party } Q \text{ upon } (\text{updateReq}, id, \vec{\theta}, t_\mathsf{stp}, R_P, Y_P) \overset{t^Q_0}{\hookleftarrow} P}$$

2) Generate $(R_Q, r_Q) \leftarrow \mathsf{GenR}$ and $(Y_Q, y_Q) \leftarrow \mathsf{GenR}$.
3) Extract $\mathtt{TX_f}$ and t_{ofl} from $\Gamma^P_{pre}(id)$.
4) Set $t_\mathsf{lock} := t^Q_0 + t_\mathsf{stp} + 4 + \Delta + t_{ofl}$ and
$$[\mathtt{TX_c}] := \mathtt{GenCommit}([\mathtt{TX_f}], I_P, I_Q, t_\mathsf{lock})$$
$$[\mathtt{TX_s}] := \mathtt{GenSplit}([\mathtt{TX_c}].\mathsf{txid}\|1, \vec{\theta})$$
where $I_P := (pk_P, R_P, Y_P)$, $I_Q := (pk_Q, R_Q, Y_Q)$.
5) Sign $s^Q_\mathsf{s} \leftarrow \mathsf{Sign}_{sk_Q}([\mathtt{TX_s}])$, send $(\text{updateInfo}, id, R_Q, Y_Q, s^Q_\mathsf{s}) \overset{t^Q_0}{\hookrightarrow} P$, $(\mathtt{PRE–UPDATE–REQ}, id, \vec{\theta}, t_\mathsf{stp}, \mathtt{TX_s}.\mathsf{txid}) \overset{t^Q_0+1}{\hookrightarrow} \mathcal{E}$.

$$\boxed{\text{Party } P \text{ upon } (\text{updateInfo}, id, h_Q, Y_Q, s^Q_\mathsf{s}) \overset{t^P_0+2}{\hookleftarrow} Q}$$

6) Extract $\mathtt{TX_f}$ and t_{ofl} from $\Gamma^Q_{pre}(id)$.
7) Set $t_\mathsf{lock} := t^P_0 + t_\mathsf{stp} + 5 + \Delta + t_{ofl}$, and
$$[\mathtt{TX_c}] := \mathtt{GenCommit}([\mathtt{TX_f}], I_P, I_Q, t_\mathsf{lock})$$
$$[\mathtt{TX_s}] := \mathtt{GenSplit}([\mathtt{TX_c}].\mathsf{txid}\|1, \vec{\theta}),$$
for $I_P := (pk_P, R_P, Y_P)$ and $I_Q := (pk_Q, R_Q, Y_Q)$. If it holds that $\mathsf{Vrfy}_{pk_Q}([\mathtt{TX_s}]; s^Q_\mathsf{s}) = 1$, $(\mathtt{PRE–SETUP}, id, \mathtt{TX_s}.\mathsf{txid}) \overset{t^P_0+2}{\hookrightarrow} \mathcal{E}$. Else stop.

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

$\boxed{\text{Party } Q}$

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

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

$\boxed{\text{Party } P}$

11) In round $t_1^P + 2$ distinguish the following cases:
   - If $(\text{updateComQ}, id, s_{\mathsf{c}}^Q) \xleftarrow{t_1^P+2} Q$, s.t. $\mathsf{pVrfy}_{pk_Q}([\mathtt{TX_c}], Y_P; s_{\mathsf{c}}^Q) = 1$, output $(\mathtt{PRE\text{–}UPDATE\text{–}OK}, id) \xhookrightarrow{t_1^P+2} \mathcal{E}$.
   - If $(\text{updateNotOk}, id, r_Q) \xleftarrow{t_1^P+2} Q$, 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 $(\mathtt{PRE\text{–}UPDATE\text{–}REJECT}) \xhookrightarrow{t_1^P+2} \mathcal{E}$ and stop.
   - Else, execute the procedure $\mathtt{Wait\text{–}if\text{–}Register}^P(id)$ and stop.

12) If $(\mathtt{PRE\text{–}REVOKE}, id) \xleftarrow{t_1^P+2} \mathcal{E}$, parse $\Gamma_{pre}^P(id)$ as $(\gamma, \mathtt{TX_f}, (\overline{\mathtt{TX}}_{\mathsf{c}}, \bar{r}_P, \bar{R}_Q, \bar{Y}_Q, \bar{s}_{\mathtt{Com}}^P), \overline{\mathtt{TX}}_{\mathsf{s}})$ 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 $\mathtt{TX_c} := ([\mathtt{TX_c}], \{\mathsf{Sign}_{sk_P}([\mathtt{TX_c}]), \mathsf{Adapt}(s_{\mathsf{c}}^Q, y_P)\})$., and send $(\text{revokeP}, id, \bar{r}_P) \xhookrightarrow{t_1^P+2} Q$. Else, execute $\mathtt{Wait\text{–}if\text{–}Register}^P(id)$ and stop.

$\boxed{\text{Party } Q}$

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

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

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

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

$\boxed{\text{Party } P}$

15) If $(\text{revokeQ}, id, \bar{r}_Q) \xleftarrow{t_1^P+4} Q$ s.t. $(\bar{R}_Q, \bar{r}_Q) \in R$, then set $\Theta^P(id) := \Theta^P(id) \cup ([\overline{\mathtt{TX}}_{\mathsf{c}}], \bar{r}_Q, \bar{Y}_Q, \bar{s}_{\mathtt{Com}}^P)$ and $(\mathtt{PRE\text{–}UPDATED}, id) \xhookrightarrow{t_1^P+4} \mathcal{E}$. 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 $\Gamma(id) := (\gamma, \mathtt{TX_f}, (\mathtt{TX_c}, r_P, R_Q, Y_Q, s_{\mathsf{c}}^P), \mathtt{TX_s})$.
   b) Set $\Gamma_{pre}^P(id) := \perp$
   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 $\mathsf{tx.TimeLock}$ to t if $t > \mathsf{now}$ 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 $X := \Gamma_{pre}^P(id)$. Then set $\Gamma_{pre}^P(id) := (X, \mathtt{in\text{–}dispute})$.

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. Virtual Channels

In this section we first describe and present the ideal functionality $\mathcal{F}_V$ that describes the ideal behavior of virtual channels. We then give a high level description of our construction of virtual channels with validity, before giving the full formal protocol description of our virtual channel constructions.

1) Ideal functionality for virtual channels: $\mathcal{F}_V$ can be viewed as an extension of the ledger channel functionality $\mathcal{F}_L$ defined in [2] and here presented in Appendix C. 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 operations in $\mathcal{F}_L$. The ideal functionality $\mathcal{F}_V$ communicates with the parties $\mathcal{P}$, the simulator $\mathcal{S}$ and the ledger $\widehat{\mathcal{L}}$ (see Appendix A). 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 at a high level.

a) 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.

b) 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.

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: (i), the update is peacefully rejected and parties simply abort the virtual channel creation, (ii) both channels are forcefully closed, in order to prevent a situation where one of the channels is updated and the other one is not, (iii) if γ.Ingrid has not published the old state of one of her channels to the ledger after Δ rounds, it forcefully closes the ledger channels using the new state i.e., where γ.Ingrid behaves maliciously and can publish both the old and new states, while γ.Alice or γ.Bob can only publish the new state.

c) Update: The update procedure for the virtual channel works in the same way as for ledger channels except in case of any disputes during the execution, the functionality calls V–ForceClose instead of L–ForceClose.

d) Offload: 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. 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). As in to [2], the ERROR message represents an impossible situation which should not happen as long as one of the parties is honest.

e) Close - channels without validity: Upon receiving $(\texttt{CLOSE}, id)$ 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. In this case the final balance of the parties is reflected on their underlying channels. 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).

If one of the $(\texttt{CLOSE}, id)$ messages was not received within T_1 rounds ("Wait for others" case), the closing procedure fails. The following cases my happen: (i) the update procedure of an underlying ledger channel was aborted prematurely by γ.Alice or γ.Bob which would cause the virtual channel to be forcefully closed. (ii) γ.Ingrid refuses to revoke her state during the update of either one of the underlying ledger channels where the functionality waits Δ rounds and if γ.Ingrid has not published the old state to the ledger the functionality forcefully closes the ledger channels using the new state.

f) Close - channels with validity.: This procedure starts in round $\gamma.\mathsf{val} - (4\Delta + 7T)$ to have 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.

g) 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 (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 $\gamma.\mathsf{cash} + \gamma.\mathsf{fee}$ to γ.Ingrid and tx_2 funds γ on-chain, then the virtual channel has been offloaded and the message (OFFLOADED) is sent to γ.users. If after T_1 rounds, only one transaction tx is on the ledger, which assigns γ.cash coins to a single honest party P and spends the funding transaction of only one of γ.subchan, the functionality sends (PUNISHED) to P. Otherwise, the functionality outputs (ERROR) to γ.users.

h) Notation: In the functionality description, we use the notion of *rooted transactions* that we now explain (see Figure 11 for a concrete example). 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) $\forall \mathsf{tx}_i \leq \mathsf{tx}. \exists \mathsf{tx}_j \in R. \mathsf{tx}_j \leq \mathsf{tx}_i \vee \mathsf{tx}_i \leq \mathsf{tx}_j$,
2) $\forall \mathsf{tx}_i, \mathsf{tx}_j \in R. \mathsf{tx}_i \neq \mathsf{tx}_j, \mathsf{tx}_i \not\leq \mathsf{tx}_j$ and
3) $\mathsf{tx} \notin R$.

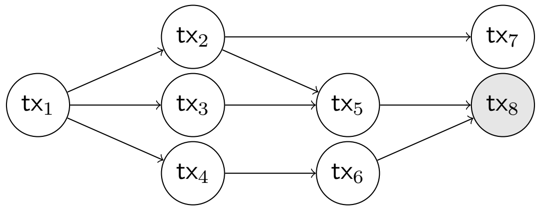

Fig. 11: The root sets of transaction tx_8 are $\{\mathsf{tx}_1\}$, $\{\mathsf{tx}_2, \mathsf{tx}_3, \mathsf{tx}_4\}$, $\{\mathsf{tx}_5, \mathsf{tx}_6\}$, $\{\mathsf{tx}_4, \mathsf{tx}_5\}$ and $\{\mathsf{tx}_2, \mathsf{tx}_3, \mathsf{tx}_6\}$.

As in the case of ledger channel functionalities, the formal description of $\mathcal{F}_V$ excludes several checks that one would expect the functionality to make. These checks are formalized in form of a functionality wrapper in Appendix E.

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

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

For messages about ledger channels, behave as $\mathcal{F}_L(T,1)$.

Create

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

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

1) Let $\mathcal{S}$ define $\vec{\theta_A}$ and $\vec{\theta_B}$ and set $(id_A, id_B) := \gamma.\mathsf{subchan}$.
2) Execute $\mathtt{UpdateState}(id_A, \vec{\theta_A})$, $\mathtt{UpdateState}(id_B, \vec{\theta_B})$, set $\Gamma(\gamma.\mathsf{id}) := \gamma$, send $(\mathtt{CREATED}, \gamma) \overset{\tau_3}{\hookrightarrow} \gamma.\mathsf{endUsers}$, stop.

Wait for others: Else wait for at most T_1 rounds to receive $(\mathtt{CREATE}, \gamma) \overset{\tau_1 \leq \tau + T_1}{\hookleftarrow} Q_1$ and $(\mathtt{CREATE}, \gamma) \overset{\tau_2 \leq \tau + T_1}{\hookleftarrow} Q_2$ where $Q_1, Q_2 \in \gamma.\mathsf{users} \setminus \{P\}$ (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 $(\text{peaceful–reject}, id_i)$, unlock id_i and stop.
- If $\gamma.\mathsf{Ingrid}$ is honest or if instructed by $\mathcal{S}$, execute $\mathtt{L\text{–}ForceClose}(id_i)$ and stop.
- Otherwise wait for Δ rounds. If tx_i still unspent, then set $\vec{\theta}_{old} := \gamma_i.\mathsf{st}$, $\gamma_i.\mathsf{st} := \{\vec{\theta}_{old}, \vec{\theta}\}$ and $\Gamma(id_i) := (\gamma_i, \mathsf{tx}_i)$. Execute $\mathtt{L\text{–}ForceClose}(id_i)$ and stop.

Update

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

Offload

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

Close

Channels without validity:

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

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

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

$$\vec{\theta_A} := ((c_A, \mathtt{One\text{–}Sig}_A), (c_B + \gamma.\mathsf{fee}/2, \mathtt{One\text{–}Sig}_I)),$$
$$\vec{\theta_B} := ((c_A + \gamma.\mathsf{fee}/2, \mathtt{One\text{–}Sig}_I), (c_B, \mathtt{One\text{–}Sig}_B)),$$

3) Unlock both subchannels and execute $\mathtt{UpdateState}(id_A, \vec{\theta_A})$ and $\mathtt{UpdateState}(id_B, \vec{\theta_B})$. Set $\Gamma(id) := \perp$ and send $(\mathtt{CLOSED}, \gamma) \overset{\tau_3}{\hookrightarrow} \gamma.\mathsf{endUsers}$.

Wait for others: Else wait for at most T_1 rounds to receive $(\mathtt{CLOSE}, \gamma) \overset{\tau_1 \leq \tau + T_1}{\hookleftarrow} Q_1$ and $(\mathtt{CLOSE}, \gamma) \overset{\tau_2 \leq \tau + T_1}{\hookleftarrow} Q_2$ where $Q_1, Q_2 \in \gamma.\mathsf{users} \setminus \{P\}$ (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 $\vec{\theta}_{old} := \gamma'.\mathsf{st}$ and distinguish:

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

Channels with validity:

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

Peaceful close: If all parties in $\gamma.\mathsf{users}$ 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 $\mathtt{V\text{–}ForceClose}(\gamma.\mathsf{id})$.

Punishment (executed at the end of every round)

For every id, where $\gamma := \Gamma(id)$ is a virtual channel, set $(id_A, id_B) := \gamma.\mathsf{subchan}$. If this is the first round when $\Gamma(id_A) = (\perp, \mathsf{tx}_A)$ or $\Gamma(id_B) = (\perp, \mathsf{tx}_B)$, 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 $\gamma.\mathsf{val} = \perp$ and $T' := \gamma.\mathsf{val} + 3\Delta$ if $\gamma.\mathsf{val} \neq \perp$, 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 $(\gamma.\mathsf{cash} + \gamma.\mathsf{fee}, \mathtt{One\text{–}Sig}_I)$. In this case $(\mathtt{OFFLOADED}, id) \overset{\tau_1}{\hookrightarrow} I$, where τ_1 is the round tx_1 appeared on $\widehat{\mathcal{L}}$.
- a transaction tx_2 with an output of value $\gamma.\mathsf{cash}$ and rooted at $\{\mathsf{tx}_A, \mathsf{tx}_B\}$, if $\gamma.\mathsf{val} = \perp$, and rooted at $\{\mathsf{tx}_A\}$, if $\gamma.\mathsf{val} \neq \perp$. Let τ_2 be the round when tx_2 appeared on $\widehat{\mathcal{L}}$. Then output $(\mathtt{OFFLOADED}, id) \overset{\tau_2}{\hookrightarrow} \gamma.\mathsf{endUsers}$, set $\gamma' = \gamma$, $\gamma'.\mathsf{Ingrid} = \perp$, $\gamma'.\mathsf{subchan} = \perp$, $\gamma.\mathsf{val} = \perp$ and define $\Gamma(id) := (\gamma', \mathsf{tx}_2)$.

Punished: Else for every honest party $P \in \gamma.\mathsf{users}$, 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 $(\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \mathtt{One\text{–}Sig}_P)$ as output. In that case, output $(\mathtt{PUNISHED}, id) \overset{\tau_1}{\hookrightarrow} P$. Set $\Gamma(id) = \perp$ in the first round when $\mathtt{PUNISHED}$ was sent to all honest parties.

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

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

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

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

2) Virtual Channels With Validity: We now briefly present our virtual channel protocol with validity. We focus mainly on the creation of the virtual channel as this illustrates the main structural differences to our construction without validity. For the full formal protocol description, we refer the reader to Appendix D3.

a) Create: Unlike the without validity case, the structure of the construction with validity is not symmetric (see Figure 12). The output of the ledger channel between A and I is used as the input for the funding transaction of the virtual channel $\texttt{TX}_\texttt{f}$, whereas the output of the channel between B and I is used for the so-called refund transaction $\texttt{TX}_\texttt{refund}$.

A can create $\texttt{TX}_\texttt{f}$ on her own from the last state of her ledger channel with I. As a second step, A and B can already create the transactions required for the virtual channel γ. Additionally, I and B create the refund transaction which returns I's collateral if the virtual channel is offloaded. Finally, the created transactions are signed in reverse order. In particular, B signs $\texttt{TX}_\texttt{refund}$ so that I is ensured that she can publish it and receive her collateral and fees. Then, I signs $\texttt{TX}_\texttt{f}$ and provides the signature to A, effectively authorizing her to publish $\texttt{TX}_\texttt{f}$, thereby allowing A to offload the virtual channel.

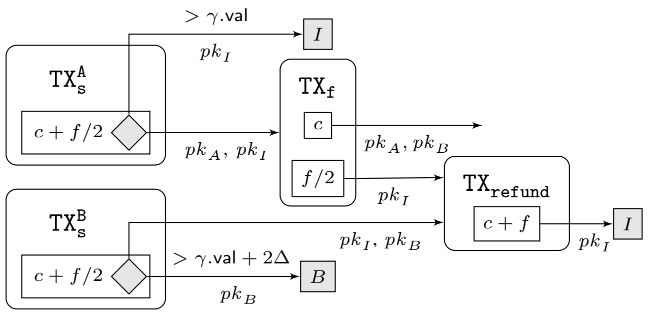

Fig. 12: Funding of a virtual channel γ with validity $\gamma.\mathsf{val}$.

b) Offload: In our virtual channel with validity, only A can offload the virtual channel γ 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 $\gamma.\mathsf{val}$ either the channel is offloaded or closed or they can punish A and get reimbursed.

c) Punish: Recall that after a successful offload, the punishment mechanisms of generalized channels apply. We now discuss other malicious behaviors specific to this construction. In this protocol, only A can post the funding transaction of the virtual channel. If the virtual channel is not closed or offloaded by $\gamma.\mathsf{val}$, A is punished. A loses her coins to I and I loses her coins to B. Therefore, though B cannot offload the channel, he will get reimbursed from his ledger channel with I and I will get reimbursed regardless of whether the virtual channel is offloaded or not. At the time val, if the virtual channel is not honestly closed or the funding is not published, I submits the punishment transaction to reimburse her collateral. Therefore, at time $\mathsf{val}+\Delta$, either the punishment or the funding transaction is posted. If the virtual channel is offloaded, I can publish the refund transaction within Δ to get her coins back.

We mention here only for our virtual channel construction with validity. We refer the reader to Appendix F2 for the full proof.

Theorem 3: Let Σ be a signature scheme that is strongly unforgeable against chosen message attacks. Then for any ledger delay $\Delta \in \mathbb{N}$, the virtual channel protocol with validity as described in Appendix D2 working in $\mathcal{F}_{preL}(3, 1)$-hybrid, UC-realizes the ideal functionality $\mathcal{F}_V(3)$.

3) Formal Virtual Channel Protocol: We now formally describe the protocol $\Pi_V(T)$ that was discussed on a high level in Section III-D and Appendix D2. Since our goal is to prove that $\Pi_V(T)$ UC-realizes $\mathcal{F}_V(T)$, we need to discuss about how parties deal with instructions about ledger channels as well as virtual channels.

a) Ledger Channels: As a first step, we discuss how parties deal with messages about ledger channels or prepared ledger channels. 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 a ledger channel is created or pre-created, party adds this channel 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 pre-cerate (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 $\text{ledger}\|id$. 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).

b) Virtual Channels: We now present the formal pseudocode for our virtual channel protocols $\Pi_V(T)$. As for ledger channels, the description excludes several checks that parties have to make. We formalize all these checks in form of a functionality wrapper in Appendix E.

Create: The creation of a virtual channel was described on a high level in Section III-D. 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 $Q := \gamma.\mathsf{otherParty}(P)$.

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

Upon receiving $(\mathtt{CREATE}, \gamma) \overset{t_0^P}{\hookleftarrow} \mathcal{E}$ 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 $(\mathtt{SETUP}, id_\alpha, tid_P) \overset{t_1^P \leq t_0^P + T}{\hookleftarrow} \mathcal{F}_{preL}$, engage in the subprotocol $\mathtt{SetupVChannel}$ with input (γ, tid_P).

$\boxed{\text{Party } I}$

Upon receiving $(\mathtt{CREATE}, \gamma) \overset{t_0^I}{\hookleftarrow} \mathcal{E}$ 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 $t_1^I \leq t_0^I + T$ you have received both $(\mathtt{UPDATE{-}REQ}, id_\alpha, \theta_A, t_{\mathsf{stp}}, tid_A) \hookleftarrow \mathcal{F}_{preL}$ and $(\mathtt{UPDATE{-}REQ}, id_\beta, \theta_B, t_{\mathsf{stp}}, tid_B) \hookleftarrow \mathcal{F}_{preL}$, then engage in the subprotocol $\mathtt{SetupVChannel}$ with inputs (γ, tid_A, tid_B). Else stop.

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

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

$\boxed{\text{Party } I}$

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

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

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

$\boxed{\text{Party } I}$

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

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

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

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

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

Subprotocol $\mathtt{SetupVChannel}$

Let t_0 be the current round.

Channels without validity

$\boxed{\text{Party } P \in \{A, B\} \text{ on input } (\gamma, tid_P)}$

1) Create the body of the funding transactions:
$$\mathtt{TX}_{\mathtt{f}}^\gamma.\mathsf{Input} := (tid_P, tid_Q)$$
$$\mathtt{TX}_{\mathtt{f}}^\gamma.\mathsf{Output} := ((\gamma.\mathsf{cash}, \mathtt{Multi{-}Sig}_{\{\gamma.\mathsf{endUsers}\}}), (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}, \mathtt{One{-}Sig}_{pk_I}))$$
2) Send $(\mathtt{PRE{-}CREATE}, \gamma, \mathtt{TX}_{\mathtt{f}}, 1, t_{ofl}) \overset{t_0}{\hookrightarrow} \mathcal{F}_{preL}$, where $t_{ofl} = 2T + 8\Delta$.
3) If $(\mathtt{PRE{-}CREATED}, \gamma.\mathsf{id}) \overset{t_1 \leq t_0 + T}{\hookleftarrow} \mathcal{F}_{preL}$, then sign the funding transaction, i.e. $s_{\mathtt{f}}^P \leftarrow \mathsf{Sign}_{sk_P}([\mathtt{TX}_{\mathtt{f}}^\gamma])$ and send $(\text{createFund}, \gamma.\mathsf{id}, s_{\mathtt{f}}^P, [\mathtt{TX}_{\mathtt{f}}^\gamma]) \overset{t_1}{\hookrightarrow} I$. Else stop.

$\boxed{\text{Party } I \text{ on input } (\gamma, tid_A, tid_B)}$

4) If you receive $(\text{createFund}, \gamma.\mathsf{id}, s_{\mathtt{f}}^A, [\mathtt{TX}_{\mathtt{f}}^\gamma]) \overset{t_2 \leq t_0 + T + 1}{\hookleftarrow} A$ and $(\text{createFund}, \gamma.\mathsf{id}, s_{\mathtt{f}}^B, [\mathtt{TX}_{\mathtt{f}}^\gamma]) \overset{t_2}{\hookleftarrow} B$, verify the funding

transaction and signatures of A and B, i.e. check:

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

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

$$s^I_\mathtt{f} := \mathsf{Sign}_{sk_I}([\mathtt{TX}^\gamma_\mathtt{f}]),$$
$$\mathtt{TX}^\gamma_\mathtt{f} := \{([\mathtt{TX}^\gamma_\mathtt{f}], s^A_\mathtt{f}, s^B_\mathtt{f}, s^I_\mathtt{f})\}.$$

Store $\Gamma^I(\gamma.\mathsf{id}) := (\bot, \mathtt{TX}^\gamma_\mathtt{f})$. Then send $(\text{createFund}, \gamma.\mathsf{id}, s^B_\mathtt{f}, s^I_\mathtt{f}) \overset{t_2}{\hookrightarrow} A$ and $(\text{createFund}, \gamma.\mathsf{id}, s^A_\mathtt{f}, s^I_\mathtt{f}) \overset{t_2}{\hookrightarrow} B$, and consider procedure successfully completed. Else stop.

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

6) Upon receiving $(\text{createFund}, \gamma.\mathsf{id}, s^Q_\mathtt{f}, s^I_\mathtt{f}) \xleftarrow{t_1+1} I$, verify all signatures, i.e. check:

$$\mathsf{Vrfy}_{pk_Q}([\mathtt{TX}^\gamma_\mathtt{f}]; s^Q_\mathtt{f}) = 1$$
$$\mathsf{Vrfy}_{pk_I}([\mathtt{TX}^\gamma_\mathtt{f}], s^I_\mathtt{f}) = 1.$$

If all checks pass define $\mathtt{TX}^\gamma_\mathtt{f} := \{([\mathtt{TX}^\gamma_\mathtt{f}], s^P_\mathtt{f}, s^Q_\mathtt{f}, s^I_\mathtt{f})\}$ and set $\Gamma^P(\gamma.\mathsf{id}) := (\bot, \mathtt{TX}^\gamma_\mathtt{f}, tid_P)$ and consider procedure successfully completed. Else stop.

Channels with validity

$\boxed{\text{Party } A \text{ on input } (\gamma, tid_A)}$

1) Send $(\text{createInfo}, \gamma.\mathsf{id}, tid_A) \overset{t_0}{\hookrightarrow} B$
2) In round $t_1 := t_0 + 1$, create the body of the funding transaction:

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

3) Send $(\mathtt{PRE\text{-}CREATE}, \gamma, \mathtt{TX}_\mathtt{f}, 1, t_{ofl}) \overset{t_1}{\hookrightarrow} \mathcal{F}_{preL}$, for $t_{ofl} = \gamma.\mathsf{val} + 3\Delta$.
4) If $(\mathtt{PRE\text{-}CREATED}, \gamma.\mathsf{id}) \xleftarrow{t_2 \leq t_1 + T} \mathcal{F}_{preL}$, then goto step (10). Else stop.

$\boxed{\text{Party } B \text{ on input } (\gamma, tid_B)}$

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

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

Else stop.

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

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

$\boxed{\text{Party } I \text{ on input } (\gamma, tid_A, tid_B)}$

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

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

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

$$s^I_\mathtt{Ref} := \mathsf{Sign}_{sk_I}([\mathtt{TX}^\gamma_\mathtt{refund}]), s^I_\mathtt{f} := \mathsf{Sign}_{sk_I}([\mathtt{TX}^\gamma_\mathtt{f}]),$$
$$\mathtt{TX}^\gamma_\mathtt{refund} := \{([\mathtt{TX}^\gamma_\mathtt{refund}], s^I_\mathtt{Ref}, s^B_\mathtt{Ref})\}.$$

Else stop.

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

$\boxed{\text{Party } A}$

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

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

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

Update: As discussed in Section III-D, 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 $\mathcal{F}_{preL} := \mathcal{F}_{preL}(T, 1)$.

$\boxed{\text{Initiating party } P\text{:}}$

1) Upon $(\mathtt{UPDATE}, id, \vec{\theta}, t_\mathsf{stp}) \overset{t_0}{\hookleftarrow} \mathcal{E}$, $(\mathtt{PRE\text{-}UPDATE}, id, \vec{\theta}, t_\mathsf{stp}) \overset{t_0}{\hookrightarrow} \mathcal{F}_{preL}$.
2) If $(\mathtt{PRE\text{-}SETUP}, id, tid_P) \xleftarrow{t_1 \leq t_0 + T} \mathcal{F}_{preL}$, $(\mathtt{SETUP}, id, tid_P) \overset{t_1}{\hookrightarrow} \mathcal{E}$. Else stop.

3) If $(\texttt{SETUP-OK}, id) \xleftarrow{t_2 \le t_1 + t_{\mathsf{stp}}} \mathcal{E}$, $(\texttt{PRE-SETUP-OK}, id) \xhookrightarrow{t_2} \mathcal{E}$. 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 $(\texttt{REVOKE}, id) \xleftrightarrow{t_3} \mathcal{E}$, $(\texttt{PRE-REVOKE}, id) \xhookrightarrow{t_3} \mathcal{F}_{preL}$. 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 $\gamma.\mathsf{st} := \vec{\theta}$ and $\Gamma(id) := \gamma$. Then $(\texttt{UPDATED}, id) \xhookrightarrow{t_4} \mathcal{F}_{preL}$. Else execute $\texttt{Offload}^P(id)$ and stop.

$\boxed{\text{Reacting party } Q}$

1) Upon $(\texttt{PRE-UPDATE-REQ}, id, \vec{\theta}, t_{\mathsf{stp}}, tid) \xleftarrow{\tau_0} \mathcal{F}_{preL}$, $(\texttt{UPDATE-REQ}, id, \vec{\theta}, t_{\mathsf{stp}}, tid) \xhookrightarrow{\tau_0} \mathcal{E}$.
2) If $(\texttt{PRE-SETUP-OK}, id) \xleftarrow{\tau_1 \le \tau_0 + t_{\mathsf{stp}} + T} \mathcal{F}_{preL}$, $(\texttt{SETUP-OK}, id) \xhookrightarrow{\tau_1} \mathcal{E}$. Else stop.
3) If $(\texttt{UPDATE-OK}, id) \xleftarrow{\tau_1} \mathcal{E}$, $(\texttt{PRE-UPDATE-OK}, id) \xhookrightarrow{\tau_1} \mathcal{F}_{preL}$. 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 $(\texttt{REVOKE}, id) \xleftrightarrow{\tau_2} \mathcal{E}$, $(\texttt{PRE-REVOKE}, id) \xhookrightarrow{\tau_2} \mathcal{F}_{preL}$. 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 $\gamma.\mathsf{st} := \vec{\theta}$ and $\Gamma(id) := \gamma$. Then $(\texttt{UPDATED}, id) \xhookrightarrow{\tau_3} \mathcal{E}$.

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

Channels without validity

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

1) Extract γ and $\texttt{TX}_{\texttt{f}}^{\gamma}$ from $\Gamma^P(id)$ and tid_P, tid_Q from $\texttt{TX}_{\texttt{f}}^{\gamma}$. Then define $id_\alpha := \gamma.\mathsf{subchan}(P)$ 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 $\Gamma^P(\gamma.\mathsf{id}) = \bot$ and stop.
3) Let $T_2 := t_1 + T + 3\Delta$ and distinguish:
   - If in round $t_2 \le T_2$ a transaction with tid_Q appeared on $\widehat{\mathcal{L}}$, then $(\text{post}, \texttt{TX}_{\texttt{f}}^{\gamma}) \xhookrightarrow{t_2} \widehat{\mathcal{L}}$.
   - Else in round T_2 create the punishment transaction $\texttt{TX}_{\mathsf{pun}}$ as $\texttt{TX}_{\mathsf{pun}}.\mathsf{Input} := tid_P$, $\texttt{TX}_{\mathsf{pun}}.\mathsf{Output} := (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \texttt{One-Sig}_{pk_P})$ and $\texttt{TX}_{\mathsf{pun}}.\mathsf{Witness} := \mathsf{Sign}_{sk_P}([\texttt{TX}_{\mathsf{pun}}])$. Then $(\text{post}, \texttt{TX}_{\mathsf{pun}}) \xhookrightarrow{T_2} \widehat{\mathcal{L}}$.
4) Let $T_3 := t_2 + \Delta$ and distinguish the following two cases:
   - The transaction $\texttt{TX}_{\texttt{f}}^{\gamma}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3 \le T_3$, then update $\Gamma_L^P(id) := \Gamma^P(id)$ and set $m :=$ offloaded.
   - The transaction $\texttt{TX}_{\mathsf{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3 \le T_3$, then set $m :=$ punished.
5) Set $\Gamma^P(id) = \bot$ and return m in round t_3.

$\boxed{\text{Party } I}$

1) Extract γ and $\texttt{TX}_{\texttt{f}}^{\gamma}$ from $\Gamma^I(id)$ and tid_A, tid_B from $\texttt{TX}_{\texttt{f}}^{\gamma}$. Then define $id_\alpha := \gamma.\mathsf{subchan}(A)$, $id_\beta := \gamma.\mathsf{subchan}(B)$ and send the messages $(\texttt{CLOSE}, id_\alpha) \xhookrightarrow{t_0} \mathcal{F}_{preL}$ and $(\texttt{CLOSE}, id_\beta) \xhookrightarrow{t_0} \mathcal{F}_{preL}$.
2) If you receive both messages $(\texttt{CLOSED}, id_\alpha) \xleftarrow{t_1^A \le t_0 + T + 3\Delta} \mathcal{F}_{preL}$ and $(\texttt{CLOSED}, id_\beta) \xleftarrow{t_1^B \le t_0 + T + 3\Delta} \mathcal{F}_{preL}$, publish $(\text{post}, \texttt{TX}_{\texttt{f}}^{\gamma}) \xhookrightarrow{t_1} \widehat{\mathcal{L}}$, where $t_1 := \max\{t_1^A, t_1^B\}$. Otherwise set $\Gamma^I(id) = \bot$ and stop.
3) Once $\texttt{TX}_{\texttt{f}}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round $t_2 \le t_1 + \Delta$, then $\Gamma^I(id) = \bot$ and return "offloaded".

Channels with validity

$\boxed{\text{Party } A}$

1) Extract γ, tid_A and $\texttt{TX}_{\texttt{f}}^{\gamma}$ from $\Gamma^A(id)$. Let $id_\alpha := \gamma.\mathsf{subchan}(A)$ 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 post $(\text{post}, \texttt{TX}_{\texttt{f}}^{\gamma,A}) \xhookrightarrow{t_2} \widehat{\mathcal{L}}$. Otherwise, set $\Gamma^A(\gamma.\mathsf{id}) = \bot$ and stop.
3) Once $\texttt{TX}_{\texttt{f}}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round $t_2 \le t_1 + \Delta$, then update $\Gamma_L^A(id) := \Gamma^A(id)$, $\Gamma^A(id) := \bot$ and return "offloaded".

$\boxed{\text{Party } B}$

1) Extract γ, tid_B and $[\texttt{TX}_{\texttt{f}}^{\gamma}]$ from $\Gamma^B(id)$. Let $id_\beta := \gamma.\mathsf{subchan}(B)$ and send $(\texttt{CLOSE}, id_\beta) \xhookrightarrow{t_0} \mathcal{F}_{preL}$.
2) If you receive $(\texttt{CLOSED}, id_\beta) \xleftarrow{t_1 \le t_0 + T + 3\Delta} \mathcal{F}_{preL}$, then continue. Otherwise, set $\Gamma^B(\gamma.\mathsf{id}) = \bot$ and stop.
3) Create the punishment transaction $\texttt{TX}_{\mathsf{pun}}$ as $\texttt{TX}_{\mathsf{pun}}.\mathsf{Input} := tid_B$, $\texttt{TX}_{\mathsf{pun}}.\mathsf{Output} := (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \texttt{One-Sig}_{pk_B})$ and set the value $\texttt{TX}_{\mathsf{pun}}.\mathsf{Witness} := \mathsf{Sign}_{sk_B}([\texttt{TX}_{\mathsf{pun}}])$. Then wait until round $t_2 := \max\{t_1, \gamma.\mathsf{val} + 2\Delta\}$ and send $(\text{post}, \texttt{TX}_{\mathsf{pun}}) \xhookrightarrow{t_2} \widehat{\mathcal{L}}$.
4) Let $T_3 := t_2 + \Delta$ and distinguish the following two cases:
   - A transaction with identifier $\texttt{TX}_{\texttt{f}}^{\gamma}.\mathsf{txid}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3 \le T_3$, then define $\Gamma_L^B(id) := \Gamma^B(id)$ and set $m :=$ offloaded.
   - The transaction $\texttt{TX}_{\mathsf{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3 \le T_3$, 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, $\mathtt{TX}^{\gamma}_{\mathtt{refund}}$ and $[\mathtt{TX}^{\gamma}_{\mathtt{f}}]$ 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 $(\mathtt{CLOSED}, id_\alpha) \xhookleftarrow{t_1^A \le t_0+T+3\Delta} \mathcal{F}_{preL}$ and $(\mathtt{CLOSED}, id_\beta) \xhookleftarrow{t_1^B \le t_0+T+3\Delta} \mathcal{F}_{preL}$, then continue. Otherwise, set $\Gamma^I(\gamma.\mathsf{id}) = \bot$ and stop.
3) Create the punishment transaction $\mathtt{TX}_{\mathtt{pun}}$ as $\mathtt{TX}_{\mathtt{pun}}.\mathsf{Input} := tid_A$, $\mathtt{TX}_{\mathtt{pun}}.\mathsf{Output} := (\gamma.\mathsf{cash}+\gamma.\mathsf{fee}/2, \mathtt{One\text{–}Sig}_{pk_I})$ and set the value $\mathtt{TX}_{\mathtt{pun}}.\mathsf{Witness} := \mathsf{Sign}_{sk_I}([\mathtt{TX}_{\mathtt{pun}}])$. Then wait until round $t_2 := \max\{t_1^A, \gamma.\mathsf{val}\}$ and send $(\text{post}, \mathtt{TX}_{\mathtt{pun}}) \overset{t_2}{\hookrightarrow} \widehat{\mathcal{L}}$.
4) Let $T_3 := t_2 + \Delta$ and distinguish the following two cases:
   - A transaction with identifier $\mathtt{TX}^{\gamma}_{\mathtt{f}}.\mathsf{txid}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3' \le T_3$, send $(\text{post}, \mathtt{TX}^{\gamma}_{\mathtt{refund}}) \overset{t_4}{\hookrightarrow} \widehat{\mathcal{L}}$ where $t_4 := \max\{t_1^B, t_3'\}$. Once $\mathtt{TX}^{\gamma}_{\mathtt{refund}}$ is accepted by $\widehat{\mathcal{L}}$ in round $t_5 \le t_4+\Delta$, then define $m :=$ offloaded and $\Gamma^I(\gamma.\mathsf{id}) = \bot$.
   - The transaction $\mathtt{TX}_{\mathtt{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3'' \le T_3$, then define $m :=$ punished and $\Gamma^I(\gamma.\mathsf{id}) = \bot$.
5) Return m in round t_6 where $t_6 := \max\{t_5, t_3''\}$.

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

1) Extract γ, $\mathtt{TX}^{\gamma}_{\mathtt{f}}$ from $\Gamma^P(id)$.
2) Parse $\gamma.\mathsf{st} = \Big((c_P, \mathtt{One\text{–}Sig}_{pk_P}), (c_Q, \mathtt{One\text{–}Sig}_{pk_Q})\Big)$.
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 $(\mathtt{SETUP}, id_\alpha, tid_P) \xhookleftarrow{t_1^P \le t_0^P + T} \mathcal{F}_{preL}$, send $(\mathtt{SETUP\text{–}OK}, id_\alpha) \overset{t_1^P}{\hookrightarrow} \mathcal{F}_{preL}$.

Party I

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

1) Extract γ, $\mathtt{TX}^{\gamma}_{\mathtt{f}}$ 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 $(\mathtt{UPDATE\text{–}REQ}, id_\alpha, tid_A, \vec{\theta}_A, 0) \hookleftarrow \mathcal{F}_{preL}$ and $(\mathtt{UPDATE\text{–}REQ}, id_\beta, tid_B, \vec{\theta}_B, 0) \hookleftarrow \mathcal{F}_{preL}$ check that for some c_A, c_B s.t. $c_A + c_B = c$ it holds
$$\vec{\theta}_A = \{(c_A, \mathtt{One\text{–}Sig}_{pk_A}), (c_B + \gamma.\mathsf{fee}/2, \mathtt{One\text{–}Sig}_{pk_I})\}$$
$$\vec{\theta}_B = \{(c_B, \mathtt{One\text{–}Sig}_{pk_B}), (c_A + \gamma.\mathsf{fee}/2, \mathtt{One\text{–}Sig}_{pk_I})\}$$
If not, then stop.
4) If in round $t_2^I \le t_1^I + T$ you receive both $(\mathtt{SETUP\text{–}OK}, id_\alpha)) \hookleftarrow \mathcal{F}_{preL}$ and $(\mathtt{SETUP\text{–}OK}, id_\beta)) \hookleftarrow \mathcal{F}_{preL}$, send $(\mathtt{UPDATE\text{–}OK}, id_\alpha) \overset{t_2^I}{\hookrightarrow} \mathcal{F}_{preL}$ and $(\mathtt{UPDATE\text{–}OK}, id_\beta) \overset{t_2^I}{\hookrightarrow} \mathcal{F}_{preL}$. If not, then stop.

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

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

Party I

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

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

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

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

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

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

Punish

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

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

1) Extract $\mathtt{TX}_\mathtt{f}^\gamma$ from $\Gamma^P(id)$ and tid_P, tid_Q from $\mathtt{TX}_\mathtt{f}^\gamma$. 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 $t_2 \leq T_2$ the transaction with tid_Q appeared on $\widehat{\mathcal{L}}$, then $(\text{post}, \mathtt{TX}_\mathtt{f}^\gamma) \overset{t_2}{\hookrightarrow} \widehat{\mathcal{L}}$.
   - Else in round T_2 create the punishment transaction $\mathtt{TX}_\mathsf{pun}$ as

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

   and $(\text{post}, \mathtt{TX}_\mathsf{pun}) \overset{T_2}{\hookrightarrow} \widehat{\mathcal{L}}$.
3) Let $T_3 := t_2 + \Delta$ and distinguish the following two cases:
   - The transaction $\mathtt{TX}_\mathtt{f}^\gamma$ was accepted by $\widehat{\mathcal{L}}$ in $t_3 \leq T_3$, then $\Gamma_L^P(id) := \Gamma^P(id)$, $\Gamma^P(id) = \perp$ and $m := \mathtt{OFFLOADED}$.
   - The transaction $\mathtt{TX}_\mathsf{pun}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3 \leq T_3$, then define $\Gamma^P(\gamma.\mathsf{id}) = \perp$ and set $m := \mathtt{PUNISHED}$.
4) Output $(m, id) \overset{t_3}{\hookrightarrow} \mathcal{E}$.

Party I

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

1) Extract $\mathtt{TX}_\mathtt{f}^\gamma$ from $\Gamma^I(id)$ and tid_A, tid_B from $\mathtt{TX}_\mathtt{f}^\gamma$. 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 $id_\alpha := \gamma.\mathsf{subchan}(Q)$ and send $(\mathtt{CLOSE}, id_\alpha) \overset{t_0}{\hookrightarrow} \mathcal{F}_{preL}$.
3) If you receive $(\mathtt{CLOSED}, id_\alpha) \xleftarrow{t_1 \leq t_0 + T + 3\Delta} \mathcal{F}_{preL}$ and tid_Q appeared on $\widehat{\mathcal{L}}$, $(\text{post}, \mathtt{TX}_\mathtt{f}^\gamma) \overset{t_1}{\hookrightarrow} \widehat{\mathcal{L}}$. Otherwise set $\Gamma^I(id) = \perp$ and stop.
4) Once $\mathtt{TX}_\mathtt{f}^\gamma$ is accepted by $\widehat{\mathcal{L}}$ in round t_2, such that $t_2 \leq t_1 + \Delta$, set $\Gamma^I(id) = \perp$ and output $(\mathtt{OFFLOADED}, id) \overset{t_2}{\hookrightarrow} \mathcal{E}$.

Punish–Validity

Party A

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

1) Extract $\mathtt{TX}_\mathtt{f}^\gamma$ from $\Gamma^A(id)$ and tid_A from $\mathtt{TX}_\mathtt{f}^\gamma$. If tid_A appeared on $\widehat{\mathcal{L}}$, then send $(\text{post}, \mathtt{TX}_\mathtt{f}^\gamma) \overset{t_1}{\hookrightarrow} \widehat{\mathcal{L}}$. Else stop.
2) Once $\mathtt{TX}_\mathtt{f}^\gamma$ is accepted by $\widehat{\mathcal{L}}$ in round $t_2 \leq t_1 + \Delta$, set $\Gamma_L^A(id) := \Gamma^A(id)$, $\Gamma^A(id) := \perp$ and output $(\mathtt{OFFLOADED}, id) \overset{t_2}{\hookrightarrow} \mathcal{E}$.

Party B

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

1) Extract tid_B and $[\mathtt{TX}_\mathtt{f}^\gamma]$ from $\Gamma^B(id)$. Check if tid_B or $[\mathtt{TX}_\mathtt{f}^\gamma].\mathsf{txid}$ appeared on $\widehat{\mathcal{L}}$. If not, then stop.
2) If a transaction $\mathtt{TX}_\mathtt{f}^\gamma$ appeared on $\widehat{\mathcal{L}}$, update set $\Gamma_L^B(id) := \Gamma^B(id)$ and $\Gamma^B(id) := \perp$. Then output $(\mathtt{OFFLOADED}, id) \overset{t_1}{\hookrightarrow} \mathcal{E}$ and stop.
3) If tid_B appeared on $\widehat{\mathcal{L}}$, create the punishment transaction $\mathtt{TX}_\mathsf{pun}$ as

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

   Then wait until round $t_2 := \max\{t_1, \gamma.\mathsf{val} + 2\Delta\}$ and send $(\text{post}, \mathtt{TX}_\mathsf{pun}) \overset{t_2}{\hookrightarrow} \widehat{\mathcal{L}}$.
4) If transaction $\mathtt{TX}_\mathsf{pun}$ was accepted by $\widehat{\mathcal{L}}$ in $t_3 \leq t_2 + \Delta$, then define $\Gamma^B(\gamma.\mathsf{id}) = \perp$ and output $(\mathtt{PUNISHED}, id) \overset{t_3}{\hookrightarrow} \mathcal{E}$.

Party I

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

1) Extract tid_A, tid_B, $\mathtt{TX}_\mathtt{refund}^\gamma$ and $[\mathtt{TX}_\mathtt{f}^\gamma]$ from $\Gamma^I(id)$. Check if tid_A or tid_B appeared on $\widehat{\mathcal{L}}$ or $t_1 = \gamma.\mathsf{val} - (3\Delta + T)$. If not, then stop.
2) Distinguish the following cases:
   - If $t_1 = \gamma.\mathsf{val} - (3\Delta + T)$, define $id_\alpha := \gamma.\mathsf{subchan}(A)$, $id_\beta := \gamma.\mathsf{subchan}(B)$ and send $(\mathtt{CLOSE}, id_\alpha) \overset{t_1}{\hookrightarrow} \mathcal{F}_{preL}$ and $(\mathtt{CLOSE}, id_\beta) \overset{t_1}{\hookrightarrow} \mathcal{F}_{preL}$.
   - If tid_B appeared on $\widehat{\mathcal{L}}$, send $(\mathtt{CLOSE}, id_\alpha) \overset{t_1}{\hookrightarrow} \mathcal{F}_{preL}$.
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 $\mathtt{TX}_\mathsf{pun}$ as

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

   Then wait until round $t_3 := \max\{t_2, \gamma.\mathsf{val}\}$ and send $(\text{post}, \mathtt{TX}_\mathsf{pun}) \overset{t_3}{\hookrightarrow} \widehat{\mathcal{L}}$.
4) Distinguish the following two cases:
   - The transaction $\mathtt{TX}_\mathtt{f}^\gamma.\mathsf{txid}$ was accepted by $\widehat{\mathcal{L}}$ in $t_4 \leq t_3 + \Delta$, send $(\text{post}, \mathtt{TX}_\mathtt{refund}^\gamma) \overset{t_5}{\hookrightarrow} \widehat{\mathcal{L}}$ where $t_5 := \max\{\gamma.\mathsf{val} + \Delta, t_4\}$. Once $\mathtt{TX}_\mathtt{refund}^\gamma$ is accepted by $\widehat{\mathcal{L}}$ in round $t_6 \leq t_5 + \Delta$, set $\Gamma^I(\gamma.\mathsf{id}) = \perp$ and output $(\mathtt{OFFLOADED}, id) \overset{t_6}{\hookrightarrow} \mathcal{E}$ and stop.
   - The transaction $\mathtt{TX}_\mathsf{pun}$ was accepted by $\widehat{\mathcal{L}}$ in $t_4 \leq t_3 + \Delta$, then set $\Gamma^I(\gamma.\mathsf{id}) = \perp$ and output $(\mathtt{PUNISHED}, id) \overset{t_4}{\hookrightarrow} \mathcal{E}$.

E. *Wrappers for Missing Checks*

In the previous sections, we provided simplified descriptions of the ideal functionalities $\mathcal{F}_L$, $\mathcal{F}_{preL}$ and $\mathcal{F}_V$ as well as of the protocols Π_{preL} and Π_V. The simplification stems from the fact that we excluded several natural checks in the ideal functionalities and protocols. In this section, we present wrappers that includes these missing checks.

1) Wrapper for Ideal Functionalities: 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 checks formally in the wrapper below which can be seen as an extension to the wrapper provided by [2] 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 $A := \gamma.\mathsf{Alice}$, $B := \gamma.\mathsf{Bob}$ and $I := \gamma.\mathsf{Ingrid}$.

Create: Upon $(\mathtt{CREATE}, \gamma, tid) \overset{\tau_0}{\hookleftarrow} P$, where $P \in \gamma.\mathsf{users}$, check if: $\Gamma(\gamma.\mathsf{id}) = \perp$, $\mathcal{F}.\Gamma_{pre}(\gamma.\mathsf{id}) = \perp$ 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 III-A; $\gamma.\mathsf{st} = \{(c_P, \mathtt{One\text{–}Sig}_{pk_P}), (c_Q, \mathtt{One\text{–}Sig}_{pk_Q})\}$ 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 $\alpha := \mathcal{F}.\Gamma(id_P) \neq \perp$ 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 $\alpha := \mathcal{F}.\Gamma(id_A) \neq \perp$ for $id_A := \gamma.\mathsf{subchan}(A)$; $\beta := \mathcal{F}.\Gamma(id_B) \neq \perp$ 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 $\gamma.\mathsf{val} \neq \perp$, then $\gamma.\mathsf{val} \geq \tau_0 + 4\Delta + 15T$.

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

Pre-Create: Upon $(\mathtt{PRE\text{–}CREATE}, \gamma, \mathtt{TX_f}, i, t_{ofl}) \overset{\tau_0}{\hookleftarrow} P$, check if: $P \in \gamma.\mathsf{users}$, $\mathcal{F}.\Gamma_{pre}(\gamma.\mathsf{id}) = \perp$, $\mathcal{F}.\Gamma_{pre}(\gamma.\mathsf{id}) = \perp$ 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 III-A; $\gamma.\mathsf{st} = \{(c_P, \mathtt{One\text{–}Sig}_{pk_P}), (c_Q, \mathtt{One\text{–}Sig}_{pk_Q})\}$ 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 $\mathcal{F}$.

(Pre)-Update: Upon $(m, id, \vec{\theta}, t_{\mathsf{stp}}) \overset{\tau_0}{\hookleftarrow} P$, check if: $\gamma := \Gamma(id) \neq \perp$ if $m = \mathtt{UPDATE}$ and $\gamma := \Gamma_{pre}(id) \neq \perp$ if $m = \mathtt{PRE\text{–}UPDATE}$. In both cases additionally check: $P \in \gamma.\mathsf{endUsers}$; 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 $\sum_{j\in[\ell]} c_i = \gamma.\mathsf{cash}$ 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 $((\mathtt{PRE\text{–}})\mathtt{SETUP\text{–}OK}, id) \overset{\tau_2}{\hookleftarrow} P$ check if: you accepted a message $((\mathtt{PRE\text{–}})\mathtt{UPDATE}, id, \vec{\theta}, t_{\mathsf{stp}}) \overset{\tau_0}{\hookleftarrow} P$, where $t_2 - t_0 \leq t_{\mathsf{stp}} + T$ and the message is a reply to the message $((\mathtt{PRE\text{–}})\mathtt{SETUP}, id, tid)$ sent to P in round τ_1 such that $\tau_2 - \tau_1 \leq t_{\mathsf{stp}}$[b]. If not, drop the message. Else proceed as $\mathcal{F}$.

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

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

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

Close: Upon $(\mathtt{CLOSE}, id) \overset{\tau_0}{\hookleftarrow} P$, check if $\gamma := \Gamma(id) \neq \perp$ and $P \in \gamma.\mathsf{endUsers}$. If γ is a virtual channel, additionally check that $\gamma.\mathsf{val} = \perp$. 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 A what we formally meant by "reply".

2) *Wrapper for Protocols:* Similar to the descriptions of our ideal functionality, the description of our channel protocols, the protocol Π_{preL} presented in Appendix C3 and the protocol Π_V, 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 [2] 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 $I := \gamma.\mathsf{Ingrid}$.

$\boxed{\text{Party } P}$

Create: Upon $(\mathtt{CREATE}, \gamma, tid) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$ check if: $P \in \gamma.\mathsf{endUsers}$; $\Gamma^P(\gamma.\mathsf{id}) = \perp$, $\Gamma^P_{pre}(\gamma.\mathsf{id}) = \perp$ 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 III-A; $\gamma.\mathsf{st} = \{(c_P, \mathtt{One\text{–}Sig}_{pk_P}), (c_Q, \mathtt{One\text{–}Sig}_{pk_Q})\}$ 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 $\alpha := \Gamma^P(id_P) \neq \perp$ 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 $\alpha := \Gamma^P(id_A) \neq \perp$ for $id_A := \gamma.\mathsf{subchan}(A)$; $\beta := \Gamma^P(id_B) \neq \perp$ 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 $\gamma.\mathsf{val} \neq \perp$, 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: $P \in \gamma.\mathsf{users}$, $\Gamma^P_{pre}(\gamma.\mathsf{id}) = \perp$, $\Gamma^P_{pre}(\gamma.\mathsf{id}) = \perp$ 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 III-A; $\gamma.\mathsf{st} = \{(c_P, \mathtt{One\text{-}Sig}_{pk_P}), (c_Q, \mathtt{One\text{-}Sig}_{pk_Q})\}$ 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 $(m, id, \vec{\theta}, t_{\mathsf{stp}}) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$ check if: $\gamma := \Gamma^P(id) \neq \perp$ if $m = \mathtt{UPDATE}$ and $\gamma := \Gamma^P_{pre}(id) \neq \perp$ 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 $((\mathtt{PRE\text{-}})\mathtt{UPDATE}, id, \vec{\theta}, t_{\mathsf{stp}}) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$, where $t_2 - t_0 \leq t_{\mathsf{stp}} + T$ 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 $\tau_2 - \tau_1 \leq t_{\mathsf{stp}}$[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: $\gamma := \Gamma(id) \neq \perp$ 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 $\gamma := \Gamma^P(id) \neq \perp$, $P \in \gamma.\mathsf{endUsers}$. If γ is a virtual channel, additionally check that $\gamma.\mathsf{val} = \perp$. 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 A what we formally meant by "reply".

F. Security Proofs

In this section we provide proofs for Theorem 2, Theorem 1 and Theorem 3.

1) Proof of Theorem 2: In our proof of Theorem 2, 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 $(\mathtt{PRE\text{-}CREATE}, \gamma, \mathtt{TX_f}, i, t_{ofl}) \overset{\tau_0}{\hookrightarrow} \mathcal{F}_{preL}$ set $T_1 = 2$ and do the following:
2) If $\mathtt{TX_f}.\mathsf{Output}[i].\mathsf{cash} \neq \gamma.\mathsf{cash}$, then ignore the message.
3) Set $id := \gamma.\mathsf{id}$, generate $(R_A, r_A) \leftarrow \mathsf{GenR}$, $(Y_A, y_A) \leftarrow \mathsf{GenR}$ and send $(\text{createInfo}, id, \mathtt{TX_f}, i, t_{ofl}, R_A, Y_A) \overset{\tau_0}{\hookrightarrow} B$.
4) If $(\text{createInfo}, id, \mathtt{TX_f}, i, t_{ofl}, R_B, Y_B) \overset{\tau_0+1}{\hookleftarrow} B$, create:
$$[\mathtt{TX_c}] := \mathtt{GenCommit}([\mathtt{TX_f}], I_A, I_B, 0)$$
$$[\mathtt{TX_s}] := \mathtt{GenSplit}([\mathtt{TX_c}].\mathsf{txid}\|1, \gamma.\mathsf{st})$$
for $I_A := (pk_A, R_B, Y_A)$, $I_B := (pk_B, R_B, Y_B)$. Else stop.
5) Compute $s_c^A \leftarrow \mathsf{pSign}_{sk_A}([\mathtt{TX_c}], Y_B)$, $s_s^A \leftarrow \mathsf{Sign}_{sk_A}([\mathtt{TX_s}])$ and send $(\text{createCom}, id, s_c^A, s_s^A) \overset{\tau_0+1}{\hookrightarrow} B$.
6) If $(\text{createCom}, id, s_c^B, s_s^B) \overset{\tau_0+2}{\hookleftarrow} B$, s.t. $\mathsf{pVrfy}_{pk_B}([\mathtt{TX_c}], Y_A; s_c^B) = 1$ and $\mathsf{Vrfy}_{pk_B}([\mathtt{TX_s}]; s_s^B) = 1$, set
$$\mathtt{TX_c} := ([\mathtt{TX_c}], \{\mathsf{Sign}_{sk_A}([\mathtt{TX_c}]), \mathsf{Adapt}(s_c^B, y_A)\})$$
$$\mathtt{TX_s} := ([\mathtt{TX_s}], \{s_s^A, s_s^B\})$$
$$\Gamma^A_{pre}(\gamma.\mathsf{id}) := (\gamma, \mathtt{TX_f}, (\mathtt{TX_c}, r_A, R_B, Y_B, s_c^A), \mathtt{TX_s}, t_{ofl}).$$
and if B has not sent $(\mathtt{PRE\text{-}CREATE}, \gamma, \mathtt{TX_f}, i, t_{ofl})$ to $\mathcal{F}_{preL}$ send this message on behalf of B.

Pre-Update

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

$\boxed{\text{Case } A \text{ is honest and } B \text{ is corrupted}}$

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

1) Generate new revocation public/secret pair $(R_P, r_P) \leftarrow \mathsf{GenR}$ and a new publishing public/secret pair $(Y_P, y_P) \leftarrow \mathsf{GenR}$ and send $(\text{updateReq}, id, \vec{\theta}, t_{\mathsf{stp}}, R_A, Y_A) \xhookrightarrow{\tau_0^A} B$.
2) Upon $(\text{updateInfo}, id, h_B, Y_B, s_{\mathsf{s}}^B) \xhookleftarrow{\tau_0^A+2} B$, set $t_{\mathsf{lock}} := \tau_0^A + t_{\mathsf{stp}} + 5 + \Delta + t_{ofl}$, extract $\texttt{TX}_\mathtt{f}$ from $\Gamma^B_{pre}(id)$ and
$$[\texttt{TX}_\mathsf{c}] := \texttt{GenCommit}([\texttt{TX}_\mathtt{f}], I_A, I_B, t_{\mathsf{lock}})$$
$$[\texttt{TX}_\mathsf{s}] := \texttt{GenSplit}([\texttt{TX}_\mathsf{c}].\mathsf{txid}\|1, \vec{\theta}),$$
for $I_A := (pk_A, R_A, Y_A)$ and $I_B := (pk_B, R_B, Y_B)$. If it holds that $\mathsf{Vrfy}_{pk_B}([\texttt{TX}_\mathsf{s}]; s_{\mathsf{s}}^B) = 1$ continue. Else mark this execution as "failed" and stop.
3) If A sends $(\texttt{PRE–SETUP–OK}, id) \xleftrightarrow{\tau_1^A \le \tau_0^A+2+t_{\mathsf{stp}}} \mathcal{F}_{preL}$, compute $s_{\mathsf{c}}^A \leftarrow \mathsf{pSign}_{sk_A}([\texttt{TX}_\mathsf{c}], Y_B), s_{\mathsf{s}}^A \leftarrow \mathsf{Sign}_{sk_A}([\texttt{TX}_\mathsf{s}])$ and send the message $(\text{update–commitA}, id, s_{\mathsf{c}}^A, s_{\mathsf{s}}^A) \xhookrightarrow{\tau_1^A} B$.
4) In round $\tau_1^A + 2$ distinguish the following cases:
   - If A receives $(\text{update–commitB}, id, s_{\mathsf{c}}^B) \xhookleftarrow{\tau_1^A+2} B$ 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 $\mathsf{pVrfy}_{pk_B}([\texttt{TX}_\mathsf{c}], Y_A; s_{\mathsf{c}}^B) = 0$, 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 $\Theta^A(id) := \Theta^A(id) \cup ([\texttt{TX}_\mathsf{c}], r_B, Y_B, s_{\mathsf{c}}^A)$, 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 $\Gamma^A_{pre}(id)$ as $(\gamma, \texttt{TX}_\mathtt{f}, (\overline{\texttt{TX}}_\mathsf{c}, \bar{r}_A, \bar{R}_B, \bar{Y}_B, \bar{s}^A_{\mathtt{Com}}), \overline{\texttt{TX}}_\mathsf{s})$ and update the channel space as $\Gamma^A_{pre}(id) := (\gamma, \texttt{TX}_\mathtt{f}, (\texttt{TX}_\mathsf{c}, r_A, R_B, Y_B, s_{\mathsf{c}}^A), \texttt{TX}_\mathsf{s})$, for $\texttt{TX}_\mathsf{s} := ([\texttt{TX}_\mathsf{s}], \{s_{\mathsf{s}}^A, s_{\mathsf{s}}^B\})$ and $\texttt{TX}_\mathsf{c} := ([\texttt{TX}_\mathsf{c}], \{\mathsf{Sign}_{sk_A}([\texttt{TX}_\mathsf{c}]), \mathsf{Adapt}(s_{\mathsf{c}}^B, y_A)\})$. 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
$$\Theta^B(id) := \Theta^A(id) \cup ([\overline{\texttt{TX}}_\mathsf{c}], \bar{r}_B, \bar{Y}_B, \bar{s}^A_{\mathtt{Com}})$$
Else execute the simulator code for the procedure $\texttt{Wait–if–Register}^A(id)$ and stop.

$\boxed{\text{Case } B \text{ is honest and } A \text{ is corrupted}}$

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

1) Upon $(\text{updateReq}, id, \vec{\theta}, t_{\mathsf{stp}}, R_A, Y_A) \xhookleftarrow{\tau_0^B} A$, generate $(R_B, r_B) \leftarrow \mathsf{GenR}$ and $(Y_B, y_B) \leftarrow \mathsf{GenR}$.
2) Set $t_{\mathsf{lock}} := \tau_0^B + t_{\mathsf{stp}} + 4 + \Delta + t_{ofl}$, extract $\texttt{TX}_\mathtt{f}$ from $\Gamma^A_{pre}(id)$ and
$$[\texttt{TX}_\mathsf{c}] := \texttt{GenCommit}([\texttt{TX}_\mathtt{f}], I_A, I_B, t_{\mathsf{lock}})$$
$$[\texttt{TX}_\mathsf{s}] := \texttt{GenSplit}([\texttt{TX}_\mathsf{c}].\mathsf{txid}\|1, \vec{\theta})$$
where $I_A := (pk_A, R_A, Y_A)$, $I_B := (pk_B, R_B, Y_B)$.
3) Compute $s_{\mathsf{s}}^B \leftarrow \mathsf{Sign}_{sk_B}([\texttt{TX}_\mathsf{s}])$, send $(\text{updateInfo}, id, R_B, Y_B, s_{\mathsf{s}}^B) \xhookrightarrow{\tau_0^B} A$.
4) If B receives $(\text{update–commitA}, id, s_{\mathsf{c}}^A, s_{\mathsf{s}}^A) \xhookleftarrow{\tau_1^B \le \tau_0^B+2+t_{\mathsf{stp}}} A$ 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 $\mathsf{pVrfy}_{pk_P}([\texttt{TX}_\mathsf{c}], Y_Q; s_{\mathsf{c}}^P) = 1$ and $\mathsf{Vrfy}_{pk_P}([\texttt{TX}_\mathsf{s}]; s_{\mathsf{s}}^P) = 1$. 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 $s_{\mathsf{c}}^B \leftarrow \mathsf{pSign}([\texttt{TX}_\mathsf{c}], Y_A)$ and send $(\text{update–commitB}, id, s_{\mathsf{c}}^B) \xhookrightarrow{\tau_1^B} A$. 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 $(\gamma, \texttt{TX}_\mathtt{f}, (\overline{\texttt{TX}}_\mathsf{c}, \bar{r}_B, \bar{R}_A, \bar{Y}_A, \bar{s}^B_{\mathtt{Com}}), \overline{\texttt{TX}}_\mathsf{s})$. If B receives $(\text{revokeA}, id, \bar{r}_A) \xhookleftarrow{\tau_1^B+2} A$, send $(\texttt{PRE–REVOKE}, id) \xhookrightarrow{\tau_1^B+2} \mathcal{F}_{preL}$ on behalf of A, if A has not sent this message.

   Else if you do not receive $(\text{revokeA}, id, \bar{r}_A) \xhookleftarrow{\tau_1^B+2} A$ or if $(\bar{R}_A, \bar{r}_A) \notin R$, execute the simulator code of the procedure $\texttt{Wait–if–Register}^B(id)$ and stop.
8) If B sends $(\texttt{PRE–REVOKE}, id) \xhookrightarrow{\tau_1^B+2} \mathcal{F}_{preL}$, then set
$$\Theta^B(id) := \Theta^B(id) \cup ([\overline{\texttt{TX}}_\mathsf{c}], \bar{r}_A, \bar{Y}_A, \bar{s}^B_{\mathtt{Com}})$$
$$\Gamma^B_{pre}(id) := (\gamma, \texttt{TX}_\mathtt{f}, (\texttt{TX}_\mathsf{c}, r_B, R_A, Y_A, s_{\mathsf{c}}^B), \texttt{TX}_\mathsf{s}),$$
for $\texttt{TX}_\mathsf{s} := ([\texttt{TX}_\mathsf{s}], \{s_{\mathsf{s}}^A, s_{\mathsf{s}}^B\})$ and $\texttt{TX}_\mathsf{c} := ([\texttt{TX}_\mathsf{c}], \{\mathsf{Sign}_{sk_B}([\texttt{TX}_\mathsf{c}]), \mathsf{Adapt}(s_{\mathsf{c}}^A, y_B)\})$. Then $(\text{revokeB}, id, \bar{r}_B) \xhookrightarrow{\tau_1^B+2} A$ and stop. Else, in round $\tau_1^B + 2$, execute the simulator code of the procedure $\texttt{Wait–if–Register}^B(id)$ and stop.

Register

$\boxed{\text{Case } A \text{ honest and } B \text{ corrupted}}$

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 $\Gamma^A_{pre}(id) := (\gamma, \texttt{TX}_\mathtt{f}, (\texttt{TX}_\mathsf{c}, r_A, R_B, Y_B, s_{\mathsf{c}}^A), \texttt{TX}_\mathsf{s}, t_{ofl}, x)$
3) If $\texttt{TX}_\mathtt{f}$ appeared on-chain in this round, then
   a) Set $\Gamma(id) := (\gamma, \texttt{TX}_\mathtt{f}, (\texttt{TX}_\mathsf{c}, r_A, R_B, Y_B, s_{\mathsf{c}}^A), \texttt{TX}_\mathsf{s})$.

b) Set $\Gamma^A_{pre}(id) := \bot$
c) If $x = \texttt{in–dispute}$, then execute the simulator code for $\texttt{L–ForceClose}^A(id)$.

$\texttt{Wait–if–Register}(id)$

$\boxed{\text{Case } A \text{ honest and } B \text{ corrupted}}$

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

2) Proof of Theorems 1 and 3: We now provide a proof for Theorem 1 and Theorem 3. 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 III-B.

Simulator for creating virtual channels

creating virtual channels

$\boxed{\text{Case } A \text{ is honest and } I \text{ are } B \text{ are corrupt}}$

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

1) Let $id_\alpha := \gamma.\mathsf{subchan}(A)$ and compute
$$\theta_A := \texttt{GenVChannelOutput}(\gamma, A).$$
2) Upon A sending $(\texttt{UPDATE}, id_\alpha, \theta_A, t_{\mathsf{stp}}) \xhookrightarrow{\tau_0^A} \mathcal{F}_L$ execute the simulator code of the update procedure for the generalized channels until the message $(\texttt{SETUP}, id_\alpha, tid_A)$ is sent by $\mathcal{F}_L$. If the execution stops send $(\text{peaceful–reject}, id_\alpha) \hookrightarrow \mathcal{F}_V$.
3) Upon A receiving $(\texttt{SETUP}, id_\alpha, tid_A) \xhookleftarrow{\tau_1^A \le \tau_0^A + T} \mathcal{F}_L$, execute the simulator code for $\texttt{SetupVChannel}$ with input (γ, tid_A).
4) If this execution of $\texttt{SetupVChannel}$ 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 $\texttt{L–ForceClose}(id_\alpha)$.
5) If B or I have not sent $(\texttt{CREATE}, \gamma) \hookrightarrow \mathcal{F}_V$ send this message on their behalf.
6) Upon A receiving $(\texttt{CREATED}, \gamma) \xhookleftarrow{\tau_2^A \le \tau_1^A + 5T} \mathcal{F}_V$, mark γ as created, i.e. update $\Gamma^A(\gamma.\mathsf{id})$ from $(\bot, x)$ to (γ, x).

$\boxed{\text{Case } I \text{ is honest and } A, B \text{ are corrupted}}$

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

1) Set $id_\alpha = \gamma.\mathsf{subchan}(A), id_\beta = \gamma.\mathsf{subchan}(B)$ and generate
$$\theta_A := \texttt{GenVChannelOutput}(\gamma, A)$$
$$\theta_B := \texttt{GenVChannelOutput}(\gamma, B)$$
2) Upon A and B sending $(\texttt{UPDATE}, id_\alpha, \theta_A, t_{\mathsf{stp}}) \xhookrightarrow{\tau_0^A} \mathcal{F}_L$ and $(\texttt{UPDATE}, id_\alpha, \theta_B, t_{\mathsf{stp}}) \xhookrightarrow{\tau_0^B} \mathcal{F}_L$, execute the simulator code for the update procedure of the generalized channel functionality until the message $(\texttt{UPDATE–REQ}, id_\alpha, \theta_A, t_{\mathsf{stp}}, tid_A)$ and $(\texttt{UPDATE–REQ}, id_\alpha, \theta_B, t_{\mathsf{stp}}, tid_B)$ are sent by $\mathcal{F}_L$.
3) If in round $\tau_1^I \le \tau_0^I + T$, I has received both $(\texttt{UPDATE–REQ}, id_\alpha, \theta_A, t_{\mathsf{stp}}, tid_A) \hookleftarrow \mathcal{F}_L$ and $(\texttt{UPDATE–REQ}, id_\beta, \theta_B, t_{\mathsf{stp}}, tid_B) \hookleftarrow \mathcal{F}_L$, then execute the simulator code of $\texttt{SetupVChannel}$ with inputs (γ, tid_A, tid_B). Or send $(\text{peaceful–reject}, id_\alpha) \hookrightarrow \mathcal{F}_V$ and $(\text{peaceful–reject}, id_\beta) \hookrightarrow \mathcal{F}_V$ if instructed by $\mathcal{E}$. Else stop.
4) If in round $\tau_2^I \le \tau_1^I + t_{\mathsf{stp}} + T$, I receives both $(\texttt{SETUP–OK}, id_\alpha)) \hookleftarrow \mathcal{F}_L$ and $(\texttt{SETUP–OK}, id_\beta)) \hookleftarrow \mathcal{F}_L$, continue executing the simulator code of the update procedure of generalized channels until the messages $(\texttt{REVOKE–REQ}, id_\alpha)$ and $(\texttt{REVOKE–REQ}, id_\alpha)$ are sent by $\mathcal{F}_L$. Otherwise stop.
5) If in round $\tau_3^I \le \tau_2^I + 4T$ you have received both $(\texttt{REVOKE–REQ}, id_\alpha) \hookleftarrow \mathcal{F}_L$ and $(\texttt{REVOKE–REQ}, id_\beta) \hookleftarrow \mathcal{F}_L$, 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 $(\texttt{CREATE}, \gamma) \hookrightarrow \mathcal{F}_V$ send this message on their behalf. Update $\Gamma^I(\gamma.\mathsf{id})$ from $(\bot, x)$ to (γ, x).

$\texttt{SetupVChannel}$ for channels without validity

$\boxed{\text{Case } A \text{ is honest and } I, B \text{ are corrupted}}$

1) Create the body of the funding transaction:
$$\texttt{TX}^\gamma_\texttt{f}.\mathsf{Input} := (tid_A, tid_B)$$
$$\texttt{TX}^\gamma_\texttt{f}.\mathsf{Output} := ((\gamma.\mathsf{cash}, \texttt{Multi–Sig}_{\{\gamma.\mathsf{endUsers}\}}), (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}, \texttt{One–Sig}_{pk_I}))$$
2) Upon A sending $(\texttt{PRE–CREATE}, \gamma, \texttt{TX}_\texttt{f}, 1, t_{ofl}) \xhookrightarrow{t_0} \mathcal{F}_{preL}$ 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 $(\texttt{PRE–CREATED}, \gamma.id) \xhookleftarrow{\tau_1 \le \tau_0 + T} \mathcal{F}_{preL}$ then sign the funding transaction, i.e. $s^B_\texttt{f} \leftarrow \mathsf{Sign}_{sk_B}([\texttt{TX}^\gamma_\texttt{f}])$ and send $(\text{createFund}, \gamma.\mathsf{id}, s^A_\texttt{f}, [\texttt{TX}^\gamma_\texttt{f}]) \xhookrightarrow{\tau_1} I$. Else record this execution as "failed" and stop.
4) Upon receiving $(\text{createFund}, \gamma.\mathsf{id}, s^B_\texttt{f}, s^I_\texttt{f}) \xhookleftarrow{\tau_1 + 1} I$, verify all signatures, i.e. check:
$$\mathsf{Vrfy}_{pk_B}([\texttt{TX}^\gamma_\texttt{f}]; s^B_\texttt{f}) = 1$$
$$\mathsf{Vrfy}_{pk_I}([\texttt{TX}^\gamma_\texttt{f}], s^I_\texttt{f}) = 1.$$
If all checks pass define
$$\texttt{TX}^\gamma_\texttt{f} := \{([\texttt{TX}^\gamma_\texttt{f}], s^A_\texttt{f}, s^B_\texttt{f}, s^I_\texttt{f})\},$$
and set
$$\Gamma^A(\gamma.\mathsf{id}) := (\bot, \texttt{TX}^\gamma_\texttt{f}, tid_A)$$
and consider procedure successfully completed. Else record this execution as "failed" and stop.

$\boxed{\text{Case } I \text{ is honest and } A, B \text{ are corrupted}}$

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

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

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

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

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

$\mathtt{SetupVChannel}$ for channels With validity

$\boxed{\text{Case } A \text{ is honest and } B, I \text{ are corrupted}}$

1) Send $(\text{createInfo}, \gamma.\mathsf{id}, tid_A) \xhookrightarrow{\tau_0} B$.
2) In round $\tau_1 = \tau_0 + 1$, create the body of the funding transaction:

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

3) Upon A sending $(\mathtt{PRE\text{-}CREATE}, \gamma, \mathtt{TX}_{\mathtt{f}}, 1, t_{ofl}) \xhookrightarrow{\tau_1} \mathcal{F}_{preL}$ where $t_{ofl} = \gamma.\mathsf{val} + 3\Delta$, execute the simulator code for the Pre-Create procedure of the $\mathcal{F}_{preL}$ functionality. If A does not receive the message $(\mathtt{PRE\text{-}CREATED}, \gamma.id) \xleftarrow{\tau_2 \leq \tau_1 + T} \mathcal{F}_{preL}$ then mark this execution as "failed" and stop.
4) If A receives $(\text{createFund}, \gamma.\mathsf{id}, s_{\mathtt{f}}^{I}) \xleftarrow{\tau_2 + 2} I$, verify the signature, i.e. check:

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

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

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

Else record this execution as "failed" and stop.
5) Set

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

and consider procedure successfully completed.

$\boxed{\text{Case } B \text{ is honest and } A, I \text{ are corrupted}}$

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

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

Else record this execution as "failed" and stop.
2) Upon B sending $(\mathtt{PRE\text{-}CREATE}, \gamma, \mathtt{TX}_{\mathtt{f}}, 1, t_{ofl}) \xhookrightarrow{\tau_1 = \tau_0 + 1} \mathcal{F}_{preL}$ where $t_{ofl} = \gamma.\mathsf{val} + 3\Delta$, execute the simulator code for the Pre-Create procedure of the $\mathcal{F}_{preL}$ functionality. If B does not receive $(\mathtt{PRE\text{-}CREATED}, \gamma.id) \xleftarrow{\tau_2 \leq \tau_1 + T} \mathcal{F}_{preL}$ then mark this execution as "failed" and stop.
3) Compute a signature on the refund transaction, i.e., $s_{\mathtt{Ref}}^{B} \leftarrow \mathsf{Sign}_{sk_B}([\mathtt{TX}_{\mathtt{refund}}^{\gamma}])$ and define $\mathtt{TX}_{\mathtt{f}}^{\gamma,\mathtt{B}} := \{([\mathtt{TX}_{\mathtt{f}}^{\gamma}])\}$. Then, send $(\text{createFund}, \gamma.\mathsf{id}, s_{\mathtt{Ref}}^{B}, [\mathtt{TX}_{\mathtt{refund}}^{\gamma}], [\mathtt{TX}_{\mathtt{f}}^{\gamma}]) \xhookrightarrow{\tau_2} I$, set

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

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

$\boxed{\text{Case } I \text{ is honest and } A, B \text{ are corrupted}}$

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

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

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

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

Store $\Gamma^{I}(\gamma.\mathsf{id}) := (\perp, [\mathtt{TX}_{\mathtt{f}}^{\gamma}], \mathtt{TX}_{\mathtt{refund}}^{\gamma}, tid_A, tid_B)$. Then send $(\text{createFund}, \gamma.\mathsf{id}, s_{\mathtt{f}}^{I}) \xhookrightarrow{\tau_3} A$, 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

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

Simulator for updating virtual channels

Update virtual channels

$\boxed{\text{Case } A \text{ is honest and } B \text{ is corrupt}}$

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

1) Upon A sending $(\texttt{UPDATE}, id, \vec{\theta}, t_{\mathsf{stp}}) \xhookrightarrow{\tau_0^A} \mathcal{F}_{preL}$, 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 $(\texttt{SETUP–OK}, id) \xhookrightarrow{\tau_2^A \leq \tau_1^A + t_{\mathsf{stp}}} \mathcal{F}_{preL}$, continue executing the simulator code until step 4. If this execution is marked "failed" stop.
3) If A does not receive $(\texttt{PRE–UPDATE–OK}, id) \xhookleftarrow{\tau_3^A \leq \tau_2^A + T} \mathcal{F}_{preL}$ or $(\texttt{PRE–UPDATE–REJECT}, id) \xhookleftarrow{\tau_3^A \leq \tau_2^A + T} \mathcal{F}_{preL}$, execute the simulator code for the procedure $\texttt{Offload}^A(id)$ and stop.
4) Upon A sending $(\texttt{PRE–REVOKE}, id) \xhookrightarrow{\tau_3^A} \mathcal{F}_{preL}$ continue executing the simulator code until the end. If this execution is marked as "failed" execute the simulator code for the procedure $\texttt{Offload}^A(id)$ and stop.
5) Upon A receiving $(\texttt{PRE–UPDATED}, id) \xhookleftarrow{\tau_4^A \leq \tau_3^A + T} \mathcal{F}_{preL}$, 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 $\texttt{Offload}^A(id)$ and stop.

$\boxed{\text{Case } B \text{ is honest and } A \text{ is corrupt}}$

1) Let τ_0^B be the round in which B receives $(\texttt{PRE–UPDATE–REQ}, id, \vec{\theta}, t_{\mathsf{stp}}, tid) \xhookleftarrow{\tau_0^B} \mathcal{F}_{preL}$.
2) Let $\tau_1^B \leq \tau_0 + t_{\mathsf{stp}} + T$ be the round in which B receives the message $(\texttt{PRE–SETUP–OK}, id) \xhookleftarrow{\tau_1^B \leq \tau_0 + t_{\mathsf{stp}} + T} \mathcal{F}_{preL}$.
3) Upon B sending $(\texttt{UPDATE–OK}, id) \xhookleftarrow{\tau_1^B} \mathcal{F}_{preL}$ 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 $\texttt{Offload}^B(id)$ and stop.
4) Upon B sending $(\texttt{PRE–REVOKE}, id) \xhookrightarrow{\tau_2^B} \mathcal{F}_{preL}$ continue executing the simulator code until the end. If this execution is marked as "failed" execute the simulator code for the procedure $\texttt{Offload}^B(id)$ and stop.
5) Upon B receiving $(\texttt{PRE–UPDATED}, id) \xhookleftarrow{\tau_3^B \leq \tau_2^B + T} \mathcal{F}_{preL}$, 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

$\boxed{\text{Case } A \text{ honest and } I, B \text{ corrupted}}$

1) Extract γ and $\texttt{TX}_{\texttt{f}}^{\gamma}$ from $\Gamma^A(id)$ and tid_A, tid_B from $\texttt{TX}_{\texttt{f}}^{\gamma}$. Then define $id_\alpha := \gamma.\mathsf{subchan}(A)$. Upon A sending $(\texttt{CLOSE}, id_\alpha) \xhookrightarrow{\tau_0} \mathcal{F}_L$ execute the simulator code for the close procedure of generalized ledger channels.
2) If A receives $(\texttt{CLOSED}, id_\alpha) \xhookleftarrow{\tau_1 \leq \tau_0 + T + 3\Delta} \mathcal{F}_L$, 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 $(\text{post}, \texttt{TX}_{\texttt{f}}^{\gamma}) \xhookrightarrow{\tau_2} \widehat{\mathcal{L}}$.
   - Else in round T_2 create the punishment transaction $\texttt{TX}_{\mathsf{pun}}$ as $\texttt{TX}_{\mathsf{pun}}.\mathsf{Input} := tid_A$, $\texttt{TX}_{\mathsf{pun}}.\mathsf{Output} := (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \texttt{One–Sig}_{pk_A})$ and $\texttt{TX}_{\mathsf{pun}}.\mathsf{Witness} := \mathsf{Sign}_{sk_A}([\texttt{TX}_{\mathsf{pun}}])$. Then $(\text{post}, \texttt{TX}_{\mathsf{pun}}) \xhookrightarrow{T_2} \widehat{\mathcal{L}}$.
4) Let $T_3 := \tau_2 + \Delta$ and distinguish the following two cases:
   - The transaction $\texttt{TX}_{\texttt{f}}^{\gamma}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3 \leq T_3$, then update $\Gamma^A := \texttt{ToLedgerChannel}(\Gamma^A, \gamma.\mathsf{id})$ and set $m :=$ offloaded.
   - The transaction $\texttt{TX}_{\mathsf{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3 \leq T_3$, then define $\Gamma^A(\gamma.\mathsf{id}) = \perp$ and set $m :=$ punished.
5) Return m in round τ_3.

$\boxed{\text{Case } I \text{ honest and } A, B \text{ corrupted}}$

1) Extract γ and $\texttt{TX}_{\texttt{f}}^{\gamma}$ from $\Gamma^I(id)$ and tid_A, tid_B from $\texttt{TX}_{\texttt{f}}^{\gamma}$. Then define $id_\alpha := \gamma.\mathsf{subchan}(A)$, $id_\beta := \gamma.\mathsf{subchan}(B)$ Upon I sending the messages $(\texttt{CLOSE}, id_\alpha) \xhookrightarrow{\tau_0} \mathcal{F}_L$ and $(\texttt{CLOSE}, id_\beta) \xhookrightarrow{\tau_0} \mathcal{F}_L$ execute the simulator code of the close procedure for the generalized channels.
2) If I receives both $(m, id_\alpha) \xhookleftarrow{\tau_1^A \leq \tau_0 + T + 3\Delta} \mathcal{F}_L$ and $(\texttt{CLOSED}, id_\beta) \xhookleftarrow{\tau_1^B \leq \tau_0 + T + 3\Delta} \mathcal{F}_L$, check that a transaction with tid_A and a transaction with tid_B appeared on $\widehat{\mathcal{L}}$. Then publish $(\text{post}, \texttt{TX}_{\texttt{f}}^{\gamma}) \xhookrightarrow{\tau_1} \widehat{\mathcal{L}}$, where $\tau_1 := \max\{\tau_1^A, \tau_1^B\}$. Otherwise set $\Gamma^I(id) = \perp$ and stop.
3) Once $\texttt{TX}_{\texttt{f}}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round $\tau_2 \leq \tau_1 + \Delta$, then $\Gamma^I(\gamma.\mathsf{id}) = \perp$ and return "offloaded".

Offloading virtual channels with validity

$\boxed{\text{Case } A \text{ honest and } B, I \text{ corrupted}}$

1) Extract γ, tid_A and $\texttt{TX}_{\texttt{f}}^{\gamma}$ from $\Gamma^A(id)$. Then define $id_\alpha := \gamma.\mathsf{subchan}(A)$ and Upon A sending $(\texttt{CLOSE}, id_\alpha) \xhookrightarrow{\tau_0} \mathcal{F}_L$ execute the simulator code of the close procedure of the generalized ledger channel.
2) If A receives $(\texttt{CLOSED}, id_\alpha) \xhookleftarrow{\tau_1 \leq \tau_0 + T + 3\Delta} \mathcal{F}_L$, then post $(\text{post}, \texttt{TX}_{\texttt{f}}^{\gamma,\texttt{A}}) \xhookrightarrow{\tau_2} \widehat{\mathcal{L}}$. Otherwise, set $\Gamma^A(\gamma.\mathsf{id}) = \perp$ and stop.
3) Once $\texttt{TX}_{\texttt{f}}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round $\tau_2 \leq \tau_1 + \Delta$, then set $\Gamma_L^A(id) := \Gamma^A(id), \Gamma^A(id) := \perp$ and return "offloaded".

$\boxed{\text{Case } B \text{ honest and } A, I \text{ corrupted}}$

1) Extract γ, tid_B and $[\texttt{TX}_{\texttt{f}}^{\gamma}]$ from $\Gamma^B(id)$. Then define $id_\beta := \gamma.\mathsf{subchan}(B)$ and Upon B sending $(\texttt{CLOSE}, id_\beta) \xhookrightarrow{\tau_0} \mathcal{F}_L$ execute the simulator code of the close procedure of the generalized ledger channel.
2) If B receives $(\texttt{CLOSED}, id_\beta) \xhookleftarrow{\tau_1 \leq \tau_0 + T + 3\Delta} \mathcal{F}_L$, then continue. Otherwise, set $\Gamma^B(\gamma.\mathsf{id}) = \perp$ and stop.
3) Create the punishment transaction $\texttt{TX}_{\mathsf{pun}}$ as $\texttt{TX}_{\mathsf{pun}}.\mathsf{Input} := tid_B$, $\texttt{TX}_{\mathsf{pun}}.\mathsf{Output} := (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \texttt{One–Sig}_{pk_B})$ and $\texttt{TX}_{\mathsf{pun}}.\mathsf{Witness} := \mathsf{Sign}_{sk_B}([\texttt{TX}_{\mathsf{pun}}])$. Then wait until

round $\tau_2 := \max\{\tau_1, \gamma.\mathsf{val} + 2\Delta\}$ and send $(\text{post}, \mathtt{TX}_{\text{pun}}) \xhookrightarrow{\tau_2} \widehat{\mathcal{L}}$.

4) Let $T_3 := \tau_2 + \Delta$ and distinguish the following two cases:
   - A transaction with identifier $\mathtt{TX}_{\mathtt{f}}^{\gamma}.\mathsf{txid}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3 \leq T_3$, then update $\Gamma_L^B(id) := \Gamma^B(id)$ and set $m :=$ offloaded.
   - The transaction $\mathtt{TX}_{\text{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3 \leq T_3$, then define $\Gamma^B(\gamma.\mathsf{id}) = \perp$, and set $m :=$ punished.
5) Return m in round τ_3.

$\boxed{\text{Case } I \text{ honest and } A, B \text{ corrupted}}$

1) Extract γ, tid_A, tid_B, $\mathtt{TX}_{\mathtt{refund}}^{\gamma}$ and $[\mathtt{TX}_{\mathtt{f}}^{\gamma}]$ from $\Gamma^I(id)$. Then define $id_\alpha := \gamma.\mathsf{subchan}(A)$, $id_\beta := \gamma.\mathsf{subchan}(B)$ and upon I sending $(\mathtt{CLOSE}, id_\alpha) \xhookrightarrow{\tau_0} \mathcal{F}_L$ and $(\mathtt{CLOSE}, id_\beta) \xhookrightarrow{\tau_0} \mathcal{F}_L$ 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 $(\mathtt{CLOSED}, id_\alpha) \xleftarrow{\tau_1^A \leq \tau_0 + T + 3\Delta} \mathcal{F}_L$ and $(\mathtt{CLOSED}, id_\beta) \xleftarrow{\tau_1^B \leq \tau_0 + T + 3\Delta} \mathcal{F}_L$,then continue. Otherwise, set $\Gamma^I(\gamma.\mathsf{id}) = \perp$ and stop.
3) Create the punishment transaction $\mathtt{TX}_{\text{pun}}$ as $\mathtt{TX}_{\text{pun}}.\mathsf{Input} := tid_A$, $\mathtt{TX}_{\text{pun}}.\mathsf{Output} := (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \mathtt{One\text{-}Sig}_{pk_I})$ and $\mathtt{TX}_{\text{pun}}.\mathsf{Witness} := \mathsf{Sign}_{sk_I}([\mathtt{TX}_{\text{pun}}])$. Then wait until round $\tau_2 := \max\{\tau_1^A, \gamma.\mathsf{val}\}$ and send $(\text{post}, \mathtt{TX}_{\text{pun}}) \xhookrightarrow{\tau_2} \widehat{\mathcal{L}}$.
4) Let $T_3 := \tau_2 + \Delta$ and distinguish the following two cases:
   - A transaction with identifier $\mathtt{TX}_{\mathtt{f}}^{\gamma}.\mathsf{txid}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3' \leq T_3$, send $(\text{post}, \mathtt{TX}_{\mathtt{refund}}^{\gamma}) \xhookrightarrow{\tau_4} \widehat{\mathcal{L}}$ where $\tau_4 := \max\{\tau_1^B, \tau_3'\}$. Once $\mathtt{TX}_{\mathtt{refund}}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round $\tau_5 \leq \tau_4 + \Delta$, set $\Gamma^I(\gamma.\mathsf{id}) = \perp$ and $m :=$ offloaded.
   - The transaction $\mathtt{TX}_{\text{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3'' \leq T_3$, then set $\Gamma^I(\gamma.\mathsf{id}) = \perp$ 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 $(\mathtt{PUNISH}) \xhookrightarrow{\tau_0} \mathcal{F}_L$, execute the simulator code for the punish procedure of the generalized channels.
- Execute the simulator code for both `Punish` and `Punish–Validity`.

<u>`Punish`</u>

$\boxed{\text{Case } A \text{ honest and } I, B \text{ corrupted}}$

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

1) Extract $\mathtt{TX}_{\mathtt{f}}^{\gamma}$ from $\Gamma^A(id)$ and tid_A, tid_B from $\mathtt{TX}_{\mathtt{f}}^{\gamma}$. 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 $\tau_2 \leq T_2$ the transaction with tid_B appeared on $\widehat{\mathcal{L}}$, then $(\text{post}, \mathtt{TX}_{\mathtt{f}}^{\gamma}) \xhookrightarrow{\tau_2} \widehat{\mathcal{L}}$.
   - Else in round T_2 create the punishment transaction $\mathtt{TX}_{\text{pun}}$ as $\mathtt{TX}_{\text{pun}}.\mathsf{Input} := tid_A$, $\mathtt{TX}_{\text{pun}}.\mathsf{Output} := (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \mathtt{One\text{-}Sig}_{pk_A})$ and $\mathtt{TX}_{\text{pun}}.\mathsf{Witness} := \mathsf{Sign}_{sk_A}([\mathtt{TX}_{\text{pun}}])$ and $(\text{post}, \mathtt{TX}_{\text{pun}}) \xhookrightarrow{T_2} \widehat{\mathcal{L}}$.
3) Let $T_3 := \tau_2 + \Delta$ and distinguish the following two cases:
   - The transaction $\mathtt{TX}_{\mathtt{f}}^{\gamma}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3 \leq T_3$, then update $\Gamma^A := \mathtt{ToLedgerChannel}(\Gamma^A, \gamma.\mathsf{id})$.
   - The transaction $\mathtt{TX}_{\text{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3 \leq T_3$, then define $\Gamma^A(\gamma.\mathsf{id}) = \perp$.

$\boxed{\text{Case } I \text{ honest and } A, B \text{ corrupted}}$

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

1) Extract $\mathtt{TX}_{\mathtt{f}}^{\gamma}$ from $\Gamma^I(id)$ and tid_A, tid_B from $\mathtt{TX}_{\mathtt{f}}^{\gamma}$. 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 $id_\alpha := \gamma.\mathsf{subchan}(Q)$ where $Q = \gamma.\mathsf{otherParty}(P)$ and upon I sending $(\mathtt{CLOSE}, id_\alpha) \xhookrightarrow{\tau_0} \mathcal{F}_L$ execute simulator the code for the punish procedure of the generalized virtual channels.
3) If I receives $(\mathtt{CLOSED}, id_\alpha) \xleftarrow{\tau_1 \leq \tau_0 + T + 3\Delta} \mathcal{F}_L$ and tid_Q appeared on $\widehat{\mathcal{L}}$, $(\text{post}, \mathtt{TX}_{\mathtt{f}}^{\gamma}) \xhookrightarrow{\tau_1} \widehat{\mathcal{L}}$. Otherwise set $\Gamma^I(id) = \perp$ and stop.
4) Once $\mathtt{TX}_{\mathtt{f}}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round τ_2, such that $\tau_2 \leq \tau_1 + \Delta$, set $\Gamma^I(id) = \perp$.

<u>`Punish–Validity`</u>

$\boxed{\text{Case } A \text{ honest and } I, B \text{ corrupted}}$

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

1) Extract tid_A and $\mathtt{TX}_{\mathtt{f}}^{\gamma}$ from $\Gamma^A(id)$. Check if tid_A appeared on $\widehat{\mathcal{L}}$. If not, then stop.
2) Send $(\text{post}, \mathtt{TX}_{\mathtt{f}}^{\gamma}) \xhookrightarrow{\tau_1} \widehat{\mathcal{L}}$.
3) Once $\mathtt{TX}_{\mathtt{f}}^{\gamma}$ is accepted by $\widehat{\mathcal{L}}$ in round $\tau_2 \leq \tau_1 + \Delta$, set $\Gamma_L^A(id) := \Gamma^A(id)$, $\Gamma^A(id) := \perp$.

$\boxed{\text{Case } B \text{ honest and } A, I \text{ corrupted}}$

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

1) Extract tid_B and $[\mathtt{TX}_{\mathtt{f}}^{\gamma}]$ from $\Gamma^B(id)$. Check if tid_B or $[\mathtt{TX}_{\mathtt{f}}^{\gamma}].\mathsf{txid}$ appeared on $\widehat{\mathcal{L}}$. If not, then stop.
2) If $\mathtt{TX}_{\mathtt{f}}^{\gamma}$ appeared on $\widehat{\mathcal{L}}$, set $\Gamma_L^B(id) := \Gamma^B(id)$, $\Gamma^B(id) := \perp$ and stop.
3) If tid_B appeared on $\widehat{\mathcal{L}}$, create the punishment transaction $\mathtt{TX}_{\text{pun}}$ as $\mathtt{TX}_{\text{pun}}.\mathsf{Input} := tid_B$, $\mathtt{TX}_{\text{pun}}.\mathsf{Output} := (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \mathtt{One\text{-}Sig}_{pk_B})$ and $\mathtt{TX}_{\text{pun}}.\mathsf{Witness} := \mathsf{Sign}_{sk_B}([\mathtt{TX}_{\text{pun}}])$. Then wait until round $\tau_2 := \max\{\tau_1, \gamma.\mathsf{val} + 2\Delta\}$ and send $(\text{post}, \mathtt{TX}_{\text{pun}}) \xhookrightarrow{\tau_2} \widehat{\mathcal{L}}$.
4) If transaction $\mathtt{TX}_{\text{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_3 \leq \tau_2 + \Delta$, then define $\Gamma^B(\gamma.\mathsf{id}) = \perp$.

$\boxed{\text{Case } I \text{ honest and } A, B \text{ corrupted}}$

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

1) Extract tid_A, tid_B, $\mathtt{TX}_{\mathtt{refund}}^{\gamma}$ and $[\mathtt{TX}_{\mathtt{f}}^{\gamma}]$ from $\Gamma^I(id)$. Check if tid_A or tid_B appeared on $\widehat{\mathcal{L}}$ or $\tau_1 = \gamma.\mathsf{val} - (3\Delta + T)$. If not, then stop.
2) Distinguish the following cases:
   - If $\tau_1 = \gamma.\mathsf{val} - (3\Delta + T)$, define $id_\alpha := \gamma.\mathsf{subchan}(A)$, $id_\beta := \gamma.\mathsf{subchan}(B)$ and upon I sending the messages $(\mathtt{CLOSE}, id_\alpha) \xhookrightarrow{\tau_1} \mathcal{F}_L$ and $(\mathtt{CLOSE}, id_\beta) \xhookrightarrow{\tau_1} \mathcal{F}_L$

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 $(\texttt{CLOSE}, id_\alpha) \overset{\tau_1}{\hookrightarrow} \mathcal{F}_L$.

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 $\texttt{TX}_{\text{pun}}$ as $\texttt{TX}_{\text{pun}}.\mathsf{Input} := tid_A$, $\texttt{TX}_{\text{pun}}.\mathsf{Output} := (\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \texttt{One–Sig}_{pk_I})$ and $\texttt{TX}_{\text{pun}}.\mathsf{Witness} := \mathsf{Sign}_{sk_I}([\texttt{TX}_{\text{pun}}])$. Then wait until round $\tau_3 := \max\{\tau_2, \gamma.\mathsf{val}\}$ and send $(\text{post}, \texttt{TX}_{\text{pun}}) \overset{\tau_3}{\hookrightarrow} \widehat{\mathcal{L}}$.
4) Distinguish the following two cases:
   - The transaction $\texttt{TX}^\gamma_{\texttt{f}}.\mathsf{txid}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_4 \leq \tau_3 + \Delta$, send $(\text{post}, \texttt{TX}^\gamma_{\texttt{refund}}) \overset{\tau_5}{\hookrightarrow} \widehat{\mathcal{L}}$ where $\tau_5 := \max\{\gamma.\mathsf{val} + \Delta, \tau_4\}$. Once $\texttt{TX}^\gamma_{\texttt{refund}}$ is accepted by $\widehat{\mathcal{L}}$ in round $\tau_6 \leq \tau_5 + \Delta$, set $\Gamma^I(\gamma.\mathsf{id}) = \perp$ and stop.
   - The transaction $\texttt{TX}_{\text{pun}}$ was accepted by $\widehat{\mathcal{L}}$ in $\tau_4 \leq \tau_3 + \Delta$, then set $\Gamma^I(\gamma.\mathsf{id}) = \perp$.

Simulator for Close in a virtual channel

Closing virtual channels

Case A honest and I, B corrupted

1) Upon A sending $(\texttt{CLOSE}, id) \overset{\tau_0^A}{\hookrightarrow} \mathcal{F}_V$ or in round $\tau_0^A = \gamma.\mathsf{val} - (4\Delta + 7T)$ if $\gamma.\mathsf{val} \neq \perp$, proceed as follows:
2) Extract γ, $\texttt{TX}^\gamma_{\texttt{f}}$, tid_A from $\Gamma^A(id)$. Parse
$$\texttt{TX}^\gamma_{\texttt{s}}.\mathsf{Output} = \Big((c_A, \texttt{One–Sig}_{pk_A}), (c_B, \texttt{One–Sig}_{pk_B})\Big).$$
3) Compute the new state of the channel $id_\alpha := \gamma.\mathsf{subchan}(A)$ as
$$\vec{\theta}_A := \{(c_A, \texttt{One–Sig}_{pk_A}), (\gamma.\mathsf{cash} - c_A + \frac{\gamma.\mathsf{fee}}{2}, \texttt{One–Sig}_{pk_I})\}$$
Then, upon A sending $(\texttt{UPDATE}, id_\alpha, \vec{\theta}_A, 0) \overset{\tau_0^A}{\hookrightarrow} \mathcal{F}_L$ execute the simulator code for the update procedure of generalized channels until $(\texttt{SETUP}, id_\alpha, \vec{tid'}_A)$ is sent by $\mathcal{F}_L$. If this execution fails instruct $\mathcal{F}_V$ to execute $\texttt{V–ForceClose}(id)$ and stop.
4) Upon A sending $(\texttt{SETUP–OK}, id_\alpha) \xhookrightarrow{\tau_1^A \leq \tau_0^A + T} \mathcal{F}_L$ continue executing the simulator code for the update procedure of generalized channels until A receives $(\texttt{UPDATE–OK}, id_\alpha) \xhookleftarrow{\tau_2^A \leq \tau_1^A + 2T} \mathcal{F}_L$. If this execution fails instruct $\mathcal{F}_V$ to execute $\texttt{V–ForceClose}(id)$ and stop.
5) Upon A sending $(\texttt{REVOKE}, id_\alpha) \overset{\tau_2^A}{\hookrightarrow} \mathcal{F}_L$ continue executing the simulator code for the update procedure of generalized channels until A receives $(\texttt{UPDATED}, id_\alpha) \xhookleftarrow{\tau_3^A \leq \tau_2^A + 2T} \mathcal{F}_L$, then set $\Gamma^A(id) := \perp$ and stop. If this execution fails instruct $\mathcal{F}_V$ to execute $\texttt{V–ForceClose}(id)$ and stop.

Case I honest and A, B corrupted

1) Upon I sending $(\texttt{CLOSE}, id) \overset{\tau_0^I}{\hookrightarrow} \mathcal{F}_V$ or in round $\tau_0^A = \gamma.\mathsf{val} - (4\Delta + 7T)$ if $\gamma.\mathsf{val} \neq \perp$, proceed as follows:
2) Extract γ, $\texttt{TX}^\gamma_{\texttt{f}}$, tid_A and tid_B from $\Gamma^I(id)$.
3) Let $id_\alpha = \gamma.\mathsf{subchan}(A)$, $id_\beta = \gamma.\mathsf{subchan}(B)$ and $c := \gamma.\mathsf{cash}$, execute the simulator code for the update procedure of generalized channels until $(\texttt{UPDATE–REQ}, id_\alpha, \vec{tid'}_A, \vec{\theta}_A, 0) \hookleftarrow \mathcal{F}_L$ and $(\texttt{UPDATE–REQ}, id_\beta, \vec{tid'}_B, \vec{\theta}_B, 0) \hookleftarrow \mathcal{F}_L$ 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
$$\vec{\theta}_A = \{(c_A, \texttt{One–Sig}_{pk_A}), (c - c_A + \gamma.\mathsf{fee}/2, \texttt{One–Sig}_{pk_I})\}$$
$$\vec{\theta}_B = \{(c_B, \texttt{One–Sig}_{pk_B}), (c - c_B + \gamma.\mathsf{fee}/2, \texttt{One–Sig}_{pk_I})\}$$
If not, then stop.
5) continue executing the simulator code for the update procedure of generalized channels until I receives $(\texttt{SETUP–OK}, id_\alpha) \hookleftarrow \mathcal{F}_L$ and $(\texttt{SETUP–OK}, id_\beta)) \hookleftarrow \mathcal{F}_L$ until round $\tau_2^I \leq \tau_1^I + T$. Otherwise stop.
6) Upon I sending $(\texttt{UPDATE–OK}, id_\beta) \overset{\tau_2^I}{\hookrightarrow} \mathcal{F}_L$ continue executing the simulator code for the update procedure of generalized channels until I receives the messages $(\texttt{REVOKE–REQ}, id_\alpha) \hookleftarrow \mathcal{F}_L$ and $(\texttt{REVOKE–REQ}, id_\beta) \hookleftarrow \mathcal{F}_L$ until round $\tau_3^I \leq \tau_2^I + 2T$.
7) Upon I sending $(\texttt{REVOKE}, id_\alpha) \overset{\tau_3^I}{\hookrightarrow} \mathcal{F}_L$ and $(\texttt{REVOKE}, id_\beta) \overset{\tau_3^I}{\hookrightarrow} \mathcal{F}_L$ executing the simulator code for the update procedure of generalized channels until the end and set $\Gamma^I(id) := \perp$.