Bitcoin-Compatible Virtual Channels

Lukas Aumayr, Matteo Maffei
Department of Computer Science
TU Wien
Wien, Austria
firstname.lastname@tuwien.ac.at

Oğuzhan Ersoy
Department of Intelligent Systems
TU Delft
Delft, Netherlands
o.ersoy@tudelft.nl

Andreas Erwig, Sebastian Faust, Siavash Riahi
Department of Computer Science
TU Darmstadt
Darmstadt, Germany
firstname.lastname@tu-darmstadt.de

Kristina Hostáková
Department of Computer Science
ETH Zürich
Zürich, Switzerland
kristina.hostakova@inf.ethz.ch

Pedro Moreno-Sanchez
IMDEA Software Institute
Madrid, Spain
pedro.moreno@imdea.org

2021 IEEE Symposium on Security and Privacy (SP) | 978-1-7281-8934-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/SP40001.2021.00097

***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 [22] 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 [15, 29, 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 [24], 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) [24, 19]. 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

DOI 10.1109/SP40001.2021.00097

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 [23, 27, 18].

An alternative approach for connecting multiple payment channels was introduced by Dziembowski et al. [12]. 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 [12]. Since establishing a payment channel requires a deposit and active monitoring, the number of channels a user can establish is limited. With payment hubs [12], 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 [12] 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 [12]). 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 [12].
- We evaluate our protocol over two different PCN constructions, the Lightning Network (LN) [24] and Generalized channels (GC) [3], 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, 19, 20, 12] 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 [3], 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 $T.\mathsf{attr}$.

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 $\mathsf{tx.Input}$ is a vector of strings which identify the inputs of tx. Similarly, the outputs of the transaction $\mathsf{tx.Output}$ 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 $\mathsf{tx.TimeLock}$. 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.

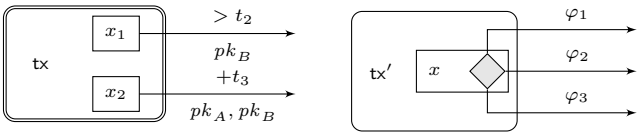

Fig. 1: (Left) Transaction tx is published on the blockchain. The output of value x_1 can be spent by a transaction signed w.r.t. pk_B after round t_2, and the output of value x_2 can be spent by a transaction signed w.r.t. pk_A and pk_B but only if at least t_3 rounds passed since tx was accepted by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

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.

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 [15, 3] 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 [24] defines the state-of-the-art payment channel construction for Bitcoin.

C. Generalized channels

The recent work of Aumayr et al. [3] 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

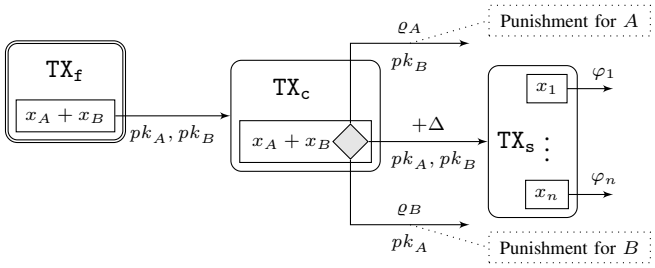

Fig. 2: A generalized channel in the state $((x_1, \varphi_1), \ldots, (x_n, \varphi_n))$. The value of Δ upper bounds the time needed to publish a transaction on a blockchain. The condition ϱ_A represents the verification of A' revocation secret and ϱ_B represents the verification of B' revocation secret.

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 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

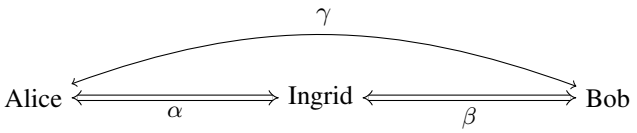

Fig. 3: A virtual channel γ built over ledger channels α, β.

(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; (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* [12]. 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 [12, 13, 11]. 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 [3] 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 [3] 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 [3] 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 $\gamma.\mathsf{Ingrid}$ is honest, she never loses coins, even if $\gamma.\mathsf{Alice}$ and $\gamma.\mathsf{Bob}$ collude.
(V2) Offload with punish: If γ is a virtual channel *without* validity (VC-NV), then $\gamma.\mathsf{Ingrid}$ can transform γ to a ledger channel. Party $P \in \gamma.\mathsf{endUsers}$ 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 $\gamma.\mathsf{Alice}$ can transform γ to a ledger channel. If γ is not transformed into a ledger channel or closed before time $\gamma.\mathsf{val}$, $\gamma.\mathsf{Ingrid}$ and $\gamma.\mathsf{Bob}$ can get financially compensated.

We first note that the instant finality with punish property (S3) does not provide any guarantees for $\mathsf{Ingrid} \notin \gamma.\mathsf{endUsers}$, 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 $\gamma.\mathsf{val}$, until when $\gamma.\mathsf{endUsers}$ have to close the virtual channel or transform it into a ledger channel (Ingrid's collateral is freed in both cases).

	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).

If $\gamma.\mathsf{endUsers}$ fail to do so, Ingrid can get her collateral back through a punishment mechanism. Hence, $\gamma.\mathsf{endUsers}$ 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.
(E2) Optimistic update: For a channel γ, this property guarantees that in the optimistic case when both parties in $\gamma.\mathsf{endUsers}$ are honest, a channel update takes a constant number of rounds.
(E3) Optimistic closure: In the optimistic case when all parties in $\gamma.\mathsf{users}$ 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 A.

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 [3] 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 instantiation with our construction with validity, we refer the reader to Appendix B. We present the formal pseudocode for the modular protocol in Appendix C.

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).

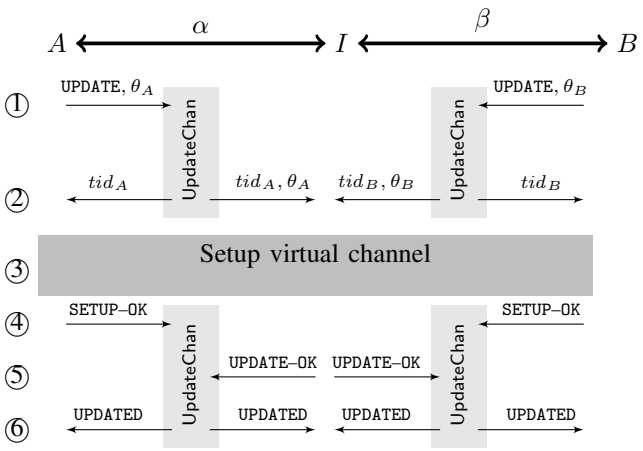

Fig. 4: Modular creation procedure of a virtual channel on top of two ledger channels α and β.

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 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.

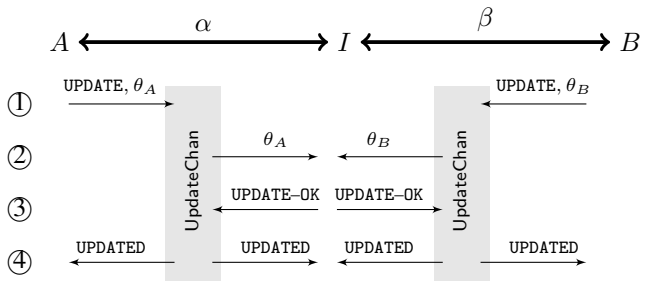

Fig. 5: Modular close procedure of a virtual channel on top of two ledger channels α and β. For $P \in \{A, B\}$, $\vec{\theta}_P := \{(c_P, \texttt{One–Sig}_{pk_P}), (c_Q + \frac{\gamma.\texttt{fee}}{2}, \texttt{One–Sig}_{pk_I})\}$ where $\gamma.\texttt{st} = \Big((c_P, \texttt{One–Sig}_{pk_P}), (c_Q,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

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 $\gamma.\texttt{st}$. To this end, each party $P \in \{A, B\}$ computes the new state for the ledger channel $\vec{\theta}_P := \{(c_P, \texttt{One–Sig}_{pk_P}), (\gamma.\texttt{cash} - c_P, \texttt{One–Sig}_{pk_I})\}$ 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 B 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 $\texttt{TX}_\texttt{s}^\texttt{A}$) and the ledger channel β (represented by $\texttt{TX}_\texttt{s}^\texttt{B}$). 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

[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.

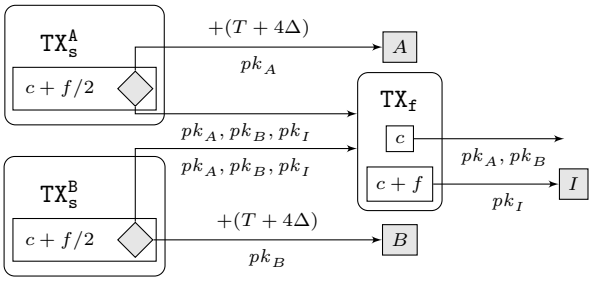

Fig. 6: Funding of a virtual channel γ without validity. T upper bounds the number of off-chain communication rounds between two parties for any operation in the ledger channel.

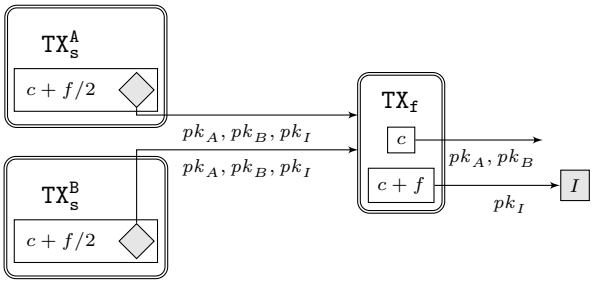

Fig. 7: Transactions published after a successful offload.

has to proactively monitor the virtual channel as she has an incentive to publish $\mathtt{TX}_\mathtt{f}$ 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 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., $\mathtt{TX}_\mathtt{f}$. 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

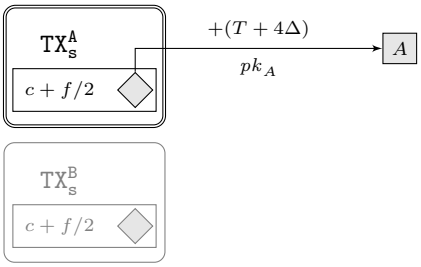

Fig. 8: Transactions published after A successfully executed the punishment procedure. The grayed transaction $\mathtt{TX}_\mathtt{s}^\mathtt{B}$ indicates that this transaction has not been published.

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 $\mathtt{TX}_\mathtt{f}$.

Let us now get into more details. Assume that A's ledger channel with I is closed, but the funding transaction $\mathtt{TX}_\mathtt{f}$ 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 $\mathtt{TX}_\mathtt{f}$ 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 $\mathtt{TX}_\mathtt{f}$. If I fails to do so in the prescribed time period, A receives all coins in her channel with I.

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 $\mathtt{TX}_\mathtt{f}$. 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 B.

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 [3]. 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 [24]. 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.

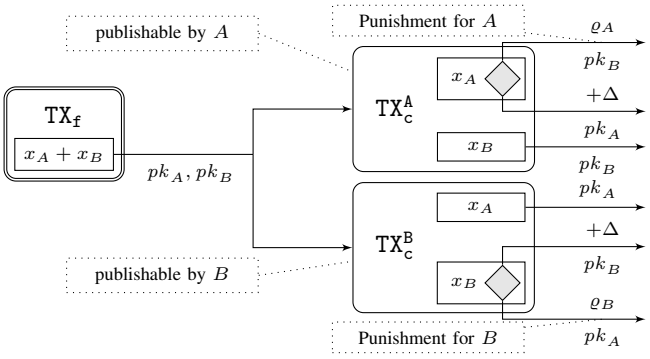

Fig. 9: A Lightning style payment channel where A has x_A coins and B has x_B coins. Δ upper bounds the time needed to publish a transaction on a blockchain. condition ϱ_A represents the verification of A' revocation secret and h represents the verification of B' revocation secret.

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 B.

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 [3]. 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 readers to the full version of the paper [2] 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 [3] when designing our virtual channel protocol.

Due to lack of space, we only mention the main security theorem and provide a high-level proof sketch here. We refer the reader to the full version of this paper [2] 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 $\mathcal{F}_V(2)$.

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

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.

four $\texttt{TX}_\texttt{f}$ of 640 bytes each (VC-NV) or two $\texttt{TX}_\texttt{f}$ of 309 and four $\texttt{TX}_{\texttt{refund}}$ of 377 bytes (VC-V). For every $\texttt{TX}_\texttt{f}$, 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 with one $\texttt{TX}_\texttt{f}$ (for VC-NV) and $\texttt{TX}_\texttt{f}$ plus $\texttt{TX}_{\texttt{refund}}$ (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 [24]. 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).

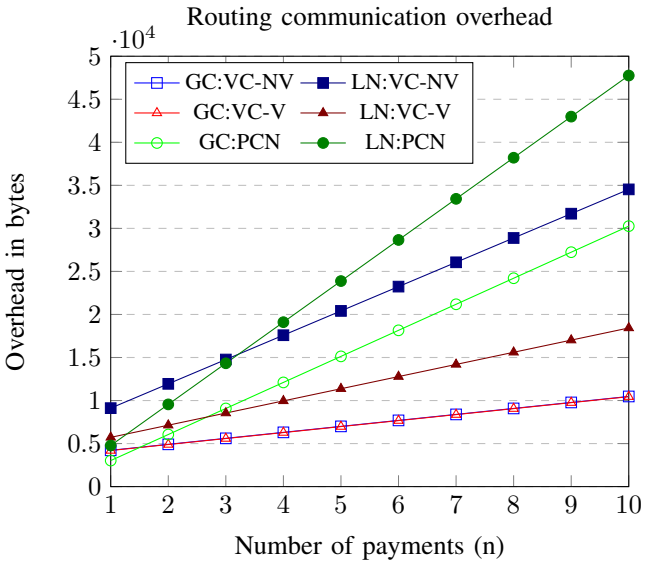

Fig. 10: Pictorial illustration of Table III.

	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.

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

In a virtual channel setting, γ.Ingrid can charge a base fee to collaborate to open and close the virtual channel, and also a relative fee to lock collateral coins in the virtual channel. However, no fees per payment are charged by Ingrid as she does not participate in them (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 $\mathsf{tx.cash}$, 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 [24], 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. [3] 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 [17] 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 [17]. 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 [24] for Bitcoin and Raiden Network [28] for Ethereum and has been widely studied in academia to research into different aspects such as privacy [19, 20], routing of payments [25], collateral management [14] and others. Similar to PCN, different constructions for PCH exist [26, 16, 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 [11, 13, 21, 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. [12] 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 [12]

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 [12] 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 [12], as we do not aim to improve the construction of [12] 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. *Bitcoin-Compatible Virtual Channels*. Cryptology ePrint Archive, Report 2020/554. https://eprint.iacr.org/2020/554. 2020.

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

[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, Y. Dodis, R. Pass, and S. Walfish. "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, L. Eckey, S. Faust, J. Hesse, and K. Hostáková. "Multi-party Virtual State Channels". In: *EUROCRYPT 2019, Part I*. 2019, pp. 625–656.

[12] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski. "Perun: Virtual Payment Hubs over Cryptocurrencies". In: *IEEE SP*. 2019, pp. 106–123.

[13] S. Dziembowski, S. Faust, and K. Hostakova. "General State Channel Networks". In: *ACM CCS*. 2018, pp. 949–966.

[14] C. Egger, P. Moreno-Sanchez, and M. Maffei. "Atomic Multi-Channel Updates with Constant Collateral in Bitcoin-Compatible Payment-Channel Networks". In: *ACM CCS*. 2019, pp. 801–815.

[15] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais. "SoK: Layer-Two Blockchain Protocols". In: *FC 2020*. 2020, pp. 201–226.

[16] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg. "TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub". In: *NDSS 2017*. The Internet Society, 2017.

[17] M. Jourenko, M. Larangeira, and K. Tanaka. "Lightweight Virtual Payment Channels". In: *CANS 2020*. 2020, pp. 365–384.

[18] 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.

[19] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi. "Concurrency and Privacy with Payment-Channel Networks". In: *ACM CCS 17*. Ed. by B. M.

Thuraisingham, D. Evans, T. Malkin, and D. Xu. ACM Press, 2017, pp. 455–471.
[20] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei. "Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability". In: *NDSS*. 2019.
[21] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry. "Sprites and State Channels: Payment Networks that Go Faster Than Lightning". In: *FC 2019*. 2019, pp. 508–526.
[22] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. http://bitcoin.org/bitcoin.pdf. 2009.
[23] U. Nisslmueller, K. Foerster, S. Schmid, and C. Decker. "Toward Active and Passive Confidentiality Attacks on Cryptocurrency Off-chain Networks". In: *ICISSP*. Ed. by S. Furnell, P. Mori, E. R. Weippl, and O. Camp. 2020, pp. 7–14.
[24] 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.
[25] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg. "Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions". In: *NDSS*. 2018.
[26] E. Tairi, P. Moreno-Sanchez, and M. Maffei. *A^2L: Anonymous Atomic Locks for Scalability and Interoperability in Payment Channel Hubs*. In press.
[27] S. Tikhomirov, P. Moreno-Sanchez, and M. Maffei. "A Quantitative Analysis of Security, Anonymity and Scalability for the Lightning Network". In: *IEEE EuroS&P*. 2020, pp. 387–396.
[28] *Update from the Raiden team on development progress, announcement of raidEX*. https://tinyurl.com/z2snp9e. Feb. 2017.
[29] A. Zamyatin et al. *SoK: Communication Across Distributed Ledgers*. In press.

APPENDIX

A. Ideal functionality for virtual channels

Here we define the ideal functionality $\mathcal{F}_V$ that describes the ideal behavior of both ledger and virtual channels. The full description of the ideal functionalities can be found in the full version of this paper [2].

$\mathcal{F}_V$ can be viewed as an extension of the ledger channel functionality $\mathcal{F}_L$ defined in [3]. 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 $\widehat{\mathcal{L}}$. 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 $\gamma.\mathsf{Alice}$, $\gamma.\mathsf{Bob}$ and $\gamma.\mathsf{Ingrid}$, 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 $\gamma.\mathsf{users}$, 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 $\gamma.\mathsf{Ingrid}$ 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 $\gamma.\mathsf{Ingrid}$ behaves maliciously and can publish both the old and new states, while $\gamma.\mathsf{Alice}$ or $\gamma.\mathsf{Bob}$ 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 $\gamma.\mathsf{endUsers}$ before round $\gamma.\mathsf{val}$, 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 $\gamma.\mathsf{subchan}$ has been spent until round $T_1+\Delta$. If not, the functionality outputs a message (`ERROR`). As in to [3], 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 $\gamma.\mathsf{users}$ 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 $\gamma.\mathsf{Alice}$ or $\gamma.\mathsf{Bob}$ which would cause the virtual channel to be force-

fully 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) $\mathsf{tx} \notin R$.

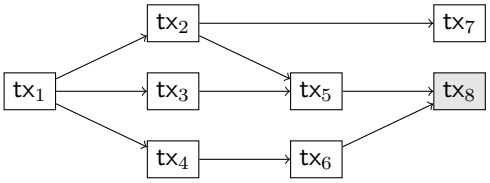

Fig. 11: The root sets of transaction tx_8 are $\{\mathsf{tx}_1\}$, $\{\mathsf{tx}_2, \mathsf{tx}_3, \mathsf{tx}_4\}$, $\{\mathsf{tx}_5, \mathsf{tx}_6\}$, $\{\mathsf{tx}_4, \mathsf{tx}_5\}$ and $\{\mathsf{tx}_2, \mathsf{tx}_3, \mathsf{tx}_6\}$.

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 $\mathcal{F}_V(T)$

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 $(\texttt{CREATE}, \gamma) \overset{\tau}{\hookleftarrow} P$, 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 $(\texttt{CREATE}, \gamma) \overset{\tau_1}{\hookleftarrow} Q_1$ and $(\texttt{CREATE}, \gamma) \overset{\tau_2}{\hookleftarrow} Q_2$, 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 $\texttt{UpdateState}(id_A, \vec{\theta_A})$, $\texttt{UpdateState}(id_B, \vec{\theta_B})$, set $\Gamma(\gamma.\mathsf{id}) := \gamma$, send $(\texttt{CREATED}, \gamma) \overset{\tau_3}{\hookrightarrow} \gamma.\mathsf{endUsers}$, stop.

Wait for others: Else wait for at most T_1 rounds to receive $(\texttt{CREATE}, \gamma) \overset{\tau_1 \leq \tau + T_1}{\longleftarrow} Q_1$ and $(\texttt{CREATE}, \gamma) \overset{\tau_2 \leq \tau + T_1}{\longleftarrow} Q_2$ 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 γ.Ingrid is honest or if instructed by $\mathcal{S}$, execute $\texttt{L–ForceClose}(id_i)$ 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 $\texttt{L–ForceClose}(id_i)$ and stop.

Update

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

Offload

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

Close

Channels without validity:

Upon $(\texttt{CLOSE}, id) \overset{\tau}{\hookleftarrow} P$, where $\gamma(id).\mathsf{val} = \bot$, 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 $(\texttt{CLOSE}, id) \overset{\tau_1}{\hookleftarrow} Q_1$ and $(\texttt{CLOSE}, id) \overset{\tau_2}{\hookleftarrow} Q_2$, 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 $\gamma.\mathsf{st} = \{(c_A, \texttt{One–Sig}_A), (c_B, \texttt{One–Sig}_B)\}$ and set
$$\vec{\theta_A} := ((c_A, \texttt{One–Sig}_A), (c_B + \gamma.\mathsf{fee}/2, \texttt{One–Sig}_I)),$$
$$\vec{\theta_B} := ((c_A + \gamma.\mathsf{fee}/2, \texttt{One–Sig}_I), (c_B, \texttt{One–Sig}_B)),$$
3) Unlock both subchannels and execute $\texttt{UpdateState}(id_A, \vec{\theta_A})$ and $\texttt{UpdateState}(id_B, \vec{\theta_B})$. Set $\Gamma(id) := \bot$ and send $(\texttt{CLOSED}, \gamma) \overset{\tau_3}{\hookrightarrow} \gamma.\mathsf{endUsers}$.

Wait for others: Else wait for at most T_1 rounds to receive $(\texttt{CLOSE}, \gamma) \xleftarrow{\tau_1 \leq \tau + T_1} Q_1$ and $(\texttt{CLOSE}, \gamma) \xleftarrow{\tau_2 \leq \tau + T_1} Q_2$ 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 $\texttt{V–ForceClose}(id_i)$ 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 $\texttt{L–ForceClose}(id_i)$ and stop.

Channels with validity:
For every $\gamma \in \Gamma$ s.t. $\gamma.\mathsf{val} \neq \bot$, 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 $\texttt{V–ForceClose}(\gamma.\mathsf{id})$.

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 $\Gamma(id_A) = (\bot, \mathsf{tx}_A)$ or $\Gamma(id_B) = (\bot, \mathsf{tx}_B)$, i.e., one of the subchannels was just closed, then let $\mathcal{S}$ set $t_1 \leq T'$, where $T' := \tau_0 + T + 5\Delta$ if $\gamma.\mathsf{val} = \bot$ and $T' := \gamma.\mathsf{val} + 3\Delta$ if $\gamma.\mathsf{val} \neq \bot$, and distinguish the following cases:

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

- a transaction tx_1 rooted at $\{\mathsf{tx}_A, \mathsf{tx}_B\}$ with an output $(\gamma.\mathsf{cash} + \gamma.\mathsf{fee}, \texttt{One–Sig}_I)$. In this case $(\texttt{OFFLOADED}, id) \overset{\tau_1}{\hookrightarrow} I$, 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 $\gamma.\mathsf{val} = \bot$, and rooted at $\{\mathsf{tx}_A\}$, if $\gamma.\mathsf{val} \neq \bot$. Let τ_2 be the round when tx_2 appeared on $\widehat{\mathcal{L}}$. Then output $(\texttt{OFFLOADED}, id) \overset{\tau_2}{\hookrightarrow} \gamma.\mathsf{endUsers}$, set $\gamma' = \gamma$, $\gamma'.\mathsf{Ingrid} = \bot$, $\gamma'.\mathsf{subchan} = \bot$, $\gamma.\mathsf{val} = \bot$ 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 $(\gamma.\mathsf{cash} + \gamma.\mathsf{fee}/2, \texttt{One–Sig}_P)$ as output. In that case, output $(\texttt{PUNISHED}, id) \overset{\tau_1}{\hookrightarrow} P$. Set $\Gamma(id) = \bot$ in the first round when PUNISHED was sent to all honest parties.

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

$\texttt{V–ForceClose}(id)$: Let τ_0 be the current round and $\gamma := \Gamma(id)$. Execute subprocedure $\texttt{Offload}(id)$. Let $T' := \tau_0 + 2T + 8\Delta$ if $\gamma.\mathsf{val} = \bot$ and $T' := \gamma.\mathsf{val} + 3\Delta$ if $\gamma.\mathsf{val} \neq \bot$. If in round $\tau_1 \leq T'$ it holds that $\Gamma(id) = (\gamma, \mathsf{tx})$, execute subprocedure $\texttt{L–ForceClose}(id)$.
Subprocedure $\texttt{Offload}(id)$: 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 $(\texttt{ERROR}) \overset{\tau_0 + \Delta}{\hookrightarrow} \gamma.\mathsf{users}$.
Subprocedure $\texttt{UpdateState}(id, \vec{\theta})$: Let $(\alpha, \mathsf{tx}) := \Gamma(id)$. Set $\alpha.\mathsf{st} := \vec{\theta}$ and update $\Gamma(id) := (\alpha, \mathsf{tx})$.

B. Concrete Instantiation 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.

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.

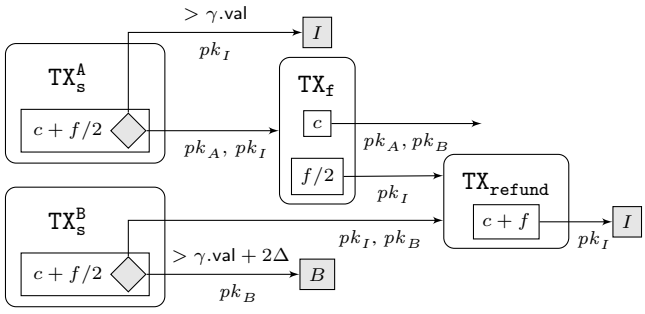

Fig. 12: Funding of a virtual channel γ with validity $\gamma.\mathsf{val}$.

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.

C. Protocol Pseudocode

In Figure 13, we present the pseudocode of our modular virtual channel protocol that was described at a high level in Section III-D.

Create virtual channel for $P \in \{A, B\}$

/ Initiate creation of γ with funding source tid_A, tid_B in round t_0

Let γ_P be the channel with id $\gamma.\mathsf{subchan}(P)$.

Compute $\theta_P := \mathtt{GenVChannelOutput}(\gamma_P, P)$

Assign $\mathsf{Setup} \leftarrow \mathtt{SetupVChannel}^P(\gamma, tid_A, tid_B)$

if $\mathsf{UpdateChan}^P(\gamma_P, \theta_P, \mathsf{Setup})$ returns UPDATE–OK Creation successful.

Close virtual channel for $P \in \{A, B\}$

/ Initiate closure of γ in round t_0^P

Let γ_P be the channel with id $\gamma.\mathsf{subchan}(P)$.

Parse $\gamma.\mathsf{st} = \Big((c_P, \mathtt{One–Sig}_{pk_P}), (c_Q, \mathtt{One–Sig}_{pk_Q})\Big)$

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

if $\mathsf{UpdateChan}^P(\gamma_P, \vec{\theta}_P, \perp)$ returns UPDATE–OK Close successful.

else Execute $\mathtt{Offload}^P(\gamma)$ and stop.

Update virtual channel for $P \in \{A, B\}$

/ Initiate update of γ with state $\vec{\theta}$ in t_0^P

if $\mathsf{UpdateChan}^P(\gamma, \vec{\theta}, \perp)$ returns UPDATE–OK Update successful.

else Execute $\mathtt{Offload}^P(\gamma)$ and stop.

$\mathtt{SetupVChannel}(\gamma, tid_A, tid_B)$

/ Return the setup procedure Setup required for the setup of the virtual channel.

/ The funding transaction and initial versions of split and commit transactions of

/ the virtual channel γ are created. Moreover, the punishment and refund

/ transactions are generated to be used in malicious cases.

$\mathtt{GenVChannelOutput}(\gamma_P, P)$

/ Return output θ of γ_P that will fund the virtual channel

Create virtual channel for I

/ React to creation of γ with funding source tid_A, tid_B in round t_0^I

Let γ_P be the channel with id $\gamma.\mathsf{subchan}(P)$ for $P \in \{A, B\}$.

Compute $\theta_P := \mathtt{GenVChannelOutput}(\gamma_P, P)$ for $P \in \{A, B\}$

Assign $\mathsf{Setup} \leftarrow \mathtt{SetupVChannel}^I(\gamma, tid_A, tid_B)$

if $\mathsf{UpdateChanSync}^I(\gamma_A, \vec{\theta}_A, \gamma_B, \vec{\theta}_B, \mathsf{Setup})$ returns UPDATE–OK Creation successful.

Close virtual channel for I

/ React to closure of γ in round t_0^I for some c_P, c_Q s.t. $c_P + c_Q = \gamma.\mathsf{cash}$

Let γ_P be the sub-channel $\gamma.\mathsf{subchan}(P)$ for $P \in \{A, B\}$.

Compute $\vec{\theta}_P = \{(c_P, \mathtt{One–Sig}_{pk_P}), (c_Q + \frac{\gamma.\mathsf{fee}}{2}, \mathtt{One–Sig}_{pk_I})\}$ for $P \in \{A, B\}$.

if $\mathsf{UpdateChanSync}^I(\gamma_A, \vec{\theta}_A, \gamma_B, \vec{\theta}_B, \perp)$ returns UPDATE–OK Close successful.

else Execute $\mathtt{Offload}^I(\gamma)$.

Punish for all parties

/ In every round check the ledger and punish misbehavior

for every open channel γ execute $\mathsf{Punish}(\gamma)$

$\mathsf{UpdateChan}^P(\gamma, \vec{\theta}, \mathsf{Setup})$ from [3]

/ Initiate update of γ with state $\vec{\theta}$ in τ_0 with setup procedure Setup.

$\mathsf{UpdateChanSync}^I(\gamma_1, \vec{\theta}_1, \gamma_2, \vec{\theta}_2, \mathsf{Setup})$

/ Initiate update of γ_i with state $\vec{\theta}_i$ with Setup for $i = 1$ and 2 simultaneously

/ using the same steps of UpdateChan. At each step, wait for both channels

/ before continuing. If one of them fails at any step, act as both failed.

$\mathsf{PreCreateChan}(\mathtt{TX}_{\mathtt{f}}^{\gamma})$ from [3]

/ Creates a channel γ with initial versions of split and commit transactions.

/ It follows the channel creation procedure given in [3], expect that

/ the funding transaction is not published in the end.

Fig. 13: Protocol for virtual channels. The protocol utilizes the generalized channel protocols from [3]. Specifically, the channel update protocol UpdateChan is used in a black-box fashion while also defining a synchronized version called UpdateChanSync. Moreover, the channel creation protocol PreCreateChan is used with the difference of not publishing the channel funding transaction of the virtual channel. The gray parts of the protocol differ between our tow constructions with and without validity and are specified in the protocol pseudocode and description.