Sleepy Channels: Bitcoin-Compatible Bi-directional Payment Channels without Watchtowers

Lukas Aumayr[1]
TU Wien
lukas.aumayr@tuwien.ac.at

Sri AravindaKrishnan Thyagarajan[1,2]
Carnegie Mellon University
t.srikrishnan@gmail.com

Giulio Malavolta
Max Planck Institute for Security and Privacy
giulio.malavolta@hotmail.it

Pedro Moreno-Sánchez
IMDEA Software Institute
pedro.moreno@imdea.org

Matteo Maffei
Christian Doppler Laboratory Blockchain Technologies for the Internet of Things, TU Wien
matteo.maffei@tuwien.ac.at

Abstract

Payment channels (PC) are a promising solution to the scalability issue of cryptocurrencies, allowing users to perform the bulk of the transactions off-chain without needing to post everything on the blockchain. Many PC proposals however, suffer from a severe limitation: Both parties need to constantly monitor the blockchain to ensure that the other party did not post an *outdated* transaction. If this event happens, the honest party needs to react promptly and engage in a *punishment* procedure. This means that prolonged absence periods (e.g., due to a power outage) may be exploited by malicious users. As a mitigation, the community has introduced *watchtowers*, a third-party monitoring the blockchain on behalf of off-line users. Unfortunately, watchtowers are either trusted, which is critical from a security perspective, or they have to lock a certain amount of coins, called collateral, for each monitored PC in order to be held accountable, which is financially infeasible for a large network.

We present *Sleepy Channels*, the first bi-directional PC protocol without watchtowers (or any other third party) that supports an unbounded number of payments and does not require parties to be persistently online. The key idea is to confine the period in which PC updates can be validated on-chain to a short, pre-determined time window, which is where the PC parties have to be online. This behavior is incentivized by letting the parties lock a collateral in the PC, which can be adjusted depending on their mutual trust and which they get back much sooner if they are online during this time window. Our protocol is compatible with any blockchain that is capable of verifying digital signatures (e.g., Bitcoin), as shown by our proof of concept. Moreover, Sleepy Channels impose a communication and computation overhead similar to state-of-the-art PC protocols while removing watchtower's collateral and fees for the monitoring service.

1 Introduction

Bitcoin has put forward an innovative payment paradigm both from the technical and the economical point of view. A permissionless and decentralized consensus protocol is leveraged to agree on the validity of the transactions that are afterwards added to an immutable ledger. This approach, however, severely restricts the transaction throughput of decentralized cryptocurrencies. For instance, Bitcoin supports about 10 transactions per second and requires confirmation times of up to 1 hour.

Payment channels (PC) [33] have emerged as one of the most promising scalability solutions. A PC enables an arbitrary number of payments between users while only two transactions are required on-chain. The most prominent example, currently deployed in Bitcoin, is the Lightning Network (LN) [23], which at the time of writing hosts bitcoins worth more than $170M$ USD, in a total of more than $27K$ nodes and more than $76K$ channels.

In a bit more detail, a PC between Alice and Bob is created with a single on-chain transaction *open-channel*, where users lock some of the coins into a shared output controlled by both users (e.g., requiring a 2-of-2 multisignature), effectively depositing their coins and creating the channel. Both users additionally make sure that they can get their coins back at a mutually agreed expiration time. After the channel has been successfully opened, they can pay each other arbitrarily many times by exchanging authenticated off-chain messages representing updates of their share of coins in the shared output. The payment channel can be finally closed by including a

[1]These two authors contributed equally.

[2]Part of the work was carried out when the author was at Friedrich Alexander Universität Erlangen-Nürnberg, Germany.

close-channel transaction on-chain that effectively submits the last authenticated distribution of bitcoins to the blockchain (or after the channel has expired).

Issue with bidirectional channels. While the initial versions of payment channels were unidirectional (i.e., only payments from Alice to Bob were allowed), several designs for bi-directional payment channels have been proposed so far. The technical crux of these protocols is to ensure that no coins are stolen between the mutually untrusted Alice and Bob. To illustrate the problem, imagine that the current balance of the channel *bal* is {Alice:10, Bob:5 }. Alice pays 3 coins to Bob, moving the channel balance to *bal'* as {Alice:7, Bob:8 }. At this point, Alice benefits from *bal* while Bob would benefit if *bal'* is the one established on-chain.

The different designs of bi-directional payment channels available so far provide alternative solutions for this crucial dispute problem (see Table 1). One approach consists on leveraging the existence of Trusted Execution Environment (TEE) at both Alice and Bob [25]. This approach, however, adds a trust assumption that goes against the decentralization philosophy of cryptocurrencies and it is unclear whether it holds in practice [11, 45]. Another approach consists on relying on a third-party committee [2, 10] to agree on the last balance accepted by Alice and Bob. Again, this adds an additional assumption on the committee and current proposals work only over smart contracts as those available in Ethereum.

The most promising approach in terms of reduced trust assumptions and backwards compatible with Bitcoin, which is the one implemented in the LN, is based on the encoding of a punishment mechanism that allows Alice (or Bob) rescue all the coins in a channel if Bob (or Alice) attempts to establish a *stale* or *outdated* balance on-chain. Following with the running example, after the balance bal' is established, Alice and Bob exchange with each other a revocation key associated to *bal* that effectively allows one of the parties to get all the coins from *bal* if it is published on-chain by the other party.

In detail, imagine that after bal' has been agreed and *bal* has been revoked, Alice (the case with Bob is symmetric) attempts to close the channel with balance *bal*. As soon as *bal* is added on-chain, a small punishment time δ is established within which Bob can transfer all coins in *bal* to himself with the corresponding revocation key. After δ has expired, *bal* is established as final. This mechanism with time δ is called *relative timelock*[1] in the blockchain folklore (i.e., relative to the time *bal* is published).

[1]This can be realized via `checkSequenceVerify` (CSV) script available in Bitcoin.

The reader might ask at this point: And what happens if Bob does not monitor the blockchain on time (e.g., Bob crashes or he is offline) to punish the publishing of *bal*? In that case, Alice effectively manages to publish an old state that would be more beneficial for her. Therefore, the above mechanism makes an important requirement for the channel users: Both Alice and Bob have to be online persistently to ensure that if one of them cheats, the other can punish within δ. However, if Alice and Bob are regular users, it is highly likely that they go offline sporadically if not for prolonged periods of time. Moreover, existing currencies like Monero do not possess the capability for *relative timelock* in their script, and therefore the approach falls short of backwards compatibility with some prominent currencies.

The role of watchtowers. In order to avoid this problem, honest users (Bob in our running example) can rely on a third party, called *Watchtower*, that does the punishing job on his behalf. Several constructions for watchtower have been proposed so far [44, 4, 22, 30, 29, 3], but they all share the same fundamental limitation: watchtowers are either trusted, which is critical from a security perspective, or they have to lock a certain amount of coins, called collateral, for each monitored channel in order to be held accountable, which is financially infeasible for a large network.

Given this state of affairs, in this work we investigate the following question: *Is it possible to design a secure, and practical payment channel protocol that does not require channel parties to be persistently online, nor additional parties (not even watchtowers) or additional trust assumptions, and is backwards compatible (no complex scripts) with current UTXO-based cryptocurrencies?*

1.1 Our Contribution

In this work, we answer this question in the affirmative. We design *Sleepy Channels* a new bi-directional payment channel protocol (Section 5) that does not require either of the channel parties to be persistently online, and therefore does not require the services of a watchtower. Our protocol allows users to schedule ahead of time when they have to come online to validate possible channel updates. Moreover, our protocol does not make use of any complex script and is therefore backwards compatible with existing UTXO-based cryptocurrencies.

At the core of our Sleepy Channels protocol, we have a novel collateral technique that plays a dual role: (1) Enables the punishment of a misbehaving channel user within a predetermined time, irrespective of when the cheating exactly takes place. In technical terms, we no

Table 1: Comparison among payment channel approaches. Online assumption refers to the honest user be online for revocation of an old state on-chain. Unrestricted lifetime means the protocol does not require users to close the channel before a pre-specified time. Unbounded payments refers to channel users making any number of payments while the channel is open. In terms of scripts, DS refers to digital signatures, SIGHASH_NOINPUT refers to a specific signature scheme [13], Seq. number refers to attaching a state number to a transaction and verifying if it is greater or smaller than the current height of the blockchain. In case of Duplex [14], d is the number of payments made in the channel. LRS refers to Linkable Ring Signature scheme used in Monero [37], and DLSAG refers to the transaction scheme proposed in [31].

	Bi-directional	Pre-schedule online	Unrestricted lifetime	Unbounded payments	#Tx. to close channel	Script Requirement[1]
Spillman [36]	✗	✓	✗	✓	1	DS
CLTV [39]	✗	✓	✗	✓	1	DS + CLTV
Duplex [14]	✓	✓[2]	✗	✗	$\log d$	DS + CLTV
Eltoo [13]	✓	✗	✓	✓	2	DS + CSV + SIGHASH_NOINPUT + Seq number
Lightning [23]	✓	✗	✓	✓	1	DS + CSV
Generalized [1]	✓	✗	✓	✓	2	DS[3] + CSV
Paymo [37]	✗	✓	✗	✓	1	Monero's LRS + CLTV
DLSAG [31]	✗	✓	✗	✓	1	DLSAG + CLTV
Teechan [25]	✓	✓	✓	✓	1	DS + TEE
This work	✓	✓	✗	✓	1	DS + CLTV
This work+[38]	✓	✓	✗	✓	1	DS

[1]: Requiring less script capabilities from the blockchain results in better compatibility with currencies, and better on-chain privacy (fungibility).

[2]: This requires that the transactions of the first level of the tree use CLTV instead of CSV.

[3]: The digital signature scheme used must have adaptor signature [1] capability.

longer require *relative timelocks* (CSV). (2) Incentivises a channel user to cooperate in closing the channel if the other channel user wishes to do so. Our collateral technique requires both users to lock same amount of collateral each, whose exact value is fully determined by the level of trust between the users: High trust level means a low collateral, while a low trust level means a high collateral.

Our protocol only involves signature generation on mutually agreed transactions, along with the use of *verifiable timed signatures* [38, 37] for achieving backward compatibility with existing currencies, especially privacy-preserving currencies like Monero for the first time. With the aid of techniques from [38, 37], the transactions in our protocol look exactly the same as any other regular transaction in the currency, thereby ensuring high fungibility. If the currency already supports `checkLockTimeVerify` (CLTV) script[2], then our protocol only requires signature generation. We evaluate the performance of our Sleepy Channels protocol in the presence of CLTV and our results show that the time and communication cost are inline with the highly efficient protocols used in LN [23].

1.2 Related Work

Spillman [36] and CLTV [39] proposed uni-directional payment channels between Alice and Bob where payments could only be made to Bob and thus the balance of Bob only increases. Therefore there was no payment revocation as Bob always preferred the most recent payment. Moreover the channel had a fixed expiry that is set at the time of the channel creation. Duplex channels [14] support bi-directional channels but only support a limited number of payments as with each successive payment, the lifetime of the channel decreases. Moreover, the protocol requires $\log d$ number of transactions to close the channel where d is the number of payments made. Other payment channel proposals typically require only one transaction to close. Eltoo [13] also supports bi-directional payments but requires special signature scheme like SIGHASH_NOINPUT, relative timelocks (CSV) and related scripts, and therefore is not compatible with several of the existing currencies, including Bitcoin itself. Lightning channels [23] are the most popular channels currently in use that support bi-directional payments but require relative timelocks (CSV). Generalized channels [1] support bi-directional payments but again require relative timelocks (CSV). More importantly they require the underlying signature scheme to support adaptor signatures [1] capability[3]. Paymo [37] and DLSAG [31] are proposals tailored for Monero that only support uni-directional payments. Teechan [25] is a bi-directional payment channel proposal but requires both users to possess TEEs. A summary of the comparison is presented in Table 1. Payment channels that support arbitrary conditional payments are referred to as *state channels* [15, 16, 10] and require complex scripts like *smart contracts* and are incompatible with UTXO-based currencies.

As discussed above, parties may avail the services of a third party like a *watchtower*. Monitor [44] is watchtower proposal requiring no special scripts. However an

[2]The script (available in Bitcoin) sets a transaction to be valid only after some pre-specified height (t) of the blockchain. That is, the transaction is set to be valid only after some point of time in the future.

[3]Recently it was shown that deterministic signatures do not possess adaptor signature capabilities [17], that includes signature schemes like BLS.

offline watchtower is not penalised and may even get rewarded if a revoked payment is successful on-chain. DCWC [3] is another such proposal that fails to penalise an offline watchtower where the honest user ends up losing coins as a revoked payment is posted on the chain. Outpost [22] requires a special OP_RETURN script and also requires the channel user (hiring the watchtower) to pay the watchtower for every channel update. PISA [29] heavily relies on smart contract support and also requires the watchtower to lock large collateral (equal to the channel capacity) along with the channel. Cerberus channels [4] and FPPW [30] are recent proposals that suffer from the problem of revealing the channel balance to the watchtower per update and therefore lack balance privacy. Similar to PISA, they also require the watchtower to lock large collateral along with the channel. All of the above watchtower proposals also fundamentally lack *channel unlinkability* as the watchtower can clearly track channel related transactions on-chain. Except for PISA, all of the above proposals still require relative timelocks (CSV), which can be replaced with absolute timelocks (CLTV) at the expense of restricted lifetimes for the channels.

2 Solution Overview

In this section we give a high level overview of our construction. We start by reviewing the state-of-the-art in payment channels, i.e., those employed in the Lightning Network [23], illustrating its limitations and, based on that, gradually introducing our solution.

Lightning channels. Two parties A and B lock up some money in a joint address (or channel). They can perform payments to each other by exchanging payment transactions tx_{Pay}, which commit to an updated balance of both users. Each party gets their own copy of this transaction, tx^A_{Pay} and tx^B_{Pay}, respectively, so that they can spend them unilaterally. In order for this mechanism to be secure, the parties need to revoke the previous state whenever an update is performed. This is done by exchanging a punishment transaction that gives the balance of the cheating user to the honest user, should the former try to post an old state. To give precedence to the punishment transaction, the party that posts their payment transaction tx_{Pay} is forced to wait for a relative timelock of Δ (in practice, one day) until they can spend their balance, in order to give time to the other party to punish. We illustrate this channel construction in Figure 1.

With this mechanism in place, a party that wants to prevent being cheated on needs to be online constantly throughout the lifetime of the channel and to monitor the blockchain for old states. If it does, it has Δ time units *immediately after* the posting of tx_{Pay} to perform the punishment. One workaround for this problem is to employ a trusted third party, a *Watchtower*, which takes over the responsibility of monitoring the ledger, thereby allowing a party to safely go offline. As pointed out previously, this approach has fundamental drawbacks such as the need for the Watchtower to lock up coins for each channel that it watches over, besides the fees requested by the Watchtower for its service.

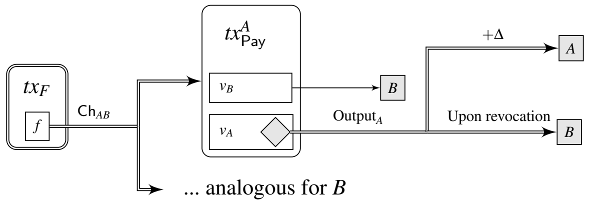

Figure 1: The transaction flow of LN channel between A and B. Rounded boxes represent transactions, rectangles within represent outputs of the transaction: here $v_A + v_B = f$. Incoming arrows represent transaction inputs, while outgoing arrows represent how an output can be spent. Double lines from transaction outputs indicate the output is a shared address. Single line from the transaction output indicate that the output is a single party address. We write the timelock (Δ) associated with a transaction over the corresponding arrow.

Replacing the relative timelock. A first attempt to solve the issue could be replacing the relative timelock of Δ time units in Figure 1 with an absolute timelock until time $\mathbf{T}$, i.e., by specifying $\mathbf{T}$ as a block height using the CLTV script. In other words, the party A that posts a state tx^A_{Pay} has to wait until time $\mathbf{T}$ (irrespectively of when tx^A_{Pay} is posted on the chain) before it can retrieve the funds. This allows B to safely go offline during the channel lifetime and only come back shortly before $\mathbf{T}$ to check if an old state was posted by A. Unfortunately, in this case an honest party that posts the latest state still needs to wait until $\mathbf{T}$ expires before having access to their funds and note that $\mathbf{T}$ could span several weeks (if $\mathbf{T}$ is too short, then either parties frequently update the channel state or the channel is closed to reflect the current state).

To fix this issue, we could implement a mechanism that unlocks A's funds as soon as B claims its own funds from tx^A_{Pay}. Another way to think of this is that B by unlocking its funds gives a confirmation that tx^A_{Pay} is indeed the latest state. However, note that the balance that B committed to in the latest state can be very small or even 0, such that the incentive for B to give this fast confirmation is small or nonexistent. This leaves A to wait for a potentially long time and opens the door to *Denial-of-Service* (DoS) attacks from B.

Incentivizing fast unlock. In the extreme case where B has balance 0, we need an incentive for B to unlock the channel early. Taking a step towards our solution, we let B add a collateral of amount c equal to the channel capacity f. B's collateral is set such that it remains locked until B gives a fast confirmation for unlocking A's coins. Note that A's coins are now guaranteed to be smaller (or equal in the worst case) to the amount of coins B has locked. This means that a malicious B attempting to perform a DoS attack on A, ends up locking at least as many coins from itself until $\mathbf{T}$. Analogously, A has to put the same amount c as a collateral for the symmetric case.

Making the collateral dynamic. We further refine this solution by changing c from the total capacity of the channel f to a parameter chosen by both parties of the channel. Depending on the level of trust between the two parties, the value of c can be anything from 0 up to f. Once the two parties agreed on a value for c, during the funding of the channel, they can fund the channel with the total channel capacity f plus the additional collateral $2c$ (c from each party). Note that the payments are still made with the channel capacity of f and the collateral coins $2c$ are only used as incentive for fast closing of channels. And after the closing, both party A and B get back their original collateral amounts of c coins each.

There is still one problem left though. Again, if the balance of B is 0 and A's balance is the capacity of the channel f, then B can lock up c coins and will lock up $c+f$ coins of A before the fast confirmation. In a final improvement, we resolve this issue by refining the tx_{Pay} transaction so that the posting party gets back their part of the collateral. This is safe since the collateral serves merely the purpose to incentivize the party who did not post the transaction, to acknowledge that that transaction does indeed corresponds to the latest channel state. Note that the posting party only unlocks its collateral right away and not its channel balance set by tx_{Pay}. Indeed, in the extreme case, if A posts tx^{A}_{Pay} on the chain, A can redeem its collateral c coins immediately while B locks up c coins and A locks up only f coins. If $c = f$, notice that B has locked the same amount of coins as A, which discourages B from launching a DoS attack on A.

Overcoming the drawbacks. With the presented construction, we indeed manage to achieve bidirectional channels with unbounded payments without the need for users to constantly be online and monitor the blockchain. Instead, they can safely go offline and can come back only shortly before the pre-defined lifetime $\mathbf{T}$ of the channel. Further, our construction requires only digital signatures and absolute timelocks in the form of CLTV or VTS [38], as we show in Section 5.

3 Preliminaries

We denote by $\lambda \in \mathbb{N}$ the security parameter and by $x \leftarrow \mathcal{A}(\mathsf{in}; r)$ the output of the algorithm $\mathcal{A}$ on input in using $r \leftarrow \{0,1\}^*$ as its randomness. We often omit this randomness and only mention it explicitly when required. We consider *probabilistic polynomial time* (PPT) machines as efficient algorithms.

Universal Composability. We model security in the *universal composability* framework with global setup [9], which lets us model concurrent executions. We consider a set of parties $\mathcal{P} = \{P_1, \dots, P_n\}$ that is running the protocol. Further, we assume *static* corruptions, where the adversary $\mathcal{A}$ announces at the beginning which parties he corrupts. We denote the environment by $\mathcal{E}$, which captures anything that happens "outside the protocol execution". We model a synchronous communication by using a global clock $\mathcal{F}_{clock}$ capturing execution rounds. Additionally, we assume authenticated communication with guaranteed delivery between users, captured by $\mathcal{F}_{GDC}$.

For a real protocol Π and an adversary $\mathcal{A}$ we write $EXEC_{\Pi,\mathcal{A},\mathcal{E}}$ to denote the ensemble corresponding to the protocol execution. For an ideal functionality $\mathcal{F}$ and an adversary $\mathcal{S}$ we write $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ to denote the distribution ensemble of the ideal world execution.

Definition 3.1 (Universal Composability)**.** *A protocol τ UC-realizes an ideal functionality $\mathcal{F}$ if for any PPT adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for any environment $\mathcal{E}$ the ensembles $EXEC_{\tau,\mathcal{A},\mathcal{E}}$ and $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable.*

Digital Signatures. A digital signature scheme DS, lets a user authenticate a message by signing it with respect to a public key. Formally, we have a key generation algorithm $\mathsf{KGen}(1^\lambda)$ that takes the security parameter 1^λ and outputs the public/secret key pair (pk, sk), a signing algorithm $\mathsf{Sign}(sk, m)$ that inputs sk and a message $m \in \{0,1\}^*$ and outputs a signature σ, and a verification algorithm $\mathsf{Vf}(pk, m, \sigma)$ that outputs 1 if σ is a valid signature on m under the public key pk, and outputs 0 otherwise. We require the standard notion unforgeability for the signature scheme [21]. A stronger notion of strong unforgeability for the signature scheme was shown to be equivalent to the UC formulation of security [5].

2-Party Computation. The aim of a secure 2-party computation (2PC) protocol is for the two participating users P_0 and P_1 to securely compute some function f over their private inputs x_0 and x_1, respectively. Apart from output correctness, we require *privacy*, i.e., the only information learned by the parties in the computation is the one determined by the function output. Note that we require the standard *security with aborts*, where the adversary can decide whether the honest party will receive the output of the computation or not. In other words, we do not assume any form of fairness or guaranteed output delivery. For a comprehensive treatment of the formal UC definition we refer the reader to [8]. In this work, we make use of 2-party signing key generation (Γ_{JKGen}) and 2-party signature generation (Γ_{Sign}) protocols [26, 18, 6].

Blockchain and Transaction Scheme. We assume the existence of an ideal ledger (blockchain) functionality $\mathbb{B}$ [28, 27, 1] that maintains the list of coins currently associated with each address (denoted by addr) and that we model as a trusted append-only bulletin board. The corresponding ideal functionality $\mathcal{F}_{\mathbb{B}}$ maintains the ledger $\mathbb{B}$ locally and updates it according to the transactions between users. Transactions are generated by the transaction function tx: A transaction tx_A that is generated as $tx_A := tx([\mathsf{addr}_1,\ldots,\mathsf{addr}_n],[\mathsf{addr}'_1,\ldots,\mathsf{addr}'_m],[v_1,\ldots,v_m])$, such that it transfers all the coins (say v coins) from the source addresses $[\mathsf{addr}_1,\ldots,\mathsf{addr}_n]$ to the destination addresses $[\mathsf{addr}'_1,\ldots,\mathsf{addr}'_m]$ such that v_1 coins are sent to addr'_1, v_2 coins are sent to addr'_2 and so on, where $v_1 + v_2 + \cdots v_m = v$. Addresses are typically public keys of digital signature schemes and the transaction is authenticated with a valid signature with respect to each of the source addresses $[\mathsf{addr}_1,\ldots,\mathsf{addr}_n]$ (as the public keys). We consider *Unspent Transaction Output* (UTXO) model where an address is tied to the transaction that creates it and is spendable (used as input to a transaction) *exactly once*, like in Bitcoin, Monero, etc.

4 Ideal Functionality Bi-directional Channels

We define an ideal functionality $\mathcal{F}$ that closely follows the bi-directional payment functionality defined in [1]. In fact, our functionalities captures the same security and efficiency notions, except that we achieve *delayed finality with punish*, which means that the channel owner has the guarantee that until time $\mathbf{T}$, the time until which the latest state is locked, either that state or one that gives all the money to the honest party can be enforced on the ledger. Whenever one party tries to close the channel with the latest state, the other party is incentivized to confirm it before $\mathbf{T}$, thereby unlocking not only the state but also their collateral c.

Specific Notation. We abbreviate γ as an attribute tuple containing the following information $\gamma := (\gamma.\mathsf{id},\gamma.\mathsf{users},\gamma.\mathsf{cash},\gamma.\mathsf{st},\gamma.\mathbf{T},\gamma.c)$, where $\gamma.\mathsf{id} \in \{0,1\}^*$ is the channel identifier, $\gamma.\mathsf{users}$ defines the two users of the channel, $\gamma.\mathsf{cash} \in \mathbb{R}_{\geq 0}$ the total capacity, $\gamma.\mathsf{st}$ defines a list of outputs (addresses and values) in, $\gamma.\mathbf{T} \in \mathbb{R}_{\geq 0}$ defines the lifetime of the channel, and $\gamma.c \in \mathbb{R}_{\geq 0}$ defines the collateral of the channel.

We denote by $m \overset{\tau}{\hookrightarrow} P$ the output of message m to party P in round τ. Similarly, $m \overset{\tau}{\hookleftarrow} P$ denotes the input of message m in round τ. A message m generally consists of ($\mathtt{MESSAGE\text{-}ID}$,*parameters*). For better readability, we omit session identifiers in messages. In our communication model, messages sent between parties are received in the next round, i.e., if A sends a message to B in round τ, B will receive it in round $\tau + 1$. Messages sent to the environment, the simulator $\mathcal{S}$ or to $\mathcal{F}$ are received in the same round.

Description. As we do not consider privacy notions, we say that $\mathcal{F}$ implicitly forwards all messages to the $\mathcal{S}$. Note that $\mathcal{F}$ cannot create signatures or prepare transaction ids. It expects the $\mathcal{S}$ to perform these tasks, e.g., expecting a transaction of a certain structure to appear on the ledger, and outputting ERROR, if this does not happen. Similarly, whenever the functionality expects the $\mathcal{S}$ to provide or set a value, but the $\mathcal{S}$ does not do it, the functionality implicitly outputs ERROR, where all guarantees are potentially lost. Hence, we are interested only in protocols that realize $\mathcal{F}$, but never output ERROR.

$\mathcal{F}$ interacts with a ledger $\mathbb{B}(\Delta,\Sigma,\mathcal{V})$ parameterized over a given upper bound Δ, after which valid transactions are appended to the ledger, a signature scheme Σ and a set $\mathcal{V}$, defining valid spending conditions, including signature verification under Σ and absolute timelocks. $\mathcal{F}$ can see the transactions on the ledger and infer ownership of coins. Following [1], we keep the functionality $\mathcal{F}$ description generic, by parameterizing it over T_p and k, both of which are independent of Δ. T_p is an upper bound on the number of consecutive off-chain communication rounds between two users, while k defines the number of states that a channel has. We present a protocol later, where $k = 2$. Both T_p and Δ are defined as upper bounds. If the actual values are less, $\mathcal{S}$ implicitly informs $\mathcal{F}$ of these values.

The ideal functionality keeps a map Γ, which maps the id of an existing channel to the channel tuple γ representing the latest state and the address of the funding transaction, Ch_{AB}. Note that during an update, there may be two states that are active $\{\gamma,\gamma'\}$. We give a formal description of $\mathcal{F}^{\mathbb{B}(\Delta,\Sigma,\mathcal{V})}$ (which we abbreviate as $\mathcal{F}$) in Figure 2. Following, we explain our functionality in prose and argue inline, why certain security and efficiency goals hold.

Create. When both parties of channel γ send a message ($\mathtt{CREATE}$,γ,tid_P) to $\mathcal{F}$ within T_p rounds, $\mathcal{F}$ expects a funding transaction to appear on $\mathbb{B}$ within Δ

rounds, spending both inputs tid_A and tid_B and holding $\gamma.\mathsf{cash} + 2\gamma.c$ coins. The channel funding address Ch_{AB} is stored in Γ and $\mathtt{CREATED}$ is sent to both parties.

Update. One party P initiates the update with $(\mathtt{UPDATE}, \mathsf{id}, \overrightarrow{\theta}, t_{\mathsf{stp}})$, where id refers to the channel identifier, $\overrightarrow{\theta}$ represents the new state (e.g., coin distribution or other applications that work under *delayed finality with punish*) and t_{stp} denotes the time needed to setup anything that is built on top of the channel. First, the parties agree on the new state. For this, $\mathcal{S}$ informs $\mathcal{F}$ of a vector of k transactions. Both parties can abort here by P not sending $\mathtt{SETUP{-}OK}$ and Q not sending $\mathtt{UPDATE{-}OK}$. When P receives $\mathtt{UPDATE{-}OK}$, they move on to the revocation. $\mathcal{F}$ expects a message $\mathtt{REVOKE}$ from both parties, and in the success case, $\mathtt{UPDATED}$ is output to both parties. In case of an error, the ForceClose subprocedure is executed, which expects the funding transaction of the channel to be spent within Δ rounds.

Close. Either party can initiate a channel's closure by sending $(\mathtt{CLOSE}, \mathsf{id})$ to $\mathcal{F}$. If the other party sends the same message within T_p rounds, $\mathcal{F}$ expects a transaction representing the latest state of the channel to appear on the ledger within Δ rounds. Should only one party request the closure or in case one party is corrupted, $\mathcal{F}$ expects either the a transaction representing the latest of the channel or an older state, followed by a punishment (see Punish). If the funding transaction remains unspent, outputs $\mathtt{ERROR}$.

Punish. To give honest parties the guarantee that either the most recent state of the channel which is locked until at most time $\mathbf{T}$ can be enforced on $\mathbb{B}$, or the honest party can get all coins, we need the punish phase. This check is executed in each round. We can model this in the UC framework, by expecting $\mathcal{E}$ to pass the execution token in every round. If $\mathcal{E}$ fails to do that, $\mathcal{F}$ outputs an error the next time it has the execution token. Whenever the funding transaction of any open channel γ in Γ is spent, $\mathcal{F}$ expects either a transaction that spends the coins in accordance to the latest state of γ, or a transaction giving $\gamma.\mathsf{cash} + \gamma.c$ coins to the honest party. Else, $\mathtt{ERROR}$ is output. In the case that a transaction in accordance to the latest state of γ appears on the ledger, either the funds of the party that has posted the transaction are locked until $\mathbf{T}$ (after which a transaction claiming them appears) or the other party unlocks them beforehand by unlocking their own funds and collateral. In the latter case, the other party loses the negligible amount (which we say is a system parameter in $\mathbb{R}_{\geq 0}$ for a ledger $\mathbb{B}$) to the first party.

5 Sleepy Channels: Our Bi-Directional Payment Channel Protocol

In this section we describe our Sleepy Channel protocol for realizing bi-directional payment channels for a currency whose transaction scheme makes use of the signature scheme Π_{DS} for authentication. For simplicity we assume the transaction scheme lets verify transaction timeouts[4], meaning that a transaction is considered valid only if it is posted after a specified timeout $\mathbf{T}$ has passed. We discuss in Section 5.1.3 how we can remove this assumption from the transaction scheme. We additionally make use of 2-party protocols whose functionality we describe below.

2-Party Key Generation. Parties A and B can jointly generate keys for a signature scheme Π_{DS}. We denote this interactive protocol by Γ_{JKGen}. It takes as input the public parameters pp from both parties and outputs the joint public key pk to both parties and outputs the secret key share sk_A to A and sk_B to B.

2-Party Signing. Parties A and B having a shared key can jointly sign messages with respect to the signature scheme Π_{DS}. We denote this interactive protocol by Γ_{Sign}. It takes as input the message m and the shared public key pk from both parties and secret key shares sk_A and sk_B from A and B, respectively. The protocol outputs the signature σ (to one of the parties), such that $\Pi_{\mathsf{DS}}.\mathsf{Vf}(pk, m, \sigma) = 1$.

We can instantiate both 2-party protocols (Γ_{JKGen} or Γ_{Sign}) with efficient interactive protocols for specific signatures schemes of interest. If the currencies use ECDSA signatures, Schnorr signatures or BLS signatures [7, 12] for transaction authentication, we can instantiate Γ_{JKGen} and Γ_{Sign} with protocols from [26], [18], or [6], respectively. Monero uses a linkable ring signature scheme [37, 31] for authentication and the corresponding tailored 2-party protocols for key generation and signing are described in [37].

5.1 Our Protocol

We consider parties A and B already have an open channel Ch_{AB} which is a shared public key pk_{AB} (between A and B) and the corresponding secret key sk_{AB} is shared among the parties. Parties can make multiple payments using the channel (in either direction) and confirm the final payment state on the chain. However, after each payment, the payment state of the channel is updated and accordingly old states are revoked. The formal description of the protocol can be found in Figures 4 and 5.

[4]Realizable through the `locktime` script that is available in Bitcoin.

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

<u>Create:</u> Upon $(\texttt{CREATE}, \gamma, tid_A) \xleftrightarrow{\tau_0} A$, distinguish:

Both agreed: If already received $(\texttt{CREATE}, \gamma, tid_B) \xleftrightarrow{\tau} B$, where $\tau_0 - \tau \leq T_p$: If $tx_F := tx([tid_A, tid_B], \mathsf{Ch}_{AB}, \gamma.\mathsf{cash} + 2\gamma.c)$ for some address Ch_{AB} appears on $\mathbb{B}$ in round $\tau_1 \leq \tau + \Delta + T_p$, set $\Gamma(\gamma.\mathsf{id}) := (\{\gamma\}, \mathsf{Ch}_{AB})$ and $(\texttt{CREATED}, \gamma.\mathsf{id}) \xhookrightarrow{\tau_1} \gamma.\mathsf{users}$. Else stop.

Wait for B: Else wait if $(\texttt{CREATE}, \mathsf{id}) \xleftrightarrow{\tau \leq \tau_0 + T_p} B$ (then, "Both agreed" option is executed). If such message is not received, stop.

<u>Update:</u> Upon $(\texttt{UPDATE}, \mathsf{id}, \overrightarrow{\theta}, t_{\mathsf{stp}}) \xleftrightarrow{\tau_0} A$, parse $(\{\gamma\}, \mathsf{Ch}_{AB}) := \Gamma(\mathsf{id})$, set $\gamma' := \gamma$, $\gamma'.\mathsf{st} := \overrightarrow{\theta}$:

1. In round $\tau_1 \leq \tau_0 + T_p$, let $\mathcal{S}$ define $\overrightarrow{tid}$ s.t. $|\overrightarrow{tid}| = k$. Then $(\texttt{UPDATE–REQ}, \mathsf{id}, \overrightarrow{\theta}, t_{\mathsf{stp}}, \overrightarrow{tid}) \xhookrightarrow{\tau_1} B$ and $(\texttt{SETUP}, \mathsf{id}, \overrightarrow{tid}) \xhookrightarrow{\tau_1} A$.
2. If $(\texttt{SETUP–OK}, \mathsf{id}) \xleftrightarrow{\tau_2 \leq \tau_1 + t_{\mathsf{stp}}} A$, then $(\texttt{SETUP–OK}, \mathsf{id}) \xhookrightarrow{\tau_3 \leq \tau_2 + T_p} B$. Else stop.
3. If $(\texttt{UPDATE–OK}, \mathsf{id}) \xleftrightarrow{\tau_3} B$, then (if B honest or instructed by $\mathcal{S}$) send $(\texttt{UPDATE–OK}, \mathsf{id}) \xhookrightarrow{\tau_4 \leq \tau_3 + T_p} A$. Else distinguish:
   - If B honest or if instructed by $\mathcal{S}$, stop *(reject)*. Else set $\Gamma(\mathsf{id}) := (\{\gamma, \gamma'\}, \mathsf{Ch}_{AB})$, run $\texttt{ForceClose}(\mathsf{id})$ and stop.
4. If $(\texttt{REVOKE}, \mathsf{id}) \xleftrightarrow{\tau_4} A$, send $(\texttt{REVOKE–REQ}, \mathsf{id}) \xhookrightarrow{\tau_5 \leq \tau_4 + T_p} B$. Else set $\Gamma(\mathsf{id}) := (\{\gamma, \gamma'\}, \mathsf{Ch}_{AB})$, run $\texttt{ForceClose}(\mathsf{id})$ and stop.
5. If $(\texttt{REVOKE}, \mathsf{id}) \xleftrightarrow{\tau_5} B$, $\Gamma(\mathsf{id}) := (\{\gamma'\}, \mathsf{Ch}_{AB})$, send $(\texttt{UPDATED}, \mathsf{id}, \overrightarrow{\theta}) \xhookrightarrow{\tau_6 \leq \tau_5 + T_p} \gamma.\mathsf{users}$ and stop *(accept)*. Else set $\Gamma(\mathsf{id}) := (\{\gamma, \gamma'\}, \mathsf{Ch}_{AB})$, run $\texttt{ForceClose}(\mathsf{id})$ and stop.

<u>Close:</u> Upon $(\texttt{CLOSE}, \mathsf{id}) \xleftrightarrow{\tau_0} A$, distinguish

Both agreed: If already received $(\texttt{CLOSE}, \mathsf{id}) \xleftrightarrow{\tau} B$, where $\tau_0 - \tau \leq T_p$, let $(\{\gamma\}, \mathsf{Ch}_{AB}) := \Gamma(\mathsf{id})$ and distinguish:

- If $tx_c := tx(\mathsf{Ch}_{AB}, [out_A, out_B], [\gamma.c + \gamma.\mathsf{st.bal}(A), \gamma.c + \gamma.\mathsf{st.bal}(B)])$ appears on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + \Delta$, set $\Gamma(\mathsf{id}) := \perp$, send $(\texttt{CLOSED}, \mathsf{id}) \xhookrightarrow{\tau_1} \gamma.\mathsf{users}$ and stop.
- Else, if at least one of the parties is not honest, run $\texttt{ForceClose}(\mathsf{id})$. Else, output $(\texttt{ERROR}) \xhookrightarrow{\tau_0 + \Delta} \gamma.\mathsf{users}$ and stop.

Wait for B: Else wait if $(\texttt{CLOSE}, \mathsf{id}) \xleftrightarrow{\tau \leq \tau_0 + T_p} B$ (in that case "Both agreed" option is executed). If such message is not received, run $\texttt{ForceClose}(\mathsf{id})$ in round $\tau_0 + T_p$.

<u>Punish:</u> (executed at the end of every round τ_0) For each $(X, \mathsf{Ch}_{AB}) \in \Gamma$ check if $\mathbb{B}$ contains a transaction $tx^A_{\mathsf{Pay},i} := tx(\mathsf{Ch}_{AB}, o_C, v_C)$ for some addresses o_C and some values v_C, s.t. $\sum_{v \in v_C} = \gamma.\mathsf{cash}$ and one address $o \in o_C$ belongs to A with the corresponding value $v \in v_C = \gamma.c$ for some $A \in \gamma.\mathsf{users}$ and $B \in \gamma.\mathsf{users} \setminus \{A\}$. If yes, then define $L := \{\gamma.\mathsf{st} \mid \gamma \in X\}$ and distinguish:

Punish: If B is honest and $tx^A_{\mathsf{Pay},i}$ does not correspond to the most recent state in X, $tx^B_{\mathsf{Pnsh},i} := tx(o \in o_C, o_P, \gamma.\mathsf{st.bal}(A))$, where o_P is an address controlled by B, appears on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + \Delta$. Afterwards, in round $\tau_2 \leq \tau_1 + \Delta$ a transaction $tx^{A,B}_{\mathsf{Fpay},i} := (o \in o_C, o_S, v_S)$, for some addresses o_S and corresponding values v_S where one address $o \in o_S$ belongs to B and the corresponding value of o is $\gamma.\mathsf{st.bal}(B) + \gamma.c - \varepsilon$, appears on $\mathbb{B}$, set $\Gamma(\mathsf{id}) = \perp$, send $(\texttt{PUNISHED}, \mathsf{id}) \xhookrightarrow{\tau_2} B$ and stop.

Close: Either $\Gamma(\mathsf{id}) = \perp$ before round $\tau_0 + \Delta$ (channel was peacefully closed) or after round $\tau_1 \leq \tau_0 + \Delta$ a transaction $tx^{A,B}_{\mathsf{Fpay},i} := (o \in o_C, o_S, v_S)$, for some addresses o_S and corresponding values v_S where one address $o \in o_S$ belongs to B and the corresponding value of o is $\gamma.\mathsf{st.bal}(B) + \gamma.c - \varepsilon$, appears on $\mathbb{B}$ before a transaction $tx^{A*}_{\mathsf{Fpay},i} := ([o \in o_C, o' \in o_S], o_F, \gamma.\mathsf{st.bal}(A) + \varepsilon)$ where address o_F of A appears on $\mathbb{B}$. Set $\Gamma(\mathsf{id}) := \perp$ and send $(\texttt{CLOSED}, \mathsf{id}) \xhookrightarrow{\tau_2 \leq \tau_1 + \Delta} \gamma.\mathsf{users}$. Else, transaction $tx^{A,A}_{\mathsf{Fpay},i} := tx(o \in o_C, o_E, \gamma.\mathsf{st.bal}(A))$ where address o_E of A appears on $\mathbb{B}$ in round $\tau_3 \leq \gamma.\mathbf{T} + \Delta$. Set $\Gamma(\mathsf{id}) := \perp$ and $(\texttt{CLOSED}, \mathsf{id}) \xhookrightarrow{\tau_3} \gamma.\mathsf{users}$ and stop.

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

Subprocedure <u>$\texttt{ForceClose}(\mathsf{id})$:</u> Let τ_0 be the current round and $(\gamma, tx) := \Gamma(\mathsf{id})$. If within Δ rounds tx is still an unspent transaction on $\mathbb{B}$, then $(\texttt{ERROR}) \xhookrightarrow{\tau_0 + \Delta} \gamma.\mathsf{users}$ and stop. Else, latest in round $\gamma.\mathbf{T} + \Delta$, $m \in \{\texttt{CLOSED}, \texttt{PUNISHED}, \texttt{ERROR}\}$ is output via Punish.

Figure 2: Ideal Functionality

5.1.1 High Level Overview

We present below the intuition for our protocol using the transaction flow presented in Figure 3.

Payment. For each payment from the channel Ch_{AB}, parties generate two versions of transactions, one version under the control of party A and the other in the control of party B. By "under control", we mean that in party A's

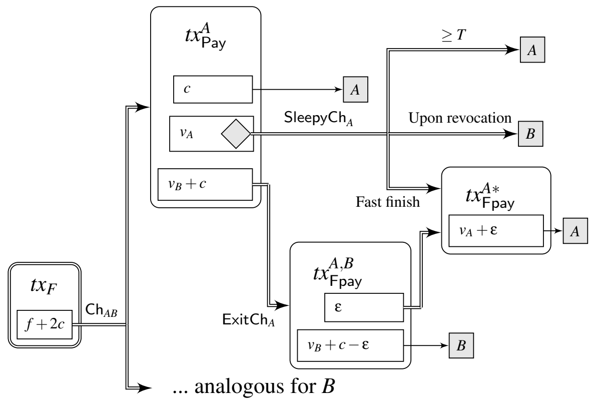

Figure 3: Transaction flow of our protocol. Here double lines from transaction outputs indicate that the output is a 2-party shared address between A and B. Single line from the transaction output indicate that the output is a single party address. We have $v_A + v_B = f$ and ε is some negligible amount of coins.

version, A has the necessary signatures to post the payment transaction tx^A_{Pay}. Analogously, B has the necessary signature to post the payment transaction tx^B_{Pay}. Both of these transactions spend from Ch_{AB}. In contrast to prior bi-directional protocols, both versions have an important asymmetry in the coin distribution among the parties.

In more detail, the channel Ch_{AB} holds in total $f + 2c$ coins where f is the payment capacity among the parties, while $2c$ is the collateral amount locked by both parties A and B with c coins from each. The value of c is agreed upon by the parties locally before they open the channel and are returned to the respective parties at the close of the channel. Consider a payment where A's balance is v_A and B's is v_B such that $v_A + v_B = f$. The payment transaction tx^A_{Pay} splits the funds of Ch_{AB} in the following way: (1) c coins to an address fully controlled by A, (2) v_A coins to a shared address between A and B referred to as the sleepy channel $\mathsf{SleepyCh}_A$, and (3) $v_B + c$ coins to a shared address between A and B referred to as the exit channel ExitCh_A.

Notice that A can immediately get c coins from output (1). To spend from output (2) (the sleepy channel $\mathsf{SleepyCh}_A$) which is a shared address, parties sign 2 different transactions.

1. Transaction $tx^{A,A}_{\mathsf{Fpay}}$, that transfers v_A to an address of A, but is valid only after a timeout $\mathbf{T}$.

2. Transaction tx^{A*}_{Fpay}, that spends from $\mathsf{SleepyCh}_A$ and an auxiliary address aux_A (contains ε coins as output in $tx^{A,B}_{\mathsf{Fpay}}$ in Figure 3) that is also a shared address between A and B. The transaction transfers v_A coins from $\mathsf{SleepyCh}_A$ and ε (a negligible amount) from aux_A, to an address of A.

The signatures on both of the above transactions are possessed by A and not B.

To spend from output (3) (the exit channel ExitCh_A) which is a shared address, parties sign a transaction $tx^{A,B}_{\mathsf{Fpay}}$ that transfers ε coins to the auxiliary address aux_A and $v_B + c - \varepsilon$ coins to an address of B. Notice that B's balance v_B and its collateral c (minus a negligible amount ε) are transferred together to B's address. In contrast to output (2), the signature on $tx^{A,B}_{\mathsf{Fpay}}$ is only available with B and not A. The version for B following tx^B_{Pay} is analogous to what we saw above except the roles are reversed.

Close. To close the channel with this payment state, we have two scenarios where either both parties are responsive, or one of them is unresponsive. For simplicity we consider A as the party closing the channel and B is either responsive or not. If B is responsive, party A posts tx^A_{Pay} with the corresponding signature that it has, on the blockchain. Since B is responsive, it posts the transaction $tx^{A,B}_{\mathsf{Fpay}}$ spending from ExitCh_A with the corresponding signature that it has, on the blockchain. Note that B now retrieves its balance v_B and collateral c, while one of the outputs of the transaction is aux_A. Now party A can finish the payment fast, by posting the transaction tx^{A*}_{Fpay} that spends from $\mathsf{SleepyCh}_A$ and aux_A simultaneously, thus retrieving its balance v_A (plus some ε). Recall that A can already retrieve its collateral c by itself.

In the latter case where B is unresponsive, party A posts tx^A_{Pay} on the blockchain as above. Now, A waits until the timeout $\mathbf{T}$ and posts the transaction $tx^{A,A}_{\mathsf{Fpay}}$ that retrieves v_A coins from $\mathsf{SleepyCh}_A$ to itself. Party B can retrieve $v_B + c - \varepsilon$ coins from ExitCh_A anytime it wishes.

Payment Revocation and Punishment. When the parties want to revoke the payment, they together generate a punishment transaction tx^A_{Pnsh} that spends from $\mathsf{SleepyCh}_A$ to an address of B. The parties generate a signature on this transaction such that B holds the signature. Similar punishment transaction and signature are generated in B's version where A holds the signature for the transaction. In total, the parties have three different transactions spending from the sleepy channel $\mathsf{SleepyCh}_A$.

If party A misbehaves, and posts tx^A_{Pay} after it has been revoked, party B has until timeout $\mathbf{T}$ to punish this behaviour by posting tx^A_{Pnsh} and the corresponding signature. This results in B getting the v_A coins. Party B then posts the transaction $tx^{A,B}_{\mathsf{Fpay}}$ spending from ExitCh_A retrieving $v_B + c - \varepsilon$. In effect, A only gets its collateral back, while B is able to retrieve the entire payment capacity f and its own collateral c.

Collateral As Incentive. Observe that the collateral that is with the channel funds are retrieved by the respective parties during closing, irrespective of a cheating event. This is because the purpose of the collateral in the Sleepy Channels protocol is to incentivise fast closure of the channel if one of the parties wishes to close the channel. Notice that if party A wishes to close the channel with an unrevoked payment, it posts the corresponding payment transaction tx^{A}_{Pay} on the chain. Now, A immediately retrieves its collateral c, while A's channel balance v_A, and B's channel balance and collateral, i.e., $v_B + c$ are still lying unspent in the outputs of tx^{A}_{Pay}. If value of c is high enough, party B is discouraged from launching a DoS attack on A: where party B does not retrieve the coins from ExitCh_A and lets party A wait until the timeout $\mathbf{T}$ to get v_A back. To see this, if party B attempts to launch the DoS attack on A, party B itself locks $v_B + c - \varepsilon$ coins in ExitCh_A until $\mathbf{T}$. On the other hand, if B retrieves its coins from ExitCh_A immediately, party A also can retrieve its coins from $\mathsf{SleepyCh}_A$ immediately with the aid of aux_A.

The value of c is determined by the level of trust between A and B. If both parties completely trust each other, the collateral c is set to 0. In the worst case where they do not trust each other at all, the collateral is set to be equal to the payment capacity, i.e., $c = f$ and have $v_A \leq v_B + c - \varepsilon$ when $\varepsilon \approx 0$. This means that during the DoS attack, party B locks at least the same amount of coins in ExitCh_A as party A does in $\mathsf{SleepyCh}_A$. Therefore, by not letting A spend its coins until timeout $\mathbf{T}$, party B also can not spend the same amount of coins until timeout $\mathbf{T}$.

5.1.2 Security

In this section we state our main theorem and we informally outline the main steps our our analysis. In Appendix A we give a formal description of our Sleepy Channels protocol Π in the UC framework. It differs from the protocol Π'' in Section 5 in that the cryptographic protocols for 2-party key generation and 2-party signing are substituted by the corresponding ideal functionalities. This is captured by the following Lemma.

Lemma 1. *Let Γ_{JKGen} be a UC-secure 2-party key-generation protocol and let Γ_{Sign} be a UC-secure 2-party signing protocol. Then the protocols Π and Π'' are computationally indistinguishable from the point of view of the environment $\mathcal{E}$.*

In Appendix A.1 we describe a simulator $\mathcal{S}$ that interacts with the ideal functionality $\mathcal{F}$ (defined in Section 4), whereas the environment interacts with $\phi_{\mathcal{F}}$ (the ideal protocol for $\mathcal{F}$). Then in Appendix A.2 we show that any attack that can be carried out against Π can also be carried out against $\phi_{\mathcal{F}}$. This allows us to state the following theorem.

Theorem 5.1. *The protocol Π UC-realizes the the ideal functionality $\mathcal{F}$.*

5.1.3 Extensions

In this section we describe possible extensions of our protocol that makes it applicable in a wider class of settings.

TimeLock script independence. The curious reader may wonder whether our protocol achieves the sought-after goal of (bi-directional) payment channels needing *only* the signature verification script from the underlying blockchain. Although we remove the dependency on *relative* timelock scripts, our protocol still relies on *absolute* timelock scripts (see point 1 in finish-payment transactions Figure 4) to guarantee the closure of the channel after some (fixed) time $\mathbf{T}$. Thus a natural question is whether one can construct bidirectional payment channels without relying on time-lock scripts *at all*. It turns out that, if one is willing to rely on time-lock puzzles [34], we can avoid the dependence from timelock scripts entirely. As it was shown in prior works [38, 37], absolute time-locks[5] can be simulated using verifiable timed signatures (VTS): VTS allow one to encapsulate a signature on a message for a pre-determined amount of time $\mathbf{T}$. At the same time, the party who is solving the puzzle, is guaranteed that the signature recovered after time $\mathbf{T}$ is a valid one. Parties are required to perform persistent background computation for the lifetime of the channel. However for currencies like Monero where we do not have any timelock script, we do not know of any other viable mechanism other than the one using VTS from [37]. A recent work [32] has enabled parties to securely outsource this computation to a decentralized network thereby removing any sort of computational load on the parties.

Extending lifetime and capacity of the channel. In contrast to Lightning Network channels, the channel Ch_{AB} between A and B is time bounded because of the bound required in Sleepy Channels. More precisely, parties have to close the channel Ch_{AB} before the timeout $\mathbf{T}$ that are set on the finish-payment transactions $tx^{A,A}_{\mathsf{Fpay},i}$ and $tx^{B,B}_{\mathsf{Fpay},i}$ that spend from $\mathsf{SleepyCh}_A$ and $\mathsf{SleepyCh}_B$, respectively. However, if both parties cooperate, they can easily extend their channel duration by transferring the coins from the current channel Ch_{AB} to a new channel Ch'_{AB} (shared between A and B) in accordance with the latest channel balance that the parties had in Ch_{AB}. In

[5]Crucially, this transformation does not work for the relative time-lock logic, since there the time depends on some event which is triggered by the attacker and thus one cannot set the time parameter of the VTS ahead of time.

Parties A and B have a payment channel Ch_{AB} with capacity $f+2c$ and secret key share for the channel are $sk^{A}_{\mathsf{Ch},AB}$ and $sk^{B}_{\mathsf{Ch},AB}$ for party A and B, respectively. Here f denotes the payment capacity of the channel and c is the collateral that a party allocates for the channel. Parties additionally have a refund transaction $tx_{\mathsf{rfnd}} := tx(\mathsf{Ch}_{AB}, [pk_A, pk_B], [v_A + c, v_B + c])$ and the corresponding signature σ_{rfnd} with respect to Ch_{AB}, where $v_A + v_B = f$ and pk_A and pk_B are some public keys of A and B, respectively.

Address Generation

1. Parties generate the following key pairs using $\Pi_{\mathsf{DS}}.\mathsf{KGen}(1^{\lambda})$
   - Party A generates $\left(pk_{\mathsf{CPay},A}, sk_{\mathsf{CPay},A}\right)$, $(pk_{\mathsf{pun},A}, sk_{\mathsf{pun},A})$, $(pk_{\mathsf{fp},A}, sk_{\mathsf{fp},A})$ and $(pk_{\mathsf{ffp},A}, sk_{\mathsf{ffp},A})$
   - Party B generates $\left(pk_{\mathsf{CPay},B}, sk_{\mathsf{CPay},B}\right)$, $(pk_{\mathsf{pun},B}, sk_{\mathsf{pun},B})$, $(pk_{\mathsf{fp},B}, sk_{\mathsf{fp},B})$ and $(pk_{\mathsf{ffp},B}, sk_{\mathsf{ffp},B})$
2. Parties run Γ_{JKGen} to generate shared addresses: $\mathsf{SleepyCh}_A, \mathsf{SleepyCh}_B, \mathsf{ExitCh}_A, \mathsf{ExitCh}_B, \mathsf{aux}_A, \mathsf{aux}_B$.

***i*-th Payment**

For the i-th payment where $v_{A,i}$ and $v_{B,i}$ are the balance of A and B, respectively with $f = v_{A,i} + v_{B,i}$, the parties do the following:

Payment Transactions: Generate payment transactions $tx^{A}_{\mathsf{Pay},i} := tx\left(\mathsf{Ch}_{AB}, [pk_{\mathsf{CPay},A}, \mathsf{SleepyCh}_A, \mathsf{ExitCh}_A], [c, v_{A,i}, v_{B,i} + c]\right)$ and $tx^{B}_{\mathsf{Pay},i} := tx\left(\mathsf{Ch}_{AB}, [pk_{\mathsf{CPay},B}, \mathsf{SleepyCh}_B, \mathsf{ExitCh}_B], [c, v_{B,i}, v_{A,i} + c]\right)$

Punishment Transactions: Generate $tx^{A}_{\mathsf{Pnsh},i} := tx\left(\mathsf{SleepyCh}_A, pk_{\mathsf{pun},B}, v_{A,i}\right)$ and $tx^{B}_{\mathsf{Pnsh},i} := tx\left(\mathsf{SleepyCh}_B, pk_{\mathsf{pun},A}, v_{B,i}\right)$

Finish-Payment Transactions:

1. Generate $tx^{A,A}_{\mathsf{Fpay},i} := tx\left(\mathsf{SleepyCh}_A, pk_{\mathsf{fp},A}, v_{A,i}\right)$ and $tx^{B,B}_{\mathsf{Fpay},i} := tx\left(\mathsf{SleepyCh}_B, pk_{\mathsf{fp},B}, v_{B,i}\right)$ both timelocked until time **T**.
2. Generate another set of faster finish-pay transactions $tx^{A,B}_{\mathsf{Fpay},i} := tx\left(\mathsf{ExitCh}_A, [pk_{\mathsf{ffp},B}, \mathsf{aux}_A], [v_{B,i} + c - \varepsilon, \varepsilon]\right)$ and $tx^{B,A}_{\mathsf{Fpay},i} := tx\left(\mathsf{ExitCh}_B, [pk_{\mathsf{ffp},A}, \mathsf{aux}_B], [v_{A,i} + c - \varepsilon, \varepsilon]\right)$.
3. Generate a set of enabler transactions $tx^{A*}_{\mathsf{Fpay},i} := tx\left([\mathsf{SleepyCh}_A, \mathsf{aux}_A], pk_{\mathsf{fp},A}, v_{A,i} + \varepsilon\right)$ and $tx^{B*}_{\mathsf{Fpay},i} := tx\left([\mathsf{SleepyCh}_B, \mathsf{aux}_B], pk_{\mathsf{fp},B}, v_{B,i} + \varepsilon\right)$ that enable a faster finish-payment.

Signature Generation: Parties generate signatures on transactions by running the interactive protocol Γ_{Sign} in each step. In case one of the party aborts at any step, the other party closes the channel with the $(i-1)$-th payment state.

1. Party A receives signature $\sigma^{A,A}_{\mathsf{Fpay},i}$ on transaction $tx^{A,A}_{\mathsf{Fpay},i}$ under the shared key $\mathsf{SleepyCh}_A$. Party B receives signature $\sigma^{B,B}_{\mathsf{Fpay},i}$ on transaction $tx^{B,B}_{\mathsf{Fpay},i}$ under the shared key $\mathsf{SleepyCh}_B$.
2. Party A receives signatures $\left(\sigma_{\mathsf{SleepyCh},A}, \sigma_{\mathsf{aux},A}\right)$ on the transaction $tx^{A*}_{\mathsf{Fpay},i}$ with respect to the shared keys $\mathsf{SleepyCh}_A$ and aux_A, respectively. Party B receives signatures $\left(\sigma_{\mathsf{SleepyCh},B}, \sigma_{\mathsf{aux},B}\right)$ on the transaction $tx^{B*}_{\mathsf{Fpay},i}$ with respect to the shared keys $\mathsf{SleepyCh}_B$ and aux_B, respectively.
3. Party A receives signature $\sigma^{B,A}_{\mathsf{Fpay},i}$ on the transaction $tx^{B,A}_{\mathsf{Fpay},i}$ under the shared key ExitCh_B. Party B receives signature $\sigma^{A,B}_{\mathsf{Fpay},i}$ on the transaction $tx^{A,B}_{\mathsf{Fpay},i}$ under the shared key ExitCh_A.
4. Party A receives signature $\sigma^{A}_{\mathsf{Pay},i}$ on the transaction $tx^{A}_{\mathsf{Pay},i}$ under the shared key Ch_{AB}. Party B receives signature $\sigma^{B}_{\mathsf{Pay},i}$ on the transaction $tx^{B}_{\mathsf{Pay},i}$ under the shared key Ch_{AB}.

Figure 4: Sleepy Channel protocol - Payment setup and payments

other words, parties can post a single transaction on the blockchain anytime before **T** to transfer the coins from Ch_{AB} to Ch'_{AB}. The channel balance of the parties in Ch'_{AB} is set according to the most recent payment state between them in the channel Ch_{AB}. Similar procedure is adopted in the Splicing protocol [35] of Lightning Network where users can periodically increase or decrease their channel capacity on-chain without violating any payments already made. Our Sleepy Channel protocol apart from extending the channel lifetime, can also update the channel capacity with this approach.

6 Performance Evaluation

We evaluated a proof of concept to (i) show correctness of our scheme, (ii) showcase compatibility with Bitcoin, and (iii) measure on- and off-chain transaction overhead. The source code is available at [19].

Implementation subtleties. There are several approaches on how Sleepy Channels can be implemented, given the scripting functionality of, say, Bitcoin. For instance, timelocks can be enforced either at a single transaction output or for the whole transaction, 2-party signing can be replaced with a multisig script (for a blow up in the transaction size) and revocation can be via exchang-

i-th Payment Revocation

To revoke the i-th payment, parties jointly generate signatures by running the interactive protocol Γ_{Sign}: Generate signature $\sigma^{A}_{\mathsf{Pnsh},i}$ on the punishment transaction $tx^{A}_{\mathsf{Pnsh},i}$ (party A receives $\sigma^{A}_{\mathsf{Pnsh},i}$ as output and gives it to B) and signature $\sigma^{B}_{\mathsf{Pnsh},i}$ on the punishment transaction $tx^{B}_{\mathsf{Pnsh},i}$ (party B receives $\sigma^{B}_{\mathsf{Pnsh},i}$ as output and gives it to A). If during the revocation either party aborts, the non-aborting party immediately closes the channel with the most recent unrevoked payment.

Channel Closing

Either party can close the channel Ch_{AB} with the j-th unrevoked payment. To do this:

1. Party A posts $\left(tx^{A}_{\mathsf{Pay},j}, \sigma^{A}_{\mathsf{Pay},j}\right)$ on $\mathbb{B}$. This is followed by one of the two cases:
   (a) Fast finish: Party B posts $\left(tx^{A,B}_{\mathsf{Fpay},i}, \sigma^{A,B}_{\mathsf{Fpay},i}\right)$ on $\mathbb{B}$, and party A posts $\left(tx^{A*}_{\mathsf{Fpay},i}, \sigma^{A*}_{\mathsf{Fpay},i}\right)$ on $\mathbb{B}$ for fast finish
   (b) Lazy finish: If not, A can post $\left(tx^{A,A}_{\mathsf{Fpay},i}, \sigma^{A,A}_{\mathsf{Fpay},i}\right)$ on $\mathbb{B}$ after timeout $\mathbf{T}$
2. Analogously, party B can post $\left(tx^{B}_{\mathsf{Pay},j}, \sigma^{B}_{\mathsf{Pay},j}\right)$ on $\mathbb{B}$. This is followed by one of the two cases:
   (a) Fast finish: Party A posts $\left(tx^{B,A}_{\mathsf{Fpay},i}, \sigma^{B,A}_{\mathsf{Fpay},i}\right)$ on $\mathbb{B}$, and party B posts $\left(tx^{B*}_{\mathsf{Fpay},i}, \sigma^{B*}_{\mathsf{Fpay},i}\right)$ on $\mathbb{B}$ for fast finish
   (b) Lazy finish: If not, B can post $\left(tx^{B,B}_{\mathsf{Fpay},i}, \sigma^{B,B}_{\mathsf{Fpay},i}\right)$ on $\mathbb{B}$ after timeout $\mathbf{T}$

Punishing Revoked payments

If A posts the j-th revoked payment $tx^{A}_{\mathsf{Pay},j}$ on $\mathbb{B}$, B can post the punishment transaction $\left(tx^{A}_{\mathsf{Pnsh},i}, \sigma^{A}_{\mathsf{Pnsh},i}\right)$ on $\mathbb{B}$ before the absolute timeout $\mathbf{T}$. If B posts the j-th revoked payment $tx^{B}_{\mathsf{Pay},j}$ on $\mathbb{B}$, A can post $\left(tx^{B}_{\mathsf{Pnsh},i}, \sigma^{B}_{\mathsf{Pnsh},i}\right)$ on $\mathbb{B}$ before the absolute timeout $\mathbf{T}$.

Figure 5: Sleepy Channel protocol - Channel closing and punishment

ing a hash secret, a private key or a signed punishment transaction upon revoking an old state).In this section, we follow our protocol description from Figures 4 and 5 and use transaction level timelocks, 2-party signing and exchange signed punishment transactions for revocation.

Deploying the transactions. Now we describe the transactions used in Sleepy Channels and we refer the reader to Table 2 in Appendix B for the details on transaction sizes and their cost in terms of on-chain fees. We also give a pointer to the corresponding transactions deployed in the Bitcoin testnet, thereby demonstrating the backwards compatibility of Sleepy Channels.

The first step in Sleepy Channels is building a funding transaction tx_F [40]. Built on top of the funding, we look at A's commitment (or state) transaction $tx^{A}_{\mathsf{Pay},i}$ [41] and note that the transactions for B are symmetric. When A puts the current state on the ledger, there are two ways how A can claim its money. On the one hand, if B unlocks its own funds by putting $tx^{A,B}_{\mathsf{Fpay},i}$ [43], then A can claim its funds with $tx^{A*}_{\mathsf{Fpay},i}$ right away [42]. On the other hand, after the lifetime expires, A can unilaterally claim its funds with $tx^{A,A}_{\mathsf{Fpay},i}$. If A puts an old state, then B can punish A via $tx^{A}_{\mathsf{Pnsh},i}$. Finally, two users can close their channel honestly with a transaction, where both funds are unlocked right away.

We find that for opening a channel in Sleepy Channels, the two parties together need to put 338 bytes on-chain and exchange 2026 bytes (8 transactions off-chain). For each subsequent updates, the two parties need to exchange 2408 bytes (10 transactions off-chain). The closing and punishment happen on-chain. For the closing there are three options. Either they close honestly (225 bytes, 1 tx), or one party closes unilaterally and unlocks its funds after the timelock expires (449 bytes, 2 tx), or one party closes unilaterally and the other one unlocks the funds right away (823 bytes, 3 tx). The punishment case requires 450 bytes and 2 transactions.

Comparison to LN. As for our construction, the LN channel functionality can be implemented with subtle differences, resulting in different outcomes. The funding transaction of LN is identical to ours, except that it locks no additional collateral. The commitment transactions differ, as they have one fewer output, and therefore only 226 bytes. Moreover, in LN there are no fast finish transactions. This totals to 338 bytes on-chain and exchanging 832 bytes (4 transactions) for opening a LN channel. For updating, the users exchange 1214 bytes (6 transactions). Note that the honest, the unilateral close and the punishment in sleepy channels is identical to LN, both in terms of transaction structure and in size.

Overhead. The Sleepy Channels protocol does not require costly cryptography. It requires computing and verifying signatures locally, 2-party signing and a maximum off-chain communication in the order of 10^3 bytes for

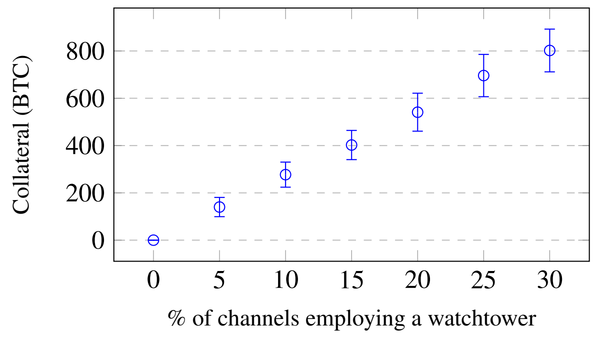

Figure 6: Results of the first simulation.

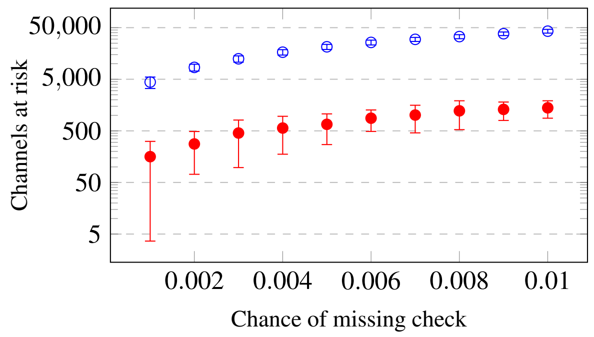

Figure 7: Results of the second simulation. (Blue = LN, Red = Sleepy Channels)

each operation. The computational time can be expected to be negligible on even commodity hardware; the communication is limited only by network latency.

6.1 Simulation

We perform some additional experiments with respect to a recent snapshot of LN. In this snapshot, there are 71k channels, 16k channel nodes and a total capacity of 2712 BTC. As the balance distribution of each channel is unknown, we assume that it is split evenly between the two users. The source code of our simulation experiments is available at [20]. We repeat the experiments 100 times for each and plot the average and standard deviation.

Watchtower collateral. We investigate the collateral a watchtower service needs to provide, in order to cover their customers should they go offline. For this, we randomly sample a percentage of nodes that wish to employ a watchtower and based on their balances in their channels, we plot the amount of collateral in Figure 6. This amount rises linearly with the amount of users that wish to employ a watchtower. If 30% of all users do so, (i) the watchtower service needs to lock up approximately 800 BTC and (ii) users needs to pay fees for that, even if there are no disputes.

Risk of failing to go online. We simulate the risk of users having to periodically monitor the blockchain in LN. In LN, there is a certain time interval, e.g., once a day, when users need to come online and check whether or not the other party tried to cheat. In our setting, we investigate a time frame of 30 days, i.e., users come online 30 times.

In our simulation, we assume that there is a certain chance that users fail to come online and monitor the blockchain in time. We further assume that neighboring nodes will notice this; a realistic assumption due to the *ping and pong* messages [24] of the LN. We assume that neighboring nodes want to maximize their profits and will exploit such a case by putting an old state and thereby, potentially stealing funds of the offline user.

The Sleepy Channels protocol would not fully prevent this behavior, but reduce it significantly. That is, for a given period of time, in this simulation 30 days, the users need to come online only once, i.e., before the channel expires. They can of course fail to come online there with the same probability, but this event occurs only once instead of 30 times. Obviously, the longer this time span is, the greater the chances for LN nodes is to miss at least one of these intervals, while for Sleepy Channels it remains the same. For 30 days, only about 3% of the channels are at risk for Sleepy Channels compared to LN, for any given chance of missing the online check.

In Figure 7 we plot the number of channels that are at risk for a given probability that a user will fail to come online in each interval, once for each the LN and Sleepy Channels. The y axis is shown in logarithmic scale.

7 Conclusion

Payment channels are one of the most promising payment solutions for blockchain-based cryptocurrencies. Despite their large adoption, many such proposal suffer from limitations, such as requiring the parties to be constantly online and monitor the network, or outsourcing this task to third parties (e.g., watchtowers). In this work, we propose a new payment channel architecture (Sleepy Channels) that supports bi-directional payments and does not require the parties to be persistently online. The protocol is backward compatible with many existing currencies (e.g., Bitcoin, Monero...) and relies on lightweight cryptographic machinery. Our performance evaluation shows that the protocol is efficient enough to be adopted in a large payment ecosystems (such as the Lightning Network). An interesting open question is whether our techniques are also applicable to account-based currencies, rather than UTXO-based currencies.

Acknowledgements

The work was partially supported by the Deutsche Forschungsgemeinschaft (DFG – German Research Foundation) under 442893093, by the state of Bavaria at the Nuremberg Campus of Technology (NCT), by CoBloX Labs, by the European Research Council (ERC) under the European Union's Horizon 2020 research (grant agreement 771527-BROWSEC), by the Austrian Science Fund (FWF) through the projects PROFET (grant agreement P31621) and the project W1255-N23, by the Austrian Research Promotion Agency (FFG) through the Bridge-1 project PR4DLT (grant agreement 13808694), the COMET K1 SBA and COMET K1 ABC, by the Vienna Business Agency through the project Vienna Cybersecurity and Privacy Research Center (VISP), by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association through the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things (CDL-BOT).

References

[1] Lukas Aumayr et al. "Generalized Channels from Limited Blockchain Scripts and Adaptor Signatures". In: *AsiaCrypt* (2021). URL: https://eprint.iacr.org/2020/476.

[2] Georgia Avarikioti, Eleftherios Kokoris-Kogias, and Roger Wattenhofer. "Brick: Asynchronous State Channels". In: *CoRR* abs/1905.11360 (2019). arXiv: 1905.11360. URL: http://arxiv.org/abs/1905.11360.

[3] Georgia Avarikioti et al. *Towards Secure and Efficient Payment Channels*. 2018. arXiv: 1811.12740 [cs.CR].

[4] Zeta Avarikioti, Orfeas Stefanos Thyfronitis Litos, and Roger Wattenhofer. "Cerberus Channels: Incentivizing Watchtowers for Bitcoin". In: *FC 2020*. Ed. by Joseph Bonneau and Nadia Heninger. Vol. 12059. LNCS. Springer, Heidelberg, Feb. 2020, pp. 346–366. DOI: 10.1007/978-3-030-51280-4_19.

[5] Michael Backes and Dennis Hofheinz. "How to Break and Repair a Universally Composable Signature Functionality". In: *ISC 2004*. Ed. by Kan Zhang and Yuliang Zheng. Vol. 3225. LNCS. Springer, Heidelberg, Sept. 2004, pp. 61–72.

[6] Dan Boneh, Manu Drijvers, and Gregory Neven. "Compact Multi-signatures for Smaller Blockchains". In: *ASIACRYPT 2018, Part II*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11273. LNCS. Springer, Heidelberg, Dec. 2018, pp. 435–464. DOI: 10.1007/978-3-030-03329-3_15.

[7] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. LNCS. Springer, Heidelberg, Dec. 2001, pp. 514–532. DOI: 10.1007/3-540-45682-1_30.

[8] Ran Canetti. "Security and Composition of Multiparty Cryptographic Protocols". In: *Journal of Cryptology* 13.1 (Jan. 2000), pp. 143–202. DOI: 10.1007/s001459910006.

[9] Ran Canetti et al. "Universally Composable Security with Global Setup". In: *TCC 2007*. Ed. by Salil P. Vadhan. Vol. 4392. LNCS. Springer, Heidelberg, Feb. 2007, pp. 61–85. DOI: 10.1007/978-3-540-70936-7_4.

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

[11] Guoxing Chen et al. "SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution". In: *2019 IEEE European Symposium on Security and Privacy (EuroS P)*. 2019, pp. 142–157. DOI: 10.1109/EuroSP.2019.00020.

[12] *Chia Network FAQ*. https://www.chia.net/faq/.

[13] Christian Decker and Rusty Russell. *eltoo: A Simple Layer2 Protocol for Bitcoin*. https://blockstream.com/eltoo.pdf.

[14] Christian Decker and Roger Wattenhofer. "A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels". In: *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings*. Ed. by Andrzej Pelc and Alexander A. Schwarzmann. Vol. 9212. Lecture Notes in Computer Science. Springer, 2015, pp. 3–18. DOI: 10.1007/978-3-319-21741-3_1. URL: https://doi.org/10.1007/978-3-319-21741-3_1.

[15] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. "General State Channel Networks". In: *ACM CCS 2018*. Ed. by David Lie et al. ACM Press, Oct. 2018, pp. 949–966. DOI: 10.1145/3243734.3243856.

[16] Stefan Dziembowski et al. "Perun: Virtual Payment Hubs over Cryptocurrencies". In: *2019 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2019, pp. 106–123. DOI: 10.1109/SP.2019.00020.

[17] Andreas Erwig et al. "Two-Party Adaptor Signatures from Identification Schemes". In: *PKC 2021, Part I*. Ed. by Juan Garay. Vol. 12710. LNCS. Springer, Heidelberg, May 2021, pp. 451–480. DOI: 10.1007/978-3-030-75245-3_17.

[18] Rosario Gennaro et al. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. LNCS. Springer, Heidelberg, May 1999, pp. 295–310. DOI: 10.1007/3-540-48910-X_21.

[19] *Github repository of our Sleepy Channels evaluation*. https://github.com/sleepy-channels/overhead.

[20] *Github repository of our Sleepy Channels simulation*. https://github.com/sleepy-channels/simulation.

[21] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. "A Digital Signature Scheme Secure Against Adaptive Chosen-message Attacks". In: *SIAM Journal on Computing* 17.2 (Apr. 1988), pp. 281–308.

[22] Majid Khabbazian, Tejaswi Nadahalli, and Roger Wattenhofer. "Outpost: A Responsive Lightweight Watchtower". In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. AFT '19. Zurich, Switzerland: Association for Computing Machinery, 2019, 31–40. ISBN: 9781450367325. DOI: 10.1145/3318041.3355464. URL: https://doi.org/10.1145/3318041.3355464.

[23] *Lightning Network*. https://lightning.network/.

[24] *Lightning Network specification, BOLT #1: Base Protocol, ping and pong messages*. https://github.com/lightningnetwork/lightning-rfc/blob/master/01-messaging.md#the-ping-and-pong-messages.

[25] Joshua Lind et al. "Teechan: Payment Channels Using Trusted Execution Environments". In: *CoRR* abs/1612.07766 (2016). arXiv: 1612.07766. URL: http://arxiv.org/abs/1612.07766.

[26] Yehuda Lindell. "Fast Secure Two-Party ECDSA Signing". In: *CRYPTO 2017, Part II*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10402. LNCS. Springer, Heidelberg, Aug. 2017, pp. 613–644. DOI: 10.1007/978-3-319-63715-0_21.

[27] Giulio Malavolta et al. "Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability". In: *NDSS 2019*. The Internet Society, Feb. 2019.

[28] Giulio Malavolta et al. "Concurrency and Privacy with Payment-Channel Networks". In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham et al. ACM Press, 2017, pp. 455–471. DOI: 10.1145/3133956.3134096.

[29] Patrick McCorry et al. "Pisa: Arbitration Outsourcing for State Channels". In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. AFT '19. Zurich, Switzerland: Association for Computing Machinery, 2019, 16–30. ISBN: 9781450367325. DOI: 10.1145/3318041.3355461. URL: https://doi.org/10.1145/3318041.3355461.

[30] Arash Mirzaei et al. *FPPW: A Fair and Privacy Preserving Watchtower For Bitcoin*. Cryptology ePrint Archive, Report 2021/117. https://ia.cr/2021/117. 2021.

[31] Pedro Moreno-Sanchez et al. "DLSAG: Non-interactive Refund Transactions for Interoperable Payment Channels in Monero". In: *Financial Cryptography and Data Security*. Ed. by Joseph Bonneau and Nadia Heninger. Cham: Springer International Publishing, 2020, pp. 325–345.

[32] "Personal Communication". In: (). To Appear at ACM CCS 2021.

[33] Joseph Poon and Thaddeus Dryja. *The bitcoin lightning network: Scalable off-chain instant payments*. 2016.

[34] R. L. Rivest, A. Shamir, and D. A. Wagner. *Time-lock Puzzles and Timed-release Crypto*. Cambridge, MA, USA, 1996.

[35] Rusty Russell. *[Lightning-dev] Splicing Proposal*. https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-October/001434.html. 2018.

[36] Jeremy Spillman. *Spillman-style payment channels*. https://tinyurl.com/uwzfb2tu.

[37] Sri Aravinda Krishnan Thyagarajan et al. *PayMo: Payment Channels For Monero*. Cryptology ePrint Archive, Report 2020/1441. https://eprint.iacr.org/2020/1441. 2020.

[38] Sri Aravinda Krishnan Thyagarajan et al. "Verifiable Timed Signatures Made Practical". In: *ACM CCS 2020*. Ed. by Jay Ligatti et al. ACM Press, Nov. 2020, pp. 1733–1750. DOI: 10.1145/3372297.3417263.

[39] Peter Todd. *CLTV-style payment channels*. https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki#Payment_Channels.

[40] *tx_F of our evaluation on the Bitcoin testnet*. https://tinyurl.com/589xku8w.

[41] *$tx^A_{\mathsf{Pay},i}$ of our evaluation on the Bitcoin testnet*. https://tinyurl.com/2w6aebr9.

[42] *$tx^{A*}_{\mathsf{Fpay},i}$ of our evaluation on the Bitcoin testnet*. https://tinyurl.com/bskz7fvx.

[43] *$tx^{A,B}_{\mathsf{Fpay},i}$ of our evaluation on the Bitcoin testnet*. https://tinyurl.com/2uwn5fvb.

[44] *Unlinkable Outsourced Channel Monitoring*. https://diyhpl.us/wiki/transcripts/scalingbitcoin/milan/unlinkable-outsourced-channel-monitoring/.

[45] Jo Van Bulck et al. "A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, 1741–1758. ISBN: 9781450367479. DOI: 10.1145/3319535.3363206. URL: https://doi.org/10.1145/3319535.3363206.

A UC Protocol

Using the notation introduced in Section 4, we here give a formal version of the protocol that is augmented in a way to model it in the UC framework. More specifically, we model the environment to capture anything that happens outside of the protocol execution as well as communication model. Additionally, we replace (i) the 2-party key generation protocol Γ_{JKGen} for a signature scheme Π_{DS} with an idealized version $\mathcal{F}_{\mathsf{JKGen}}$ and (ii) the 2-party signing protocol Γ_{Sign} for a signature scheme with an idealized version $\mathcal{F}_{\mathsf{Sign}}$. Finally, we add the possibility to honestly close payment channels in a way that requires only one on-chain transaction, i.e., by creating a transaction spending from the funding transaction and giving each user their respective balance right away.

In order to improve the readability of the protocol, we exclude checks that an honest user would naturally perform, such as that parameters given from the environment are well-formed, there is an input of the fund belonging to each of the two users holding the right amount of coins, verifying that channels to be updated or closed exist, the new state is valid or that a channel to be updated or closed is not currently being updated or closed. This can be formally handled by using a protocol wrapper, that performs these checks on the messages from the environment and drops invalid ones. We refer to [1], where such a wrapper for payment channels is formally defined and use the same in this work. Similarly, for the ideal functionality we use such a wrapper as well.

Sleepy channel protocol Π

<u>Create</u>

<u>Party A upon $(\texttt{CREATE}, \mathsf{id}, \gamma, tid_A) \overset{t_0}{\hookleftarrow} \mathcal{E}$:</u>

1. Generate $(pk_{\mathsf{CPay},A}, sk_{\mathsf{CPay},A}), (pk_{\mathsf{pun},A}, sk_{\mathsf{pun},A})$, $(pk_{\mathsf{fp},A}, sk_{\mathsf{fp},A})$ and $(pk_{\mathsf{ffp},A}, sk_{\mathsf{ffp},A})$. Let $pkey^A_{set}$ be the set of public keys of these key pairs.
2. Extract $v_{A,0}$ and $v_{B,0}$ from $\gamma.\mathsf{st}$, and $c := \gamma.c$
3. Send $(\text{createInfo}, \mathsf{id}, tid_A, pkey^A_{set}) \overset{t_0}{\hookrightarrow} B$.
4. If $(\text{createInfo}, \mathsf{id}, tid_B, pkey^B_{set}) \xleftarrow{t_0+1} B$, continue. Else, go idle.
5. Using $pkey^A_{set}$ and $pkey^B_{set}$, A together with B runs $\mathcal{F}_{\mathsf{JKGen}}$ to generate the following set of shared addresses: $addr_{set} := \{\mathsf{Ch}_{AB}, \mathsf{SleepyCh}_A, \mathsf{SleepyCh}_B, \mathsf{ExitCh}_A, \mathsf{ExitCh}_B, \mathsf{aux}_A, \mathsf{aux}_B\}$ which takes t_g rounds. In case of failure, abort.
6. Generate $tx_f := tx([tid_A, tid_B], [\mathsf{Ch}_{AB}], [2 \cdot c + v_{A,0} + v_{B,0}])$
7. Let $tx_{set0} \leftarrow \texttt{GenerateTxs}(addr_{set}, pkey^A_{set}, pkey^B_{set}, c, v_{A_i}, v_{B_i})$
8. Let $sig^A_{set0} \leftarrow \texttt{SignTxs}^A(tx_{set0}, addr_{set}, pkey^A_{set} \cup pkey^B_{set})$
9. A generates a signature σ_{tid_A} for the output tid_A and sends $(\text{createFund}, \mathsf{id}, \sigma_{tid_A}) \xhookrightarrow{t_0+1+t_g+t_s} A$.
10. If $(\text{createFund}, \mathsf{id}, \sigma_{tid_B}) \xleftarrow{t_0+2+t_g+t_s} B$, post $(tx_F, \{\sigma_{tid_A}, \sigma_{tid_B}\})$ to $\mathbb{B}$.
11. If tx_F is accepted by $\mathbb{B}$ in round $t_1 \leq t_0 + 2 + t_g + t_s + \Delta$, store $\Gamma^A(\mathsf{id}) := (tx_F, tx_{set0}, sig^A_{set0}, addr_{set}, pkey^A_{set}, pkey^B_{set})$ and $(\texttt{CREATED}, \mathsf{id}) \overset{t_1}{\hookrightarrow} \mathcal{E}$.

<u>Update</u>

<u>Party A upon $(\texttt{UPDATE}, \mathsf{id}, \overrightarrow{\theta}, t_{\mathsf{stp}}) \overset{t_0}{\hookleftarrow} \mathcal{E}$</u>

1. $(\text{updateReq}, \mathsf{id}, \overrightarrow{\theta}, t_{\mathsf{stp}}) \overset{t_0}{\hookrightarrow} B$

Party B upon $(\text{updateReq}, \mathsf{id}, \overrightarrow{\theta}, t_{\mathsf{stp}}) \xhookleftarrow{\tau_0} A$

1. Retrieve $(tx_F, tx_{set_{i-1}}, sig_{set_{i-1}}^B, addr_{set}, pkey_{set}^A, pkey_{set}^B) = \Gamma^B(\mathsf{id})$
2. Extract $v_{A,i}$ and $v_{B,i}$ from $\overrightarrow{\theta}$, and c from tx_F
3. Let $tx_{set_i} \leftarrow \texttt{GenerateTxs}(addr_{set}, pkey_{set}^A, pkey_{set}^B, c, v_{A_i}, v_{B_i})$
4. Let $\overrightarrow{tid} := (tx^A_{\mathsf{Pay},i}.\mathsf{id}, tx^B_{\mathsf{Pay},i}.\mathsf{id})$ be a tuple of the transaction ids of transaction $tx^A_{\mathsf{Pay},i}$ and $tx^B_{\mathsf{Pay},i}$.
5. $(\texttt{UPDATE-REQ}, \mathsf{id}, \overrightarrow{\theta}, t_{\mathsf{stp}}, \overrightarrow{tid}) \xhookrightarrow{\tau_0} \mathcal{E}$
6. $(\text{updateInfo}, \mathsf{id}) \xhookrightarrow{\tau_0} A$

Party A upon $(\text{updateInfo}, \mathsf{id}) \xhookleftarrow{t_0+2} B$

1. Retrieve $(tx_F, tx_{set_{i-1}}, sig_{set_{i-1}}^A, addr_{set}, pkey_{set}^A, pkey_{set}^B) = \Gamma^A(\mathsf{id})$
2. Extract $v_{A,i}$ and $v_{B,i}$ from $\overrightarrow{\theta}$, and c from tx_F
3. Let $tx_{set_i} \leftarrow \texttt{GenerateTxs}(addr_{set}, pkey_{set}^A, pkey_{set}^B, c, v_{A_i}, v_{B_i})$
4. Let $\overrightarrow{tid} := (tx^A_{\mathsf{Pay},i}.\mathsf{id}, tx^B_{\mathsf{Pay},i}.\mathsf{id})$ be a tuple of the transaction ids of transaction $tx^A_{\mathsf{Pay},i}$ and $tx^B_{\mathsf{Pay},i}$.
5. $(\texttt{SETUP}, \mathsf{id}, \overrightarrow{tid}) \xhookrightarrow{t_0+2} \mathcal{E}$
6. If $(\texttt{SETUP-OK}, \mathsf{id}) \xhookleftarrow{t_1 \leq t_0+2+t_{\mathsf{stp}}} \mathcal{E}$, send $(\text{updateCom}, \mathsf{id}) \xhookrightarrow{t_1} B$
7. Wait one round.
8. $\texttt{SignTxs}^A(tx_{set\,i}, addr_{set}, pkey_{set}^A \cup pkey_{set}^B)$

Party B upon $(\text{updateCom}, \mathsf{id}) \xhookleftarrow{\tau_1 \leq \tau_0+2+t_{\mathsf{stp}}} A$

9. $(\texttt{SETUP-OK}, \mathsf{id}) \xhookrightarrow{\tau_1} \mathcal{E}$
10. If not $(\texttt{UPDATE-OK}, \mathsf{id}) \xhookleftarrow{\tau_1} \mathcal{E}$, go idle.
11. $\texttt{SignTxs}^A(tx_{set\,i}, addr_{set}, pkey_{set}^A \cup pkey_{set}^B)$

Party A in round $t_1 + 1 + t_s$

12. If $sig_{set_i}^A$ is returned from $\texttt{SignTxs}^A$, $(\texttt{UPDATE-OK}, \mathsf{id}) \xhookrightarrow{t_1+1+t_s} \mathcal{E}$. Else, execute $\texttt{ForceClose}(\mathsf{id})$ and go idle.
13. If not $(\texttt{REVOKE}, \mathsf{id}) \xhookleftarrow{t_1+1+t_s} \mathcal{E}$, go idle.
14. A together with B runs the interactive protocol $\mathcal{F}_{\mathsf{Sign}}$ to generate the following signature. $\sigma^A_{\mathsf{Pnsh},i}$ on the punishment transaction $tx^A_{\mathsf{Pnsh},i}$.Party A receives $\sigma^A_{\mathsf{Pnsh},i}$ as output after t_r. In case of failure, execute $\texttt{ForceClose}(\mathsf{id})$.
15. $(\mathsf{revoke}, \mathsf{id}, \sigma^A_{\mathsf{Pnsh},i}) \xhookrightarrow{t_1+1+t_s+t_r} B$

Party B in round $\tau_1 + t_s$

16. If $sig_{set_i}^B$ is not returned from $\texttt{SignTxs}^A$, execute $\texttt{ForceClose}(\mathsf{id})$ and go idle.
17. Participate in the signing of $tx^A_{\mathsf{Pnsh},i}$.
18. Upon $(\mathsf{revoke}, \mathsf{id}, \sigma^A_{\mathsf{Pnsh},i}) \xhookleftarrow{\tau_1+1+t_s+t_r} A$, continue. Else, execute $\texttt{ForceClose}(\mathsf{id})$ and go idle.
19. $(\texttt{REVOKE-REQ}, \mathsf{id}) \xhookrightarrow{\tau_1+1+t_s+t_r} \mathcal{E}$
20. If not $(\texttt{REVOKE}, \mathsf{id}) \xhookleftarrow{\tau_1+1+t_s+t_r} \mathcal{E}$, go idle.
21. B together with A runs the interactive protocol $\mathcal{F}_{\mathsf{Sign}}$ to generate the following signature. $\sigma^B_{\mathsf{Pnsh},i}$ on the punishment transaction $tx^B_{\mathsf{Pnsh},i}$. Party B receives $\sigma^B_{\mathsf{Pnsh},i}$ as output after t_r. In case of failure, execute $\texttt{ForceClose}(\mathsf{id})$.
22. $(\mathsf{revoke}, \mathsf{id}, \sigma^B_{\mathsf{Pnsh},i}) \xhookrightarrow{\tau_1+1+t_s+2t_r} A$
23. $\Theta^B(\mathsf{id}) := \Theta^B \cup \left\{ (tx_{set\,i-1}, sig_{set_{i-1}}^B, \sigma^B_{\mathsf{Pnsh},i-1}) \right\}$
24. $\Gamma^B(\mathsf{id}) := (tx_F, tx_{set\,i}, sig_{set_i}^B, addr_{set}, pkey_{set}^A, pkey_{set}^B)$
25. $(\texttt{UPDATED}, \mathsf{id}) \xhookrightarrow{\tau_1+2+t_s+2t_r} \mathcal{E}$

Party A in round $t_1 + 2 + t_s + t_r$

26. Participate in the signing of $tx^B_{\mathsf{Pnsh},i}$.
27. If $(\mathsf{revoke}, \mathsf{id}, \sigma^B_{\mathsf{Pnsh},i}) \xhookleftarrow{t_1+3+t_s+2t_r} B$ and the signature is valid, go to next step. Else, execute $\texttt{ForceClose}(\mathsf{id})$.
28. $\Theta^A(\mathsf{id}) := \Theta^A \cup \left\{ (tx_{set_{i-1}}, sig_{set_{i-1}}^A, \sigma^B_{\mathsf{Pnsh},i-1}) \right\}$
29. $\Gamma^A(\mathsf{id}) := (tx_F, tx_{set\,i}, sig_{set_i}^A, addr_{set}, pkey_{set}^A, pkey_{set}^B)$
30. $(\texttt{UPDATED}, \mathsf{id}) \xhookrightarrow{t_1+3+t_s+2t_r} \mathcal{E}$

Close

Party A upon $(\texttt{CLOSE}, \mathsf{id}) \xhookleftarrow{t_0} \mathcal{E}$

1. Extract $(tx_F, tx_{set\,i}, sig_{set_i}^A, addr_{set}, pkey_{set}^A, pkey_{set}^B)$ from $\Gamma^A(\mathsf{id})$.
2. Extract $v_{A,i}$ and $v_{B,i}$ from $tx^A_{\mathsf{Pay},j} \in tx_{set\,i}$, and c from tx_F
3. Create transaction $tx_c := tx(\mathsf{Ch}_{AB}, \{pk_A, pk_B\}, \{v_{A,i}+c, v_{B,i}+c\})$, where pk_A is an address controlled by A and pk_B an address controlled by B.
4. A together with B runs the interactive protocol $\mathcal{F}_{\mathsf{Sign}}$ to generate the following signature, σ_{tx_c} on the transaction tx_c. This takes t_r rounds.
5. In case the signature generation was successful, post (tx_c, σ_{tx_c}) on $\mathbb{B}$. Else, execute $\texttt{ForceClose}(\mathsf{id})$.
6. If tx_c appears on $\mathbb{B}$ in round $t_1 \leq t_0 + t_r + \Delta$, set $\Theta^A(\mathsf{id}) := \perp$, $\Gamma^A(\mathsf{id}) := \perp$ and send $(\texttt{CLOSED}, \mathsf{id}) \xhookrightarrow{t_2} \mathcal{E}$.

Punish

Party A upon $\texttt{PUNISH} \xhookleftarrow{t_0} \mathcal{E}$:

For each $\mathsf{id} \in \{0,1\}^*$ s.t. $\Theta^P(\mathsf{id}) \neq \perp$:

1. Iterate over all elements $(tx_{set\,i}, sig_{set\,i}^{A}, \sigma^{B}_{\mathsf{Pnsh},i})$ in $\Theta^{P}(\mathsf{id})$
2. If the revoked payment $tx^{B}_{\mathsf{Pay},i} \in tx_{set\,i}$ is on $\mathbb{B}$, post $\left(tx^{B}_{\mathsf{Pnsh},i}, \sigma^{B}_{\mathsf{Pnsh},i}\right)$ on $\mathbb{B}$ before the absolute timeout $\mathbf{T}$.
3. Let $tx^{B}_{\mathsf{Pnsh},i}$ be accepted by $\mathbb{B}$ in round $t_1 \le t_0 + \Delta$. Post $(tx^{B,A}_{\mathsf{Fpay},i}, \sigma_{tx^{B,A}_{\mathsf{Fpay},i}} \in sig_{set\,i}^{A})$
4. After $tx^{B,A}_{\mathsf{Fpay},i}$ is accepted by $\mathbb{B}$ in round $t_2 \le t_1 + \Delta$, set $\Theta^{A}(\mathsf{id}) := \perp$, $\Gamma^{A}(\mathsf{id}) := \perp$ and output $(\mathtt{PUNISHED}, \mathsf{id}) \overset{t_1}{\hookrightarrow} \mathcal{E}$.

Subprotocols

`ForceClose`(id):
Let t_0 be the current round

1. Extract $(tx_F, tx_{set\,0}, sig_{set\,0}^{A}, addr_{set}, pkey^{A}_{set}, pkey^{B}_{set})$ from $\Gamma^{A}(\mathsf{id})$ and extract $tx^{A}_{\mathsf{Pay},j}$ from tx_{set} and $\sigma^{A}_{\mathsf{Pay},j}$ and sig_{set}.
2. Party A posts $\left(tx^{A}_{\mathsf{Pay},j}, \sigma^{A}_{\mathsf{Pay},j}\right)$ on $\mathbb{B}$
3. Let $t_1 \le t_0 + \Delta$ be the round in which $tx^{A}_{\mathsf{Pay},j}$ is accepted by $\mathbb{B}$.
4. If $tx^{A,B}_{\mathsf{Fpay},i}$ appears on $\mathbb{B}$ at or after round $t_2 \le t_1 + \Delta$ and before $\mathbf{T}$, post $\left(tx^{A}_{\mathsf{Pay},j}, \sigma^{A}_{\mathsf{Pay},j}\right)$ and send $(\mathtt{CLOSED}, \mathsf{id}) \xhookrightarrow{t_3 \le t_2 + \Delta} \mathcal{E}$. Otherwise, post $\left(tx^{A,A}_{\mathsf{Fpay},i}, \sigma^{A,A}_{\mathsf{Fpay},i}\right)$ after $\mathbf{T}$ and send $(\mathtt{CLOSED}, \mathsf{id}) \xhookrightarrow{t_4 \le \mathbf{T} + \Delta} \mathcal{E}$.
5. Set $\Gamma^{P}(\mathsf{id}) := \perp$, $\Theta^{P}(\mathsf{id}) := \perp$.

`GenerateTxs`$(addr_{set}, pkey^{A}_{set}, pkey^{B}_{set}, c, v_{A_i}, v_{B_i})$:

1. Using the addresses in $addr_{set}$ and the public keys in $pkey^{A}_{set}$ and $pkey^{B}_{set}$, do the following.
2. Generate $tx^{A}_{\mathsf{Pay},i} := tx(\mathsf{Ch}_{AB}, [pk_{\mathsf{CPay},A}, \mathsf{SleepyCh}_{A}, \mathsf{ExitCh}_{B}], [c, v_{A,i}, v_{B,i} + c])$
3. Generate $tx^{B}_{\mathsf{Pay},i} := tx(\mathsf{Ch}_{AB}, [pk_{\mathsf{CPay},B}, \mathsf{SleepyCh}_{B}, \mathsf{ExitCh}_{A}], [c, v_{B,i}, v_{A,i} + c])$
4. Generate punishment transactions $tx^{A}_{\mathsf{Pnsh},i} := tx(\mathsf{SleepyCh}_{A}, pk_{\mathsf{pun},B}, v_{A,i})$ and $tx^{B}_{\mathsf{Pnsh},i} := tx(\mathsf{SleepyCh}_{B}, pk_{\mathsf{pun},A}, v_{B,i})$
5. Generate finish-pay transactions $tx^{A,A}_{\mathsf{Fpay},i} := tx(\mathsf{SleepyCh}_{A}, pk_{\mathsf{fp},A}, v_{A,i})$ and $tx^{B,B}_{\mathsf{Fpay},i} := tx(\mathsf{SleepyCh}_{B}, pk_{\mathsf{fp},B}, v_{B,i})$ both timelocked until time $\mathbf{T}$.
6. Generate a set of faster finish-pay transactions $tx^{A,B}_{\mathsf{Fpay},i} := tx(\mathsf{ExitCh}_{A}, [pk_{\mathsf{ffp},B}, \mathsf{aux}_{A}], [v_{B,i} + c - \varepsilon, \varepsilon])$ and $tx^{B,A}_{\mathsf{Fpay},i} := tx(\mathsf{ExitCh}_{B}, [pk_{\mathsf{ffp},A}, \mathsf{aux}_{B}], [v_{A,i} + c - \varepsilon, \varepsilon])$.
7. Generate a set of enabler transactions $tx^{A*}_{\mathsf{Fpay},i} := tx([\mathsf{SleepyCh}_{A}, \mathsf{aux}_{A}], pk_{\mathsf{fp},A}, v_{A,i} + \varepsilon)$ and $tx^{B*}_{\mathsf{Fpay},i} := tx([\mathsf{SleepyCh}_{B}, \mathsf{aux}_{B}], pk_{\mathsf{fp},B}, v_{B,i} + \varepsilon)$ that enable a faster finish-payment.
8. Return $tx_{set} := \{tx^{A}_{\mathsf{Pay},i}, tx^{B}_{\mathsf{Pay},i}, tx^{A}_{\mathsf{Pay},i}, tx^{A}_{\mathsf{Pnsh},i}, tx^{B}_{\mathsf{Pnsh},i}, tx^{A,A}_{\mathsf{Fpay},i}, tx^{B,B}_{\mathsf{Fpay},i}, tx^{A,B}_{\mathsf{Fpay},i}, tx^{B,A}_{\mathsf{Fpay},i}, tx^{A*}_{\mathsf{Fpay},i}, tx^{B*}_{\mathsf{Fpay},i}\}$

`SignTxs`$^{A}(tx_{set}, addr_{set}, pkey^{A}_{set} \cup pkey^{B}_{set})$:
Party A (specified by the superscript of the function) is the one that receives the signatures first.
Upon agreement, i.e., A and B start executing this subprotocol in the same round with the same parameters, the following is executed. Extracting the transactions, addresses and public keys from the parameters, Party A together with B runs $\mathcal{F}_{\mathsf{Sign}}$ to sign the transactions as follows.

1. Party A receives signature $\sigma^{A,A}_{\mathsf{Fpay},i}$ on transaction $tx^{A,A}_{\mathsf{Fpay},i}$ under the shared key $\mathsf{SleepyCh}_{A}$.
2. Party B receives signature $\sigma^{B,B}_{\mathsf{Fpay},i}$ on transaction $tx^{B,B}_{\mathsf{Fpay},i}$ under the shared key $\mathsf{SleepyCh}_{B}$.
3. Party A receives signatures $\left(\sigma_{\mathsf{SleepyCh},A}, \sigma_{\mathsf{aux},A}\right)$ on the transaction $tx^{A*}_{\mathsf{Fpay},i}$ with respect to the shared keys $\mathsf{SleepyCh}_{A}$ and aux_{A}, respectively.
4. Party B receives signatures $\left(\sigma_{\mathsf{SleepyCh},B}, \sigma_{\mathsf{aux},B}\right)$ on the transaction $tx^{B*}_{\mathsf{Fpay},i}$ with respect to the shared keys $\mathsf{SleepyCh}_{B}$ and aux_{B}, respectively.
5. Party A receives signature $\sigma^{A,B}_{\mathsf{Fpay},i}$ on the transaction $tx^{A,B}_{\mathsf{Fpay},i}$ under the shared key ExitCh_{B}.
6. Party B receives signature $\sigma^{B,A}_{\mathsf{Fpay},i}$ on the transaction $tx^{B,A}_{\mathsf{Fpay},i}$ under the shared key ExitCh_{A}.
7. Party A receives signature $\sigma^{A}_{\mathsf{Pay},i}$ on the transaction $tx^{A}_{\mathsf{Pay},i}$ under the shared key Ch_{AB}.
8. Party B receives signature $\sigma^{B}_{\mathsf{Pay},i}$ on the transaction $tx^{B}_{\mathsf{Pay},i}$ under the shared key Ch_{AB}.

This takes t_s rounds and in case of failure (i.e., a signatures is not received or not valid for the specified transaction and output), execute the steps in Close. In case of success, returns to A $sig_{set\,i}^{A} := \left\{\sigma^{A,A}_{\mathsf{Fpay},i}, \left(\sigma_{\mathsf{SleepyCh},A}, \sigma_{\mathsf{aux},A}\right), \sigma^{B,A}_{\mathsf{Fpay},i}, \sigma^{A}_{\mathsf{Pay},i}\right\}$ and to B $sig_{set\,i}^{B} := \left\{\sigma^{B,B}_{\mathsf{Fpay},i}, \left(\sigma_{\mathsf{SleepyCh},B}, \sigma_{\mathsf{aux},B}\right), \sigma^{A,B}_{\mathsf{Fpay},i}, \sigma^{B}_{\mathsf{Pay},i}\right\}$

Indistinguishability: What is left at this point is to show that the UC version of the protocol is computationally indistinguishable from the one described in Section 5.

More specifically, in the UC version of the protocol we substituted (i) the 2-party key generation protocol Γ_{JKGen} for a signature scheme Π_{DS} with an idealized version $\mathcal{F}_{\mathsf{JKGen}}$ and (ii) the 2-party signing protocol Γ_{Sign} for a signature scheme Π_{DS} with an idealized version $\mathcal{F}_{\mathsf{Sign}}$. For the UC formulations we refer the reader to [5, 8]. Let Π'' be the protocol we presented in Section 5.

Π': We define Π' as Π'' except that the (UC-secure) 2-party key generation protocol Γ_{JKGen} for a signature scheme Π_{DS} is replaced by an idealized version $\mathcal{F}_{\mathsf{JKGen}}$. Such ideal functionality samples a key pair honestly and simulates the shares of the corrupted party.

$\Pi'' \approx \Pi'$: Towards a contradiction, we assume that there exists an adversary $\mathcal{A}$ that can computationally distinguish between Π' and Π''. We can construct a reduction algorithm $\mathcal{R}$ that uses $\mathcal{A}$ as a subprocedure. Since the two protocols only differ in Γ_{JKGen} being replaced by $\mathcal{F}_{\mathsf{JKGen}}$, $\mathcal{R}$ using $\mathcal{A}$ can be used to distinguish a keyshare of Γ_{JKGen} from the data received in $\mathcal{F}_{\mathsf{JKGen}}$, which in turn would break the security of our 2-party key generation protocol with non-negligible probability.

Π: We define Π as Π' except that the (UC-secure) 2-party signing protocol Γ_{Sign} for a signature scheme Π_{DS} is replaced with an idealized version $\mathcal{F}_{\mathsf{Sign}}$, which signs messages locally and simulates the interaction of corrupted parties. Note that this corresponds to the UC version of the protocol.

$\Pi' \approx \Pi$: Towards a contradiction, we assume that there exists an adversary $\mathcal{A}$ that can computationally distinguish between Π and Π'. Since the two protocols only differ in Γ_{Sign} being replaced by $\mathcal{F}_{\mathsf{Sign}}$, this means that $\mathcal{A}$ is able to distinguish a real interaction from a simulated one with non-negligible probability. This is a contradiction against the UC-security of Γ_{Sign}.

A.1 UC Simulator

In this section we give the pseudocode of a simulator for the formal Sleepy Channel protocol Π of Appendix A in the ideal world. Our simulator interacts with $\mathcal{F}$ and $\mathbb{B}$. The subprotocol $\texttt{SignTxs}^P$ refers to the one given in the formal protocol description. Normally, the challenge of providing a UC-simulation proof is that the simulator is not given the secret inputs of parties sent by the environment. Instead, the functionality usually specifies exactly what is leaked to the simulator, and the simulator has to generate a simulated transcript merely from this leaked information. The simulated transcript has to be indistinguishable from the transcript that is the result of the real world protocol execution.

Note that in our model, all messages to the functionality are implicitly forwarded to the simulator, i.e., there are no secret inputs. Hence, we can omit the simulation of the case where both protocol participants are honest; the simulator in this case would merely need to recreate the side-effect of the protocol code, which can be easily achieved with access to all the messages sent to the functionality. Indeed, the main challenge in our setting is to handle any behavior of malicious parties.

Simulator for Create

<u>Case A is honest and B is corrupted</u>

Upon A sending $(\texttt{CREATE}, \gamma, tid_A) \overset{\tau_0}{\hookrightarrow} \mathcal{F}$, if B does not send $(\texttt{CREATE}, \gamma, tid_B) \overset{\tau}{\hookrightarrow} \mathcal{F}$ where $|\tau_0 - \tau| \leq T_1$, then distinguish the following cases:

1. If B sends $(\text{createInfo}, \mathsf{id}, tid_B, pkey^B_{set}) \overset{\tau_0}{\hookrightarrow} A$, then send $(\texttt{CREATE}, \gamma, tid_B) \overset{\tau_0}{\hookrightarrow} \mathcal{F}$ on behalf of B.
2. Otherwise stop.

Do the following:

1. Set $\mathsf{id} := \gamma.\mathsf{id}$, generate $(pk_{\mathsf{CPay},A}, sk_{\mathsf{CPay},A})$, $(pk_{\mathsf{pun},A}, sk_{\mathsf{pun},A})$, $(pk_{\mathsf{fp},A}, sk_{\mathsf{fp},A})$ and $(pk_{\mathsf{ffp},A}, sk_{\mathsf{ffp},A})$. Let $pkey^A_{set}$ be the set of public keys of these key pairs. Send $(\text{createInfo}, \mathsf{id}, tid_A, pkey^A_{set}) \overset{\tau_0}{\hookrightarrow} B$.
2. If you receive $(\text{createInfo}, \mathsf{id}, tid_B, pkey^B_{set}) \xleftarrow{\tau_0+1} B$, do the following. Else go idle.
3. Using $pkey^A_{set}$ and $pkey^B_{set}$, the simulator on behalf of A together with B runs $\mathcal{F}_{\mathsf{JKGen}}$ to generate the following set of shared addresses: $addr_{set} := \{\mathsf{Ch}_{AB}, \mathsf{SleepyCh}_A, \mathsf{SleepyCh}_B, \mathsf{ExitCh}_A, \mathsf{ExitCh}_B, \mathsf{aux}_A, \mathsf{aux}_B\}$ which takes t_g rounds. In case of failure, abort.
4. Generate $tx_f := tx([tid_A, tid_B], [\mathsf{Ch}_{AB}], [2 \cdot c + v_{A,0} + v_{B,0}])$
5. Let $tx_{set_0} \leftarrow \texttt{GenerateTxs}(addr_{set}, pkey^A_{set}, pkey^B_{set}, c, v_{A_i}, v_{B_i})$
6. Let $sig_{set_0}^A \leftarrow \texttt{SignTxs}^A(tx_{set0}, addr_{set}, pkey^A_{set} \cup pkey^B_{set})$
7. Generates a signature on behalf of A, σ_{tid_A}, for the output tid_A and send $(\text{createFund}, \mathsf{id}, \sigma_{tid_A}) \xhookrightarrow{\tau_0+1+t_g+t_s} A$.
8. If you $(\text{createFund}, \mathsf{id}, \sigma_{tid_B}) \xhookleftarrow{\tau_0+2+t_g+t_s} B$, post $(tx_F, \{\sigma_{tid_A}, \sigma_{tid_B}\})$ to $\mathbb{B}$.
9. If tx_F is accepted by $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + 2 + t_g + t_s + \Delta$, store $\Gamma^A(\mathsf{id}) := (tx_F, tx_{set0}, sig_{set_0}^A, addr_{set}, pkey^A_{set}, pkey^B_{set})$.

Simulator for Update

<u>Case A is honest and B is corrupted</u>

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

1. $(\text{updateReq}, \mathsf{id}, \overrightarrow{\theta}, t_{\mathsf{stp}}) \overset{t_0}{\hookrightarrow} B$
2. Upon $(\text{updateInfo}, \mathsf{id}) \xhookleftarrow{t_0+2} B$, do the following
3. Retrieve $(tx_F, tx_{set_{i-1}}, sig_{set_{i-1}}^A, addr_{set}, pkey_{set}^A, pkey_{set}^B) = \Gamma^A(\mathsf{id})$
4. Extract $v_{A,i}$ and $v_{B,i}$ from $\overrightarrow{\theta}$, and c from tx_F
5. Let $tx_{set_i} \leftarrow \mathtt{GenerateTxs}(addr_{set}, pkey_{set}^A, pkey_{set}^B, c, v_{A_i}, v_{B_i})$
6. Let $\overrightarrow{tid} := (tx_{\mathsf{Pay},i}^A.\mathsf{id}, tx_{\mathsf{Pay},i}^B.\mathsf{id})$ be a tuple of the transaction ids of transaction $tx_{\mathsf{Pay},i}^A$ and $tx_{\mathsf{Pay},i}^B$. Inform $\mathcal{F}$ of $\overrightarrow{tid}$ in round $t_0 + 2$.
7. If A sends $(\mathtt{SETUP{-}OK}, \mathsf{id}) \xhookrightarrow{t_1 \le t_0+2+t_{\mathsf{stp}}} \mathcal{F}$, send $(\text{updateCom}, \mathsf{id}) \overset{t_1}{\hookrightarrow} B$
8. Wait one round.
9. If in round $t_1 + 1$, B starts executing $\mathtt{SignTxs}^A(tx_{set_i}, addr_{set}, pkey_{set}^A \cup pkey_{set}^B)$, send $(\mathtt{UPDATE{-}OK}, \mathsf{id}) \xhookrightarrow{t_1+1} \mathcal{F}$ on behalf of B
10. $\mathtt{SignTxs}^A(tx_{set_i}, addr_{set}, pkey_{set}^A \cup pkey_{set}^B)$
11. If $sig_{set_i}^A$ is returned from $\mathtt{SignTxs}^A$, instruct $\mathcal{F}$ to $(\mathtt{UPDATE{-}OK}, \mathsf{id}) \xhookrightarrow{t_1+1+t_s} \mathcal{E}$ via A. Else, execute $\mathtt{ForceClose}^A(\mathsf{id})$ and go idle.
12. If A does not send $(\mathtt{REVOKE}, \mathsf{id}) \xhookrightarrow{t_1+1+t_s} \mathcal{F}$, go idle.
13. The simulator on behalf of A together with B runs the interactive protocol $\mathcal{F}_{\mathsf{Sign}}$ to generate the following signature. $\sigma_{\mathsf{Pnsh},i}^A$ on the punishment transaction $tx_{\mathsf{Pnsh},i}^A$. Party A receives $\sigma_{\mathsf{Pnsh},i}^A$ as output. This takes t_r rounds. In case of failure, execute $\mathtt{ForceClose}^A(\mathsf{id})$.
14. $(\mathsf{revoke}, \mathsf{id}, \sigma_{\mathsf{Pnsh},i}^A) \xhookrightarrow{t_1+1+t_s+t_r} B$
15. If B starts $\mathcal{F}_{\mathsf{Sign}}$ to sign $tx_{\mathsf{Pnsh},i}^B$ in round $t_1 + 2 + t_s + t_r$, send $(\mathtt{REVOKE}, \mathsf{id}) \xhookrightarrow{t_1+2+t_s+t_r} \mathcal{F}$ on behalf of B and participate in the signing on behalf of A.
16. If $(\mathsf{revoke}, \mathsf{id}, \sigma_{\mathsf{Pnsh},i}^B) \xhookleftarrow{t_1+3+t_s+2t_r} B$ and the signature is valid, go to next step. Else, execute $\mathtt{ForceClose}^A(\mathsf{id})$.
17. $\Theta^A(\mathsf{id}) := \Theta^A \cup \left\{(tx_{set_{i-1}}, sig_{set_{i-1}}^A, \sigma_{\mathsf{Pnsh},i-1}^B)\right\}$
18. $\Gamma^A(\mathsf{id}) := (tx_F, tx_{set_i}, sig_{set_i}^A, addr_{set}, pkey_{set}^A, pkey_{set}^B)$

<u>Case B is honest and A is corrupted</u>

Upon A sending $(\text{updateReq}, \mathsf{id}, \overrightarrow{\theta}, t_{\mathsf{stp}}) \overset{t_0}{\hookrightarrow} B$, send $(\mathtt{UPDATE}, \mathsf{id}, \overrightarrow{\theta}, t_{\mathsf{stp}}) \overset{t_0}{\hookrightarrow} \mathcal{F}$ on behalf of A, if A has not already sent this message. Proceed as follows:

1. Upon $(\text{updateReq}, \mathsf{id}, \overrightarrow{\theta}, t_{\mathsf{stp}}) \overset{\tau_0}{\hookleftarrow} A$, do the following
2. Retrieve $(tx_F, tx_{set_{i-1}}, sig_{set_{i-1}}^B, addr_{set}, pkey_{set}^A, pkey_{set}^B) = \Gamma^B(\mathsf{id})$
3. Extract $v_{A,i}$ and $v_{B,i}$ from $\overrightarrow{\theta}$, and c from tx_F
4. Let $tx_{set_i} \leftarrow \mathtt{GenerateTxs}(addr_{set}, pkey_{set}^A, pkey_{set}^B, c, v_{A_i}, v_{B_i})$
5. Let $\overrightarrow{tid} := (tx_{\mathsf{Pay},i}^A.\mathsf{id}, tx_{\mathsf{Pay},i}^B.\mathsf{id})$ be a tuple of the transaction ids of transaction $tx_{\mathsf{Pay},i}^A$ and $tx_{\mathsf{Pay},i}^B$. Inform $\mathcal{F}$ of $\overrightarrow{tid}$.
6. $(\text{updateInfo}, \mathsf{id}) \overset{\tau_0}{\hookrightarrow} A$
7. Upon A sending $(\text{updateCom}, \mathsf{id}) \xhookrightarrow{\tau_0+1+t_{\mathsf{stp}}} B$, send $(\mathtt{SETUP{-}OK}, \mathsf{id}) \overset{\tau_1}{\hookrightarrow} \mathcal{F}$ on behalf of A.
8. Receive $(\text{updateCom}, \mathsf{id}) \xhookleftarrow{\tau_1 \le \tau_0+2+t_{\mathsf{stp}}} A$
9. If B sends $(\mathtt{UPDATE{-}OK}, \mathsf{id}) \overset{\tau_1}{\hookrightarrow} \mathcal{F}$, $\mathtt{SignTxs}^A(tx_{set_i}, addr_{set}, pkey_{set}^A \cup pkey_{set}^B)$
10. If $sig_{set_i}^B$ is not returned from $\mathtt{SignTxs}^A$ in round $\tau_1 + t_s$, execute $\mathtt{ForceClose}^B(\mathsf{id})$ and go idle.
11. If A starts the $\mathcal{F}_{\mathsf{Sign}}$ in round $\tau_1 + t_s$ to generate $\sigma_{\mathsf{Pnsh},i}^A$, send $(\mathtt{REVOKE}, \mathsf{id}) \xhookrightarrow{\tau_1+t_s} \mathcal{F}$ on behalf of A. Participate in the signing on behalf of B.
12. Upon $(\mathsf{revoke}, \mathsf{id}, \sigma_{\mathsf{Pnsh},i}^A) \xhookleftarrow{\tau_1+1+t_s+t_r} A$, continue. Else, execute $\mathtt{ForceClose}^B(\mathsf{id})$ and go idle.
13. If B does not send $(\mathtt{REVOKE}, \mathsf{id}) \xhookrightarrow{\tau_1+1+t_s+t_r} \mathcal{F}$, go idle.
14. $\mathcal{S}$ on behalf of B together with A runs the interactive protocol $\mathcal{F}_{\mathsf{Sign}}$ to generate the following signature. $\sigma_{\mathsf{Pnsh},i}^B$ on the punishment transaction $tx_{\mathsf{Pnsh},i}^B$. Party B receives $\sigma_{\mathsf{Pnsh},i}^B$ as output after t_r. In case of failure, execute $\mathtt{ForceClose}^B(\mathsf{id})$.
15. $(\mathsf{revoke}, \mathsf{id}, \sigma_{\mathsf{Pnsh},i}^B) \xhookrightarrow{\tau_1+1+t_s+2t_r} A$
16. $\Theta^B(\mathsf{id}) := \Theta^B \cup \left\{(tx_{set i-1}, sig_{set_{i-1}}^B, \sigma_{\mathsf{Pnsh},i-1}^B)\right\}$
17. $\Gamma^B(\mathsf{id}) := (tx_F, tx_{set_i}, sig_{set_i}^B, addr_{set}, pkey_{set}^A, pkey_{set}^B)$

Simulator for Close

<u>Case A is honest and B is corrupted</u>

Upon A sending $(\mathtt{CLOSE}, \mathsf{id}) \overset{t_0}{\hookrightarrow} \mathcal{F}$, do the following.

1. Extract $(tx_F, tx_{set_i}, sig_{set_i}^A, addr_{set}, pkey_{set}^A, pkey_{set}^B)$ from $\Gamma^A(\mathsf{id})$.
2. Extract $v_{A,i}$ and $v_{B,i}$ from $tx_{\mathsf{Pay},j}^A \in tx_{set_i}$, and c from tx_F
3. Create transaction $tx_c := tx(\mathsf{Ch}_{AB}, \{pk_A, pk_B\}, \{v_{A,i} + c, v_{B,i} + c\})$, where pk_A is an address controlled by A and pk_B an address controlled by B.
4. The simulator on behalf of A together with B runs the inter-

active protocol $\mathcal{F}_{\mathsf{Sign}}$ to generate the following signature, σ_{tx_c} on the transaction tx_c. This takes t_r rounds.

5. In case the signature generation was successful, post (tx_c, σ_{tx_c}) on $\mathbb{B}$ and send $(\texttt{CLOSE}, \mathsf{id}) \xhookrightarrow{t_0+t_r} \mathcal{F}$ on behalf of B. Else, execute $\texttt{ForceClose}^A(\mathsf{id})$.
6. If tx_c appears on $\mathbb{B}$ in round $t_1 \leq t_0 + t_r + \Delta$, set $\Theta^A(\mathsf{id}) := \bot$, $\Gamma^A(\mathsf{id}) := \bot$.

Simulator for Punish

Case A is honest and B is corrupted

Upon A sending $\texttt{PUNISH} \overset{\tau_0}{\hookrightarrow} \mathcal{F}$, for each $\mathsf{id} \in \{0,1\}^*$ such that $\Theta^A(\mathsf{id}) \neq \bot$ do the following:

1. Parse $\{(tx_{set\,i}, sig^A_{set\,i}, \sigma^B_{\mathsf{Pnsh},i})\}_{i \in m} := \Theta^A(\mathsf{id})$ and extract γ from $\Gamma^A(\mathsf{id})$. If for some $i \in m$, there exist a transaction $tx^B_{\mathsf{Pay},i} \in tx_{set\,i}$ on $\mathbb{B}$ do the following.
2. Post $\left(tx^B_{\mathsf{Pnsh},i}, \sigma^B_{\mathsf{Pnsh},i}\right)$ on $\mathbb{B}$ before the absolute timeout $\mathbf{T}$.
3. Let $tx^B_{\mathsf{Pnsh},i}$ be accepted by $\mathbb{B}$ in round $t_1 \leq t_0 + \Delta$. Post $(tx^{B,A}_{\mathsf{Fpay},i}, \sigma_{tx^{B,A}_{\mathsf{Fpay},i}} \in sig^A_{set\,i})$
4. After $tx^{B,A}_{\mathsf{Fpay},i}$ is accepted by $\mathbb{B}$ in round $t_2 \leq t_1 + \Delta$, set $\Theta^A(\mathsf{id}) := \bot$, $\Gamma^A(\mathsf{id}) := \bot$.

Simulator for $\texttt{ForceClose}^P(\mathsf{id})$

Let τ_0 be the current round

1. Extract $(tx_F, tx_{set0}, sig^A_{set0}, addr_{set}, pkey^A_{set}, pkey^B_{set})$ from $\Gamma^A(\mathsf{id})$ and extract $tx^A_{\mathsf{Pay},j}$ from tx_{set} and $\sigma^A_{\mathsf{Pay},j}$ and sig_{set}.
2. Post $\left(tx^A_{\mathsf{Pay},j}, \sigma^A_{\mathsf{Pay},j}\right)$ on $\mathbb{B}$
3. Let $t_2 \leq t_1 + \Delta$ be the round in which $tx^A_{\mathsf{Pay},j}$ is accepted by $\mathbb{B}$.
4. If $tx^{A,B}_{\mathsf{Fpay},i}$ appears on $\mathbb{B}$ at or after round $t_3 \leq t_2 + \Delta$ and before $\mathbf{T}$, post $\left(tx^A_{\mathsf{Pay},j}, \sigma^A_{\mathsf{Pay},j}\right)$. Otherwise, post $\left(tx^{A,A}_{\mathsf{Fpay},i}, \sigma^{A,A}_{\mathsf{Fpay},i}\right)$ after $\mathbf{T}$. Set $\Gamma^P(\mathsf{id}) := \bot$, $\Theta^P(\mathsf{id}) := \bot$.

A.2 Simulation proof

To proof that the protocol is a (G)UC-realization of the functionality $\mathcal{F}$, we show that the execution ensembles $EXEC_{\Pi,\mathcal{A},\mathcal{E}}$ and $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable. I.e., for the simulator $\mathcal{S}$ presented in Appendix A.1, for every environment the interaction with $\mathcal{S}$ and $\mathcal{F}$ is computationally indistinguishable from the interaction with $\mathcal{A}$ and Π. We show this for the different phases Create, Update, Close, Punish as well as the subprotocol ForceClose.

For readability we define $m[\tau]$ to capture the fact that a message m is observed by the environment in round τ. Note that messages sent to parties in the protocol that are under adversarial control observe the message after one round. Additionally, we interact with other functionalities, e.g., for signing and the ledger. To capture any side effect observable by the environment including messages sent parties who are potentially controlled by the adversary or changing public variables such as the ledger, we do the following. We denote $\mathsf{obsSet}(\mathsf{action}, \tau)$ as the set of all observable side effects triggered by action action in round τ. Finally, we refer to a message by the message identifier, e.g., $\texttt{CREATE}$ or createInfo. We note that other message parameters are omitted. Instead, we refer to relevant parts in the ideal world and the real world, where one can verify that indeed the same objects are created, checks are performed, etc.

We require a SUF-CMA secure signature scheme Σ and a ledger $\mathbb{B}(\Delta, \Sigma, \mathcal{V})$ where $\mathcal{V}$ allows for transaction authorization under Σ and absolute time-locks.[6] The former property is needed to ensure that the environment and malicious party cannot generate signatures on behalf of honest parties with non-negligible probability. Instead, only the simulator can generate signatures on behalf of honest parties. Further, we require a ledger that supports transaction authorization under Σ and absolute time-locks for encoding our construction.

Lemma 2. *The Create phase of Π UC-realizes the Create phase of $\mathcal{F}$.*

Proof. We consider the case where A is honest and B is corrupted. Note that the reverse case is symmetric.

Real World: After receiving $\texttt{CREATE}$ in round t_0, A sends createInfo to B in t_0. If A receives also createInfo in $t_0 + 1$, A will perform first the action $a_0 :=$ "run address generation" in round $t_0 + 1$ and on success, create the transactions for the channel followed by $a_1 :=$ "create signatures" in round $t_0 + 1 + t_g$. If this is successful, A generates the signature for the funding tx tx_F and sends the signature via createFund to B in $t_0 + 1 + t_g + t_s$. If A receives also createFund from B in round $t_0 + 2 + t_g + t_s$, it will perform action $a_2 :=$ "Post funding tx on $\mathbb{B}$". If it is accepted in round $t_1 \leq t_0 + 2 + t_g + t_s + \Delta$, finally A will output $\texttt{CREATED}$. Thus, the execution ensemble is $EXEC^{\text{create}}_{\Pi,\mathcal{A},\mathcal{E}} := \{\text{createFund}[t_0+1], \mathsf{obsSet}(a_0, t_0+1), \mathsf{obsSet}(a_1, t_0+1+t_g), \text{createFund}[t_0+2+t_g+t_s], \mathsf{obsSet}(a_2, t_0+2+t_g+t_s), \texttt{CREATED}[t_1]\}$.

[6] The necessity for time-locks can be dropped when using verifiable timed signatures (VTS) as discussed in Section 5.1.3, although we do not provide a formal analysis for such variant here.

Ideal World: After A sending CREATE in round t_0 to $\mathcal{F}$, the simulator sends createInfo to B. If B sends createInfo to A, the simulator informs $\mathcal{F}$ and performs a_0 in round t_0+1. Upon success, $\mathcal{S}$ creates the transactions for the channel and performs a_1 in round t_0+1+t_g. If this was successful, the simulator on behalf of A generates the signature of tx_F and sends createFund to B in $t_0+1+t_g+t_s$. If B sends also createFund to A, received in $t_0+2+t_g+t_s+\Delta$, perform a_2 in $t_0+2+t_g+t_s+\Delta$. If the funding tx is accepted in round $t_1 \leq t_0+2+t_g+t_s+\Delta$, $\mathcal{F}$ (which expects it after being informed by $\mathcal{S}$) outputs CREATED in round $t_1 \leq t_0+2+t_g+t_s+\Delta$. Thus, the execution ensemble is $EXEC^{\text{create}}_{\mathcal{F},\mathcal{S},\mathcal{E}} := \{\text{createFund}[t_0+1], \mathsf{obsSet}(a_0, t_0+1), \mathsf{obsSet}(a_1, t_0+1+t_g), \text{createFund}[t_0+2+t_g+t_s], \mathsf{obsSet}(a_2, t_0+2+t_g+t_s), \mathtt{CREATED}[t_1]\}$ □

Lemma 3. *The ForceClose subprotocol of* Π *UC-realizes the ForceClose subprocedure of* $\mathcal{F}$.

Proof. We consider the case where A is honest and B is corrupted. Note that the reverse case is symmetric.

Real World: Taking the latest state, a performs action $a_0 :=$ "post $\left(tx^{A}_{\mathsf{Pay},j}, \sigma^{A}_{\mathsf{Pay},j}\right)$ on $\mathbb{B}$" in round t_0. After the transaction appears on $\mathbb{B}$ in round $t_1 \leq t_0+\Delta$, do the following depending on B. Either (i) the transaction $tx^{A,B}_{\mathsf{Fpay},i}$ appears on $\mathbb{B}$ in round $t_2 \leq t_1+\Delta$ and before $\mathbf{T}$. In this case, A posts $\left(tx^{A}_{\mathsf{Pay},j}, \sigma^{A}_{\mathsf{Pay},j}\right)$, which we denote as action a_1, followed by sending CLOSED in round $t_m := t_3 leq t_2+\Delta$. Otherwise, (ii) A posts $\left(tx^{A,A}_{\mathsf{Fpay},i}, \sigma^{A,A}_{\mathsf{Fpay},i}\right)$ after $\mathbf{T}$, which we denote as action a_2, followed by sending CLOSED in round $t_m := t_4 \leq \mathbf{T}+\Delta$. Thus, the execution ensemble is $EXEC^{\text{forceclose}}_{\Pi,\mathcal{A},\mathcal{E}} := \{\mathsf{obsSet}(a_0,t_0), o \in \{\mathsf{obsSet}(a_1,t_2), \mathsf{obsSet}(a_2,\mathbf{T})\}, \mathtt{CLOSED}[t_m]\}$.

Ideal World: Taking the latest state, the simulator will mirror the behavior of the real world. In round t_0, it will performs action a_0. After the transaction appears on $\mathbb{B}$ in round $t_1 \leq t_0+\Delta$, do the following depending on B. Either (i) the transaction $tx^{A,B}_{\mathsf{Fpay},i}$ appears on $\mathbb{B}$ in round $t_2 \leq t_1+\Delta$ and before $\mathbf{T}$. In this case, the simulator posts $\left(tx^{A}_{\mathsf{Pay},j}, \sigma^{A}_{\mathsf{Pay},j}\right)$, which we denote as action a_1. Otherwise, (ii) the simulator posts $\left(tx^{A,A}_{\mathsf{Fpay},i}, \sigma^{A,A}_{\mathsf{Fpay},i}\right)$ after $\mathbf{T}$, which we denote as action a_2.Meanwhile, the functionality $\mathcal{F}$ expects that either of these transactions appears on $\mathbb{B}$. If this happens, either in round $t_m := t_3 \leq t_2+\Delta$ in case (i) or in round $t_m := t_4 \leq \mathbf{T}+\Delta$, it outputs CLOSED. Thus, the execution ensemble is $EXEC^{\text{forceclose}}_{\mathcal{F},\mathcal{S},\mathcal{E}} := \{\mathsf{obsSet}(a_0,t_0), o \in \{\mathsf{obsSet}(a_1,t_2), \mathsf{obsSet}(a_2,\mathbf{T})\}, \mathtt{CLOSED}[t_m]\}$. □

Lemma 4. *The Update phase of* Π *UC-realizes the Update phase of* $\mathcal{F}$.

Proof. We start by considering the case where A is honest and B is corrupted.

Real World: A upon UPDATE in round t_0 does the following. The update phase consists of the following steps: Informing B, generating the transactions for the new state, signing these transactions, signing the revocation for B and signing the revocation for A. We capture the steps visible to the $\mathcal{E}$ below, together with their dependencies. The execution ensemble $EXEC^{\text{update}}_{\Pi,\mathcal{A},\mathcal{E}}$ follows as a list for better readability.

- updateReq to B in round t_0
- SETUP to $\mathcal{E}$ in t_0+2 (if received updateInfo from B)
- updateCom to B in round $t_1 \leq t_0+2+t_{\mathsf{stp}}$(if received SETUP–OK from $\mathcal{E}$)
- SignTxs in t_1+1
- UPDATE–OK to $\mathcal{E}$ in round t_1+1+t_s (if signing successful)
- sign revocation of B with B in round t_1+1+t_s (if REVOKE from $\mathcal{E}$)
- revoke to B in round $t_1+1+t_s+t_r$ (if signing successful)
- sign revocation of A with B in round $t_1+2+t_s+t_r$
- UPDATED to $\mathcal{E}$ in round $t_1+3+t_s+2t_r$ (if signature for revocation received from B)

Ideal World: Upon A sending UPDATE in round t_0 to $\mathcal{F}$, $\mathcal{S}$ simulates the protocol view to $\mathcal{E}$. The same steps of the update phase have to be conducted: Informing B, generating the transactions for the new state, signing these transactions, signing the revocation for B and signing the revocation for A. We capture the steps visible to the $\mathcal{E}$ below, together with their dependencies and if they are executed by $\mathcal{S}$ or $\mathcal{F}$. The execution ensemble $EXEC^{\text{update}}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ follows as a list for better readability.

- updateReq to B in round t_0 ($\mathcal{S}$)
- SETUP to $\mathcal{E}$ in t_0+2 (if received updateInfo from B) ($\mathcal{F}$)
- updateCom to B in round $t_1 \leq t_0+2+t_{\mathsf{stp}}$ (if received SETUP–OK from $\mathcal{E}$) ($\mathcal{S}$)
- SignTxs in t_1+1 ($\mathcal{S}$)
- UPDATE–OK to $\mathcal{E}$ in round t_1+1+t_s (if signing successful) ($\mathcal{F}$ after instructed by $\mathcal{S}$)

- sign revocation of B with B in round t_1+1+t_s (if REVOKE from $\mathcal{E}$) ($\mathcal{S}$)
- revoke to B in round $t_1+1+t_s+t_r$ (if signing successful) ($\mathcal{S}$
- sign revocation of A with B in round $t_1+2+t_s+t_r$ ($\mathcal{S}$)
- UPDATED to $\mathcal{E}$ in round $t_1+3+t_s+2t_r$ (if signature for revocation received from B) ($\mathcal{F}$)

Now we consider the case where B is honest and A is corrupted.

Real World: A upon UPDATE in round t_0 does the following. The update phase consists of the following steps: Generating the transactions for the new state, signing these transactions, signing the revocation for A and signing the revoaction for B. Similar to the previous case, we capture the steps visible to the $\mathcal{E}$ below, together with their dependencies. The execution ensemble $EXEC^{\text{update}}_{\Pi,\mathcal{A},\mathcal{E}}$ follows as a list for better readability.

- UPDATE–REQ to $\mathcal{E}$ in round τ_0 (if received updateReq from A)
- updateInfo to A in round τ_0
- SETUP–OK to $\mathcal{E}$ in round $\tau_1 \leq \tau_0+2+t_{\text{stp}}$ (if received updateCom from A)
- SignTxs in τ_1
- sign revocation of B with A in round τ_1+t_s (if previous signing was successful)
- REVOKE–REQ to $\mathcal{E}$ in round τ_1+1+t_s (after receiving revoke from A in that round)
- sign revocation of A with A in round $\tau_1+1+t_s+t_r$
- revoke to A in round $\tau_1+1+t_s+2t_r$ (in case revocation was signed successfully)
- UPDATED to $\mathcal{E}$ in round $\tau_1+2+t_s+2t_r$

Ideal World: Upon A sending UPDATE in round t_0 to $\mathcal{F}$, $\mathcal{S}$ simulates the protocol view to $\mathcal{E}$. The same steps of the update phase have to be conducted: Generating the transactions for the new state, signing these transactions, signing the revocation for B and signing the revocation for A. We capture the steps visible to the $\mathcal{E}$ below, together with their dependencies and if they are executed by $\mathcal{S}$ or $\mathcal{F}$. The execution ensemble $EXEC^{\text{update}}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ follows as a list for better readability.

- UPDATE–REQ to $\mathcal{E}$ in round τ_0 (if received updateReq from A) ($\mathcal{F}$)
- updateInfo to A in round τ_0 ($\mathcal{S}$)
- SETUP–OK to $\mathcal{E}$ in round $\tau_1 \leq \tau_0+2+t_{\text{stp}}$ (if received updateCom from A) ($\mathcal{F}$)
- SignTxs in τ_1 ($\mathcal{S}$)
- sign revocation of B with A in round τ_1+t_s (if previous signing was successful) ($\mathcal{S}$)
- REVOKE–REQ to $\mathcal{E}$ in round τ_1+1+t_s (after receiving revoke from A in that round) ($\mathcal{F}$)
- sign revocation of A with A in round $\tau_1+1+t_s+t_r$ ($\mathcal{S}$)
- revoke to A in round $\tau_1+1+t_s+2t_r$ (in case revocation was signed successfully) ($\mathcal{S}$)
- UPDATED to $\mathcal{E}$ in round $\tau_1+2+t_s+2t_r$ ($\mathcal{F}$)

□

Lemma 5. *The Close phase of* Π *UC-realizes the Close phase of* $\mathcal{F}$*.*

Proof. We consider the case where A is honest and B is corrupted. Note that the reverse case is symmetric.

Real World: After receiving CLOSE in round t_0, A creates a closing transaction tx_c from the latest state of the channel. A then performs action $a_0 :=$ create signature for tx_c with B. In case of success, A performs $a_1 :=$ post tx_c on $\mathbb{B}$ in round t_0+t_r. If it appears in round $t_1 \leq t_0+t_r+\Delta$, send CLOSED. If the signature generation was unsuccessful in round $t_2 \geq t_0$, A runs $a_2 :=$ ForceClose. Thus, the execution ensemble is either $EXEC^{\text{close}}_{\Pi,\mathcal{A},\mathcal{E}} := \{\mathsf{obsSet}(a_0,t_0), \mathsf{obsSet}(a_1,t_0+t_r), \texttt{CLOSED}[t_1]\}$ or $EXEC^{\text{close}}_{\Pi,\mathcal{A},\mathcal{E}} := \{\mathsf{obsSet}(a_0,t_0), \mathsf{obsSet}(a_2,t_2)\}$.

Ideal World: In this case, after A receving CLOSE in round t_0, $\mathcal{S}$ handles creating the transaction and performing a_0 in round t_0 and a_1 in t_0+t_r, while $\mathcal{F}$ sends CLOSED if the closing transaction appears on $\mathbb{B}$ in round $t_1 \leq t_0+t_r+\Delta$. If the signature generation was unsuccessful in round $t_2 \geq t_0$, the simulator will perform a_2 and instruct $\mathcal{F}$ to do the same (by not sending CLOSE on behalf of B). Thus, the execution ensemble is $EXEC^{\text{close}}_{\mathcal{F},\mathcal{S},\mathcal{E}} := \{\mathsf{obsSet}(a_0,t_0), \mathsf{obsSet}(a_1,t_0+t_r), \texttt{CLOSED}[t_1]\}$ or $EXEC^{\text{close}}_{\mathcal{F},\mathcal{S},\mathcal{E}} := \{\mathsf{obsSet}(a_0,t_0), \mathsf{obsSet}(a_2,t_2)\}$.

□

Lemma 6. *The Punish phase of* Π *UC-realizes the Punish phase of* $\mathcal{F}$*.*

Proof. We consider the case where A is honest and B is corrupted. Note that the reverse case is symmetric.

Table 2: Overhead for operations, given a current fee of 102 satoshi per byte and a price of $57,202$ USD per BTC.

	txs off-chain	bytes	txs on-chain	bytes	USD
create	$2 \cdot (tx^{A}_{\mathsf{Pay},i} + tx^{A,B}_{\mathsf{Fpay},i} + tx^{A*}_{\mathsf{Fpay},i} + tx^{A,A}_{\mathsf{Fpay},i})$	2026	tx_F	338	2.13
update	$2 \cdot (tx^{A}_{\mathsf{Pay},i} + tx^{A,B}_{\mathsf{Fpay},i} + tx^{A*}_{\mathsf{Fpay},i} + tx^{A,A}_{\mathsf{Fpay},i} + tx^{A}_{\mathsf{Pnsh},i})$	2408	-	-	-
close (optimistic)	-	-	$tx^{A}_{\mathsf{Pay},i}$	225	1.42
close (slow)	-	-	$tx^{A}_{\mathsf{Pay},i} + tx^{A,A}_{\mathsf{Fpay},i}$	449	2.83
close (fast)	-	-	$tx^{A}_{\mathsf{Pay},i} + tx^{A,B}_{\mathsf{Fpay},i} + tx^{A*}_{\mathsf{Fpay},i}$	823	5.18
punish	-	-	$tx^{A}_{\mathsf{Pay},i} + tx^{A}_{\mathsf{Pnsh},i}$	450	2.83

Real World: After A receives PUNISH from $\mathcal{E}$ in round t_0,[7] A checks if there is a transaction on the ledger that belongs to an old state of one of its channels. If yes, using the corresponding revocation secret, A performs action $a_0 :=$ post punishment transaction in round t_0. After it is accepted in round $t_1 \leq t_0 + \Delta$, A performs $a_1 :=$ post collateral unlock transaction. If that is accepted in round $t_2 \leq t_1 + \Delta$, A outputs message PUNISHED. Thus, the execution ensemble is $EXEC^{\text{punish}}_{\Pi,\mathcal{A},\mathcal{E}} := \{\mathsf{obsSet}(a_0, t_0), \mathsf{obsSet}(a_1, t_1), \mathtt{PUNISHED}[t_2]\}$.

Ideal World: The ideal functionality checks at the end of every round t_0 (this is achieved by marking itself stale if not invoked by $\mathcal{E}$, see Section 4) if a transaction spending the funding transaction that is not the most recent state is on the ledger. If it is, and the other party is honest, it expects a punishment transaction to appear in round $t_1 \leq t_0 + \Delta$. Additionally, it expects that the collateral unlock transaction of that party appears in round $t_2 \leq t_1 + \Delta$. If both appear, $\mathcal{F}$ outputs PUNISHED in round t_2. Meanwhile, the simulator will take care of posting both the punishment a_0 and the collateral unlock transaction a_1 in rounds t_0 and t_1, respectively. Thus, the execution ensemble is $EXEC^{\text{punish}}_{\mathcal{F},\mathcal{S},\mathcal{E}} := \{\mathsf{obsSet}(a_0, t_0), \mathsf{obsSet}(a_1, t_1), \mathtt{PUNISHED}[t_2]\}$.

□

Theorem A.1. *The protocol* Π *UC-realizes the the ideal functionality* $\mathcal{F}$.

Proof. The proof of the theorem follows by a standard hybrid argument and an application of Lemmas 2 to 6.

□

B Deployment cost

To further evaluate our Sleepy Channels protocol, we want to measure the cost in terms of on-chain fees when using the protocol. Taking the numbers from Section 6, we do the following. To post a Bitcoin transaction to the blockchain, one has to give a certain amount of fees to the miner. This fee is dependent on the size of the transaction. At the time of writing, the fee of including a transaction to the next block is 11 satoshis per byte and the price of 1 Bitcoin in USD is 57202,30. Together with the fact that there are 10^8 satoshis in one Bitcoin, we can compute the fees in USD for each of the Sleepy Channels operations. We show our results in Table 2.

[7] Note that we require the environment to send this message, as we defined that all security guarantees of $\mathcal{F}$ are lost in the case of message ERROR. However, this is exactly what happens if the environment does not give the execution token to $\mathcal{F}$ via PUNISH, see Section 4