Sleepy Channels: Bi-directional Payment Channels without Watchtowers

Lukas Aumayr*
TU Wien
Vienna, Austria
lukas.aumayr@tuwien.ac.at

Sri AravindaKrishnan Thyagarajan*
Carnegie Mellon University
Pittsburgh, USA
t.srikrishnan@gmail.com

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

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

Matteo Maffei
Christian Doppler Laboratory
Blockchain Technologies for the Internet of Things, TU Wien
Vienna, Austria
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limitation: Both parties need to constantly monitor the blockchain to ensure that the other party did not post an *outdated* transaction.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., 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 when 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, our experimental results show that 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.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**.

KEYWORDS

cryptocurrencies; payment channels; watchtowers; blockchain

ACM Reference Format:
Lukas Aumayr, Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, Pedro Moreno-Sanchez, and Matteo Maffei. 2022. Sleepy Channels: Bi-directional Payment Channels without Watchtowers. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/3548606.3559370

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) [40] 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) [6], which at the time of writing hosts bitcoins worth more than 130M USD, in a total of more than 19k nodes and more than 81k 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.

*These two authors contributed equally to the work.

CCS '22, November 7–11, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9450-5/22/11.
https://doi.org/10.1145/3548606.3559370

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 PC can be finally closed by including a *close-channel* transaction on-chain that effectively submits the last authenticated distribution of coins to the blockchain (or after the PC 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 [33]. This approach, however, adds a trust assumption that goes against the decentralization philosophy of cryptocurrencies and it is unclear whether it holds in practice [23, 48]. Another approach consists on relying on a third-party committee [14, 22] 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 compatibility 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).

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.

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

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 watchtower constructions have been proposed so far [11, 15, 16, 32, 37, 38], 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. This requirement is present even in the watchtower proposals [11, 15, 16, 32, 37, 38], where the users are required to come online before a specific time to ensure the watchtower has acted correctly. Moreover, our protocol does not make use of any complex script and is therefore backwards compatible with existing UTXO-based cryptocurrencies, many of which can avail bi-directional payment channels without additional trust assumptions for the very first time.

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 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 some amount of collateral each (same or different amounts for the two users), whose exact value is 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* [44, 46] for achieving backward compatibility with existing currencies, especially privacy-preserving currencies like Monero for the *first time*. With the aid of techniques from [44, 46], the transactions in our protocol look exactly the same as any other regular transaction in the currency, thereby ensuring high *fungibility*. If the

Table 1: Comparison among payment channel approaches. We do not consider [14, 22] as they rely on third-party committees with additional trust assumptions. 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 [24], 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 [25], d is the number of payments made in the channel. LRS refers to Linkable Ring Signature scheme used in Monero [46], and DLSAG refers to the transaction scheme proposed in [39].

	Bi-directional	Pre-schedule online	Unrestricted lifetime	Unbounded payments	Script requirements[1]
Spillman [43]	✗	✓	✗	✓	DS
CLTV [47]	✗	✓	✗	✓	DS + CLTV
Duplex [25]	✓	✓[2]	✗	✗	DS + CLTV
Eltoo [24]	✓	✗	✓	✓	DS + CSV + SIGHASH_NOINPUT + Seq number
Lightning [6]	✓	✗	✓	✓	DS + CSV
Generalized [13]	✓	✗	✓	✓	DS[3] + CSV
Paymo [46]	✗	✓	✗	✓	Monero's LRS + CLTV
DLSAG [39]	✗	✓	✗	✓	DLSAG + CLTV
Teechan [33]	✓	✓	✓	✓	DS + TEE
This work	✓	✓	✗	✓	DS + CLTV
This work+[44]	✓	✓	✗	✓	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 [13] capability.

currency already supports `checkLockTimeVerify` (CLTV) script[2], then our protocol only requires signature generation.

We formally prove the security of our Sleepy Channels protocol in the *Universal Composability (UC)* [20] framework. For this, we design an ideal functionality (in Section 4) that captures a bi-directional payment channel with the same security and efficiency guarantees as the functionality from [13], except that we achieve *delayed finality with punish*. This notion guarantees that until some time **T**, an honest party can receive coins according to either the latest payment state or all the coins from the channel (if the other misbehaves). Due to space constraints, the formal protocol description and the security analysis in the UC framework can be found in Appendix A.

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 Lightning Network (LN) [6]. We further conduct two simulation experiments. In the first, we measure how much centralized collateral watchtower service providers need to allocate, in order to serve certain percentage of the LN. We analyze watchtower proposals that fully collateralize the channels, e.g., [16, 37, 38]. For 30% of the LN, this amounts to around 890 BTC (or roughly 39M USD) of collateral. For Sleepy Channels on the other hand, the collateral is distributed, without the need of a central entity owning this amount of money. In the second experiment, we measure the channels at risk of having their funds stolen given a chance of failing to come online once a day over a given time period. Using LN channels over a one month period, for a chance of 0.1% there are 5k channels at risk, for a chance of 1% there are 49k channels are at risk (roughly 60% of LN). For Sleepy Channels over the same period, there are around 97% fewer channels at risk.

1.2 Related Work

Below, we discuss and compare other prior works that are relevant to our work.

Comparison to other payment channel protocols. CLTV [47] and Spillman [43] 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 [25] 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 [24] 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 [6] are the most popular channels currently in use that support bi-directional payments but require relative timelocks (CSV). Generalized channels [13] support bi-directional payments but again require relative timelocks (CSV). More importantly they require the underlying signature scheme to support adaptor signatures [13] capability[3]. Paymo [46] and DLSAG [39] are proposals tailored for Monero that only support uni-directional

[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 [29], that includes signature schemes like BLS.

payments. Teechan [33] 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* [22, 27, 28] and require complex scripts like *smart contracts* and are incompatible with UTXO-based currencies. Bi-directional payments can also be realized making use of the smart contract support of a third ledger (like Ethereum) via *ZK-Rollups* [12]. However the solutions available are far from ideal either due to high computational costs off the chain or high costs on-chain in terms of gas costs or transaction fees [2]. Moreover, zk-rollups rely on a coordinator for liveness, meaning that if the coordinator goes offline, every user must submit a punishment transaction on-chain, which is costly, effectively closes all channels and largely increases the overhead on-chain. Finally, such a coordinator is in the position to observe every single transaction between any two users (thus largely limiting their privacy) and decide whether to process such transactions or censor them instead.

Additionally, the payment channel proposals can be compared based on the number of transactions it requires to close a channel. For comparable security, we consider the prior payment channel protocols to be supported by the state-of-the-art watchtower proposal [38]. We have that when parties are honest and trustful of each other, prior works require 2 transactions to close a channel (one to close the channel and one for watchtower collateral), while Sleepy Channels requires only 1 transaction. In case where parties are honest and distrustful, prior works require in total 3 transactions to close the channel same as Sleepy Channels if parties wish for a fast closure. Notable exception is Duplex which requires $\log\ d$. In case where parties are dishonest and in the worst case, prior works like Duplex requires $\log d$, Eltoo requires 5, Lightning requires 4 and generalized channels requires 5 transactions in total. Sleepy channels on the other hand requires only 3 transactions in total. Here total refers to the total number of transactions to misbehave, punish and close the channel.

Advantages over Watchtowers. As discussed above, parties may avail the services of a third party like a *watchtower*. Monitor [11] is a watchtower proposal requiring no special scripts. However, an offline watchtower is not penalised and may even get rewarded if a revoked payment is successful on-chain. DCWC [15] 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 [32] requires a OP_RETURN script and also requires the channel user (hiring the watchtower) to pay the watchtower for every channel update. The OP_RETURN script (available in Bitcoin) is used to enter arbitrary information of limited size into a transaction. This however increases the size of the transaction thus requiring a transaction higher fee, and also affects the fungibility of the coins involved in the transaction. PISA [37] 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 [16] and FPPW [38] 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. To incentivize watchtowers, the above protocols require the users to pay a one-time or a persistent fee to the watchtower *even* if the users behaved honestly. On the other hand, users of Sleepy Channels do not lose any coins under honest behaviour as they are guaranteed to get back their collateral.

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 (LN) [6], illustrating its limitations and, based on that, gradually introducing our solution. Our solution consists of a base solution that removes the need for users to constantly be online and an optional extension which aims to disincentivize users blocking funds in the case that they are online.

Lightning channels. Two parties A and B lock up some money in a joint address (or channel) Ch_{AB}, as described in Figure 1. They can perform payments to each other by exchanging payment transactions tx_{Pay}, which commit to an updated balance of both users, v_A for A and v_B for B in this case. Party A gets a signed transaction tx^A_{Pay}, while party B gets a signed transaction tx^B_{Pay}, both of which reflect the above payment state. 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 (revoked) state. To give precedence to the punishment transaction, if party A posts tx^A_{Pay}, it is forced to wait for a relative timelock of Δ (in practice, one day) until it can spend the balance v_A, in order to give time to the other party B to punish. Notice that party B can spend its balance v_B immediately after tx^A_{Pay} is posted. On the other hand, we have the analogous case for party B with the transaction tx^B_{Pay}.

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.

Attempt to remove relative timelock. To drop the requirement for users to constantly be online, an attempt is to replace the relative timelock of Δ time units in Figure 1 with an absolute timelock until time $\mathbf{T}$. This is done 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}

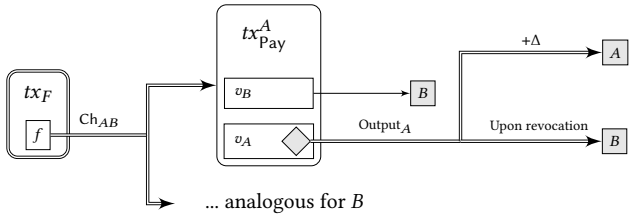

Figure 1: The transaction flow of LN channel between A and B. Rounded boxes represent transactions, rectangles within represent outputs of the transaction: here $v_A + v_B = f$. Incoming arrows represent transaction inputs, while outgoing arrows represent how an output can be spent. Double lines from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

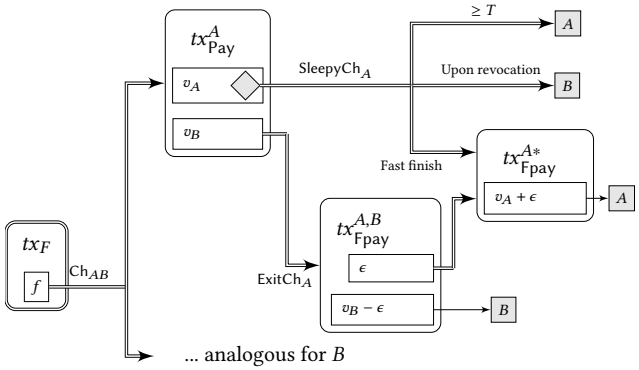

Figure 2: Transaction flow of our base solution. Here double lines from transaction outputs indicate that the output is a 2-party shared address between A and B. A single line from the transaction output indicates that the output is a single party address. We have $v_A + v_B = f$ and ϵ is some negligible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

has to wait until time **T** (irrespective of when tx^A_{Pay} is posted on the chain) before it can retrieve the funds v_A. This allows B to safely go offline during the channel lifetime and only come back shortly before **T** to check if an old state was posted by A. We note that this is completely symmetric: A can safely go offline until shortly before **T** and then check whether or not B has posted an old state tx^B_{Pay}. However, this naive attempt punishes honest parties that wish to close their channels. That is, in the case where an honest party, w.l.o.g. say A, posts the latest state, it still needs to wait until time **T** before having access to its funds v_A. This which could be undesirable as **T** could span several weeks.

Counter-party Confirmation. While it is true that B (the counter-party) can safely go offline until shortly before **T**, this is of course optional and one could think of cases where B is not offline. In the case that B is online, B can go ahead and confirm that A did not misbehave, i.e., A posted the latest state. So if B is online and decides to retrieve its funds v_B (thereby implicitly confirming the state dictated by tx^A_{Pay}), A's funds should be automatically unlocked as well.

We can implement this improvement as shown in Figure 2 where the counter-party B can confirm a payment transaction thus enabling party A to immediately retrieve its funds v_A and not wait until **T**. To do this, after A posts the state tx^A_{Pay}, B has the option (in the case B is online) to post the transaction $tx^{A,B}_{\text{Fpay}}$ (along with a signature on it) which lets A unlock its funds immediately by means of posting the transaction tx^{A*}_{Fpay} (along with a signature on it). Another way to think of this is that by unlocking B's funds using $tx^{A,B}_{\text{Fpay}}$, B gives a confirmation that tx^A_{Pay} is indeed the latest state. On a technical level, the parties A and B would create a fast unlock transaction tx^{A*}_{Fpay} that can be spent if B puts its transaction $tx^{A,B}_{\text{Fpay}}$, using an output thereof as input. With this improvement, A's money v_A either stays locked until **T** if B is offline or A's money becomes unlocked as soon as B spends its output v_B, in case B is online before **T**.

2.1 Extension: Incentivizing a fast unlock

In the above solution, note that the balance v_B 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 (until **T**) and opens the door to *Denial-of-Service* (DoS) attacks from B.

To avoid a situation where B is online, but has no or little incentive to unlock its funds and thereby let A unlock its channel balance early, we add the following extension (as described in Figure 3). To add an incentive for B to unlock early, we let B add a collateral of amount c. For simplicity, let c be equal to the channel capacity f. B's collateral is locked in such a way 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 that is online and attempts to perform a DoS attack on A, ends up locking at least as many coins from itself until **T**. Also note that for the case where A posts an old state, B can first punish and then immediately unlock his coins plus collateral. Analogously, A puts the same amount c as a collateral for the symmetric case. Later in Section 5.2, we discuss scenarios where the two parties may lock different amounts of collateral each.

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. We note that setting $c = 0$ yields the base solution. 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

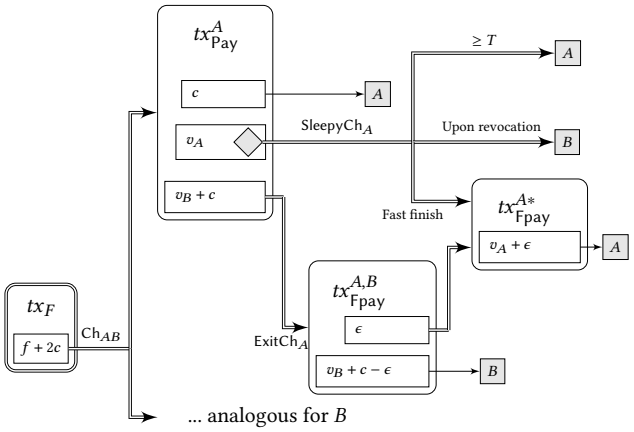

Figure 3: Transaction flow of the extension to our protocol. Again, $v_A + v_B = f$ and ϵ is some negligible amount of coins. The collateral c can be chosen as a value $0 \leq c \leq f$. For $c = 0$, we get Figure 2.

fast confirmation. In a final improvement, we resolve this issue by refining the transaction tx^{A}_{Pay} so that A gets back its part of the collateral immediately. This is safe since the collateral serves merely the purpose of incentivizing the counter-party (in this case B), to acknowledge that the transaction indeed corresponds to the latest channel state. Note that the posting party A only unlocks its collateral right away and not its channel balance set by tx^{A}_{Pay}. Indeed, in the extreme case, if A posts tx^{A}_{Pay} on the chain, A can redeem its collateral c 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 constructions (Figure 2 and Figure 3), we indeed manage to achieve bidirectional channels with unbounded payments without the need for users to constantly be online and monitor the blockchain. We offer our base solution and our extension, that puts an additional incentive on the other user to confirm states early in the case they happen to be online. However, in both solutions they can safely go offline and can come back only shortly before the pre-defined lifetime **T** of the channel. Further, our construction requires only digital signatures and absolute timelocks in the form of CLTV.

We wish to emphasise that a similar requirement of A (or B) going online shortly before **T** is present even in the watchtower proposals [11, 15, 16, 32, 37, 38]. In that case, the hiring (channel) user Bob, is required to come online at a specific point in time **T** to check if the watchtower performed according to the protocol specification. That is, check if the watchtower indeed punished a misbehaving A correctly.

Timelock Independence and Compatibility. The absolute timelock in the form of CLTV makes the protocol not compatible with currencies like Monero where the CLTV script is not supported. However, the requirement of CLTV script in Sleepy Channels can be removed by making use of timed payments through *verifiable timed signatures (VTS)* [44]. This makes Sleepy Channels applicable in a wider range of currencies as it only requires a digital signature script for cryptographic authentication from the underlying currency. For the case of Monero, making use of a variation of VTS from [46] for linkable ring signatures (instead of a standard digital signature), we can realize timed payments and thus a bi-directional payment channel in the form of Sleepy Channels for the first time. We discuss more details of the same in Section 5.

3 PRELIMINARIES

We denote by $\lambda \in \mathbb{N}$ the security parameter and by $x \leftarrow \mathcal{A}(\text{in}; r)$ 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 [21], which lets us model concurrent executions. We consider a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ that is running the protocol. Further, we assume *static* corruptions, where the adversary $\mathcal{A}$ announces at the beginning which parties it 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. We assume authenticated communication with guaranteed delivery between users, as in $\mathcal{F}_{GDC}$.

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 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 [31]. A stronger notion of strong unforgeability for the signature scheme was shown to be equivalent to the UC formulation of security [17].

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 [20]. In this work, we make use of 2-party signing key generation (Γ_{JKGen}) and 2-party signature generation (Γ_{Sign}) protocols [18, 30, 34].

Blockchain and Transaction Scheme. We assume the existence of an ideal ledger (blockchain) functionality $\mathbb{B}$ [13, 35, 36] 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 [13]. 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 **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 can safely be offline until before **T**, but if it stays online is incentivized to confirm it before **T**, thereby unlocking not only the state but also their collateral c. We present the ideal functionality for our solution with extension and note that setting $c = 0$ yields the functionality for the base solution without collateral.

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, $\gamma.\mathsf{st}$ the list of outputs (addresses and values) in, $\gamma.\mathbf{T} \in \mathbb{R}_{\geq 0}$ defines the lifetime of the channel, and $\gamma.c \in \mathbb{R}_{\geq 0}$ 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 (MESSAGE-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 simulator $\mathcal{S}$. Note that $\mathcal{F}$ cannot create signatures or prepare transaction ids. It expects $\mathcal{S}$ 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 $\mathcal{S}$ to provide or set a value, but $\mathcal{S}$ 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 [13], 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 4. 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 (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 CREATED is sent to both parties.

Update. One party P initiates the update with (UPDATE, 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 SETUP–OK and Q not sending UPDATE–OK. When P receives UPDATE–OK, they move on to the revocation. $\mathcal{F}$ expects a message REVOKE from both parties, and in the success case, 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 (CLOSE, 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 a transaction representing the latest state of the channel or an older state, followed by a punishment (see Punish). If the funding transaction remains unspent, outputs ERROR.

Punish. To give honest parties the guarantee that either the most recent state of the channel which is locked until at most time **T** can be enforced on $\mathbb{B}$, or the honest party can get all coins (minus the other party's collateral), 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, 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 **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 ϵ (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 **T** has passed. We discuss in Section 5.2 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 [1, 19] for transaction authentication, we can instantiate Γ_{JKGen} and Γ_{Sign} with protocols from [34], [30], or [18], respectively. Monero uses a linkable ring signature scheme [39, 46] for authentication and the corresponding tailored 2-party protocols for key generation and signing are described in [46].

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 Figure 5.

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

5.1.1 *High Level Overview.*

We present below the intuition for our protocol in prose and refer to Figure 3 in Section 2 for the transaction flow of the construction.

Payment. For each payment from the channel Ch_{AB}, parties generate two versions of transactions, tx^{A}_{Pay} and tx^{B}_{Pay}, 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 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 **T**.
(2) Transaction tx^{A*}_{Fpay}, that spends from $\mathsf{SleepyCh}_A$ and an auxiliary address aux_A (contains ϵ coins as output in $tx^{A,B}_{\mathsf{Fpay}}$, see below) 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 $v_B + c - \epsilon$ 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.

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

Create: Upon $(\texttt{CREATE}, \gamma, tid_A) \overset{\tau_0}{\hookleftarrow} A$, distinguish:

Both agreed: If already received $(\texttt{CREATE}, \gamma, tid_B) \overset{\tau}{\hookleftarrow} B$, 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 $(\texttt{CREATED}, \gamma.\mathsf{id}) \overset{\tau_1}{\hookrightarrow} \gamma.\mathsf{users}$. Else stop.

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

Update: Upon $(\texttt{UPDATE}, \mathsf{id}, \overrightarrow{\theta}, t_{\mathsf{stp}}) \overset{\tau_0}{\hookleftarrow} A$, 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 $(\texttt{UPDATE-REQ}, \mathsf{id}, \overrightarrow{\theta}, t_{\mathsf{stp}}, \overrightarrow{tid}) \overset{\tau_1}{\hookrightarrow} B$ and $(\texttt{SETUP}, \mathsf{id}, \overrightarrow{tid}) \overset{\tau_1}{\hookrightarrow} A$.
(2) If $(\texttt{SETUP-OK}, \mathsf{id}) \overset{\tau_2 \leq \tau_1 + t_{\mathsf{stp}}}{\longleftarrow} A$, then $(\texttt{SETUP-OK}, \mathsf{id}) \overset{\tau_3 \leq \tau_2 + T_p}{\longrightarrow} B$. Else stop.
(3) If $(\texttt{UPDATE-OK}, \mathsf{id}) \overset{\tau_3}{\hookleftarrow} B$, then (if B honest or instructed by $\mathcal{S}$) send $(\texttt{UPDATE-OK}, \mathsf{id}) \overset{\tau_4 \leq \tau_3 + T_p}{\longrightarrow} A$. 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 $(\texttt{REVOKE}, \mathsf{id}) \overset{\tau_4}{\hookleftarrow} A$, send $(\texttt{REVOKE-REQ}, \mathsf{id}) \overset{\tau_5 \leq \tau_4 + T_p}{\longrightarrow} B$. Else set $\Gamma(\mathsf{id}) := (\{\gamma, \gamma'\}, \mathsf{Ch}_{AB})$, run $\texttt{ForceClose}(\mathsf{id})$ and stop.
(5) If $(\texttt{REVOKE}, \mathsf{id}) \overset{\tau_5}{\hookleftarrow} B$, $\Gamma(\mathsf{id}) := (\{\gamma'\}, \mathsf{Ch}_{AB})$, send $(\texttt{UPDATED}, \mathsf{id}, \overrightarrow{\theta}) \overset{\tau_6 \leq \tau_5 + T_p}{\longrightarrow} \gamma.\mathsf{users}$ and stop *(accept)*. Else set $\Gamma(\mathsf{id}) := (\{\gamma, \gamma'\}, \mathsf{Ch}_{AB})$, run $\texttt{ForceClose}(\mathsf{id})$ and stop.

Close: Upon $(\texttt{CLOSE}, \mathsf{id}) \overset{\tau_0}{\hookleftarrow} A$, distinguish

Both agreed: If already received $(\texttt{CLOSE}, \mathsf{id}) \overset{\tau}{\hookleftarrow} B$, 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 $\Gamma(\mathsf{id}) := \bot$, send $(\texttt{CLOSED}, \mathsf{id}) \overset{\tau_1}{\hookrightarrow} \gamma.\mathsf{users}$ and stop.
• Else, if at least one of the parties is not honest, run $\texttt{ForceClose}(\mathsf{id})$. Else, output $(\texttt{ERROR}) \overset{\tau_0 + \Delta}{\longrightarrow} \gamma.\mathsf{users}$ and stop.

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

Punish: (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 $\gamma.\mathsf{st.bal}(B) + \gamma.c - \epsilon$, appears on $\mathbb{B}$, set $\Gamma(\mathsf{id}) = \bot$, send $(\texttt{PUNISHED}, \mathsf{id}) \overset{\tau_2}{\hookrightarrow} B$ and stop.

Close: Either $\Gamma(\mathsf{id}) = \bot$ 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 $\gamma.\mathsf{st.bal}(B) + \gamma.c - \epsilon$, appears on $\mathbb{B}$ before a transaction $tx^{A*}_{\mathsf{Fpay},i} := ([o \in o_C, o' \in o_S], o_F, \gamma.\mathsf{st.bal}(A) + \epsilon)$ where address o_F of A appears on $\mathbb{B}$. Set $\Gamma(\mathsf{id}) := \bot$ and send $(\texttt{CLOSED}, \mathsf{id}) \overset{\tau_2 \leq \tau_1 + \Delta}{\longrightarrow} \gamma.\mathsf{users}$. 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 $\Gamma(\mathsf{id}) := \bot$ and $(\texttt{CLOSED}, \mathsf{id}) \overset{\tau_3}{\hookrightarrow} \gamma.\mathsf{users}$ and stop.

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

Subprocedure $\texttt{ForceClose}(\mathsf{id})$: 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 $(\texttt{ERROR}) \overset{\tau_0 + \Delta}{\longrightarrow} \gamma.\mathsf{users}$ and stop. Else, latest in round $\gamma.\mathbf{T} + \Delta$, $m \in \{\texttt{CLOSED}, \texttt{PUNISHED}, \texttt{ERROR}\}$ is output via Punish.

Figure 4: Ideal Functionality

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 $v_B + c - \epsilon$ 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 $v_B + c - \epsilon$. In effect, A only gets its collateral back, while B is able to retrieve the entire payment capacity f and its own collateral c.

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 $\mathbf{T}$.

(2) Generate another set of faster finish-pay transactions $tx^{A,B}_{\mathsf{Fpay},i} := tx\left(\mathsf{ExitCh}_A, [pk_{\mathsf{ffp},B}, \mathsf{aux}_A], [v_{B,i} + c - \epsilon, \epsilon]\right)$ and $tx^{B,A}_{\mathsf{Fpay},i} := tx\left(\mathsf{ExitCh}_B, [pk_{\mathsf{ffp},A}, \mathsf{aux}_B], [v_{A,i} + c - \epsilon, \epsilon]\right)$.

(3) Generate a set of enabler transactions $tx^{A*}_{\mathsf{Fpay},i} := tx\left([\mathsf{SleepyCh}_A, \mathsf{aux}_A], pk_{\mathsf{fp},A}, v_{A,i} + \epsilon\right)$ and $tx^{B*}_{\mathsf{Fpay},i} := tx\left([\mathsf{SleepyCh}_B, \mathsf{aux}_B], pk_{\mathsf{fp},B}, v_{B,i} + \epsilon\right)$ 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}.

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 - Payment setup, payments, closing and punishment

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 $\mathcal{F}$.*

5.2 Discussion

In this section we discuss key aspects about our collateral requirement and describe extensions of our protocol that makes it applicable in a wider class of settings.

Collateral as incentive. Observe that the collateral of the party initiating the closing is retrieved by that party during closing, irrespective of a cheating event. This is because the purpose of the collateral in the Sleepy Channels protocol is to incentivize fast closure of the channel by the other party if one of the parties wishes to close the channel and the other party happens to be online. 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 $v_B + c - \epsilon$ 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 $v_A \leq v_B + c - \epsilon$ when $\epsilon \approx 0$. 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 at least the same amount of coins until timeout $\mathbf{T}$.

Asymmetric collateral. Consider the case where A has significantly more money than party B (e.g., A is a merchant and B is one of A's customers). In this case, party A may be able to easily afford to lock a collateral value c (same as B) to prevent party B from getting its coins back before time $\mathbf{T}$. To account for this apparent disparity in the financial strength between parties A and B, we can instantiate our Sleepy Channels with both parties locking different amounts of collateral. In our example, party A and party B open their channel in such a manner that A locks collateral amount c_A that is higher than the collateral amount c_B locked by party B. c_A could theoretically even be larger than the full channel capacity. This strongly discourages party A (i.e., more than when using smaller or equal collateral to that of B) to deny party B a fast channel closure. We note that our Sleepy Channels protocol is flexible in how the parties set each other's collateral before opening their channel.

Punishment cost. Note that if party A misbehaves and posts a revoked payment on the chain, party B has until time $\mathbf{T}$ to punish this behaviour on the chain. It is possible that the punishment transaction posted by B costs more in terms of transaction fee than what it stands to gain after the punishment if for example, the revoked payment is a very small coin transfer. To account for that, A (i.e., the party creating and funding the channel) can unconditionally include a certain amount for B to cover such transaction fees, as it is currently implemented in the Lightning Network [26]. We emphasize that this is an issue that is present throughout off-chain solutions [6, 13, 24, 25] including ZK-rollups [12].

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 5) 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 [41], we can avoid the dependence from timelock scripts entirely. As it was shown in prior works [44, 46], 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 [46]. A recent work [45] 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 $tx^{A,A}_{\mathsf{Fpay},i}$

[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.

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 other words, parties can post a single transaction on the blockchain anytime before $\mathbf{T}$ 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 [42] 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 show (i) correctness of our scheme, (ii) compatibility with Bitcoin, and (iii) on- and off-chain transaction overhead. The source code is available at [4].

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 done via exchanging a hash secret, a private key or a signed punishment transaction upon revoking an old state. In this section, we follow our protocol as in Figure 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 [3]. Built on top of the funding, we look at A's commitment (or state) transaction $tx^{A}_{\mathsf{Pay},i}$ [9] 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 $tx^{A,B}_{\mathsf{Fpay},i}$ [10], then A can claim its funds with $tx^{A*}_{\mathsf{Fpay},i}$ right away [8]. 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 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 (January 2022). In this snapshot, there are 81k channels, 19k channel nodes and a total capacity of 2990 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 including the snapshot is available at [5]. 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. We analyze watchtower constructions which fully collateralize the channels, e.g., [16, 37, 38]. 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 890 BTC and (ii) users needs to pay fees for that, even if there are no disputes. Currently, this total capacity that has to be available to the watchtower service as collateral amounts to roughly 39M USD.

Risk of failing to go online. We simulate the risk of users having to periodically monitor the blockchain in LN. In LN, the time frame for punishment is one day (144 blocks). I.e., in this time users need to come online at least once and check whether or not the other party tried to cheat. In our setting, we investigate a time period of 30 days with users trying to come online each day.

In our simulation, we assume that there is a certain chance that users fail to come online and monitor the blockchain in a given time frame, e.g., due to power outages, DoS attacks, etc. We further assume that neighboring nodes will notice this; a realistic assumption due to the *ping and pong* messages [7] 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.

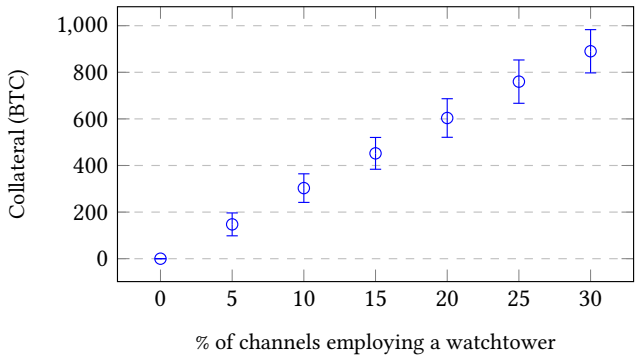

Figure 6: Results of the first simulation.

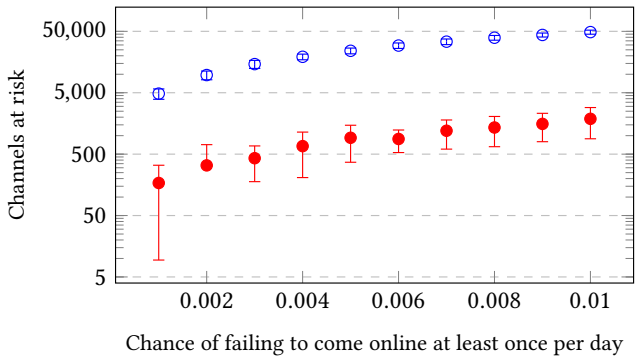

Figure 7: Results of the second simulation. (Blue = LN, Red = Sleepy Channels)

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, e.g., 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 chance 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. Over a one month period, there are 5k channels (0.1% chance) and 49k (1% chance) channels are at risk (roughly 60% of the LN) for LN channels. For Sleepy Channels, these numbers are 170 channels (0.1% chance) and 1.9k channels (1% chance).

7 CONCLUSION

Payment channels are one of the most promising payment solutions for blockchain-based cryptocurrencies. Despite their large adoption, many such proposals 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.

Acknowledgments

This work has been also partially supported by Madrid regional government as part of the program S2018/TCS-4339 (BLOQUES-CM) co-funded by EIE Funds of the European Union, by grant IJC2020-043391-I/MCIN/AEI/10.13039/501100011033 and European Union NextGenerationEU/PRTR, by SCUM Project (RTI2018-102043-B-I00) MCIN/AEI/10.13039/501100011033/ERDF A way of making Europe, and by the project HACRYPT. This work was also partially supported 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), by the Austrian Research Promotion Agency (FFG) through 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) and by CoBloX Labs. This work was further partially supported by the German Federal Ministry of Education and Research BMBF (grant 16K15K042, project 6GEM). The work was also funded through the support of THE DAVID AND LUCILLE PACKARD FOUNDATION - Award #202071730, SRI INTERNATIONAL - Award #53978 / Prime: DEFENSE ADVANCED RESEARCH PROJECTS AGENCY - Award #HR00110C0086, NATIONAL SCIENCE FOUNDATION - Award #2212746.

REFERENCES

[1] [n. d.]. Chia Network FAQ. https://www.chia.net/faq/.
[2] [n. d.]. Cryptographic Frontier, Open Problems in Ethereum Research. https://sites.google.com/view/cryptofrontier21.
[3] [n. d.]. Funding transaction of our evaluation on the Bitcoin testnet. https://tinyurl.com/589xku8w.
[4] [n. d.]. Github repository of our Sleepy Channels evaluation. https://github.com/sleepy-channels/overhead.
[5] [n. d.]. Github repository of our Sleepy Channels simulation. https://github.com/sleepy-channels/simulation.
[6] [n. d.]. Lightning Network. https://lightning.network/.
[7] [n. d.]. 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.
[8] [n. d.]. Pay A star transaction of our evaluation on the Bitcoin testnet. https://tinyurl.com/bskz7fvx.
[9] [n. d.]. Pay A transaction of our evaluation on the Bitcoin testnet. https://tinyurl.com/2w6aebr9.
[10] [n. d.]. Pay A,B transaction of our evaluation on the Bitcoin testnet. https://tinyurl.com/2uwn5fvb.
[11] [n. d.]. Unlinkable Outsourced Channel Monitoring. https://diyhpl.us/wiki/transcripts/scalingbitcoin/milan/unlinkable-outsourced-channel-monitoring/.
[12] [n. d.]. ZK-Rollup. https://tinyurl.com/yc2n79r5.

[13] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. 2021. Generalized Channels from Limited Blockchain Scripts and Adaptor Signatures. *AsiaCrypt* (2021). https://eprint.iacr.org/2020/476

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

[15] Georgia Avarikioti, Felix Laufenberg, Jakub Sliwinski, Yuyi Wang, and Roger Wattenhofer. 2018. Towards Secure and Efficient Payment Channels. arXiv:1811.12740 [cs.CR]

[16] Zeta Avarikioti, Orfeas Stefanos Thyfronitis Litos, and Roger Wattenhofer. 2020. Cerberus Channels: Incentivizing Watchtowers for Bitcoin. In *FC 2020 (LNCS, Vol. 12059)*, Joseph Bonneau and Nadia Heninger (Eds.). Springer, Heidelberg, 346–366. https://doi.org/10.1007/978-3-030-51280-4_19

[17] Michael Backes and Dennis Hofheinz. 2004. How to Break and Repair a Universally Composable Signature Functionality. In *ISC 2004 (LNCS, Vol. 3225)*, Kan Zhang and Yuliang Zheng (Eds.). Springer, Heidelberg, 61–72.

[18] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. Compact Multi-signatures for Smaller Blockchains. In *ASIACRYPT 2018, Part II (LNCS, Vol. 11273)*, Thomas Peyrin and Steven Galbraith (Eds.). Springer, Heidelberg, 435–464. https://doi.org/10.1007/978-3-030-03329-3_15

[19] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In *ASIACRYPT 2001 (LNCS, Vol. 2248)*, Colin Boyd (Ed.). Springer, Heidelberg, 514–532. https://doi.org/10.1007/3-540-45682-1_30

[20] Ran Canetti. 2000. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology* 13, 1 (Jan. 2000), 143–202. https://doi.org/10.1007/s001459910006

[21] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. 2007. Universally Composable Security with Global Setup. In *TCC 2007 (LNCS, Vol. 4392)*, Salil P. Vadhan (Ed.). Springer, Heidelberg, 61–85. https://doi.org/10.1007/978-3-540-70936-7_4

[22] Manuel M. T. Chakravarty, Sandro Coretti, Matthias Fitzi, Peter Gazi, Philipp Kant, Aggelos Kiayias, and Alexander Russell. 2020. Hydra: Fast Isomorphic State Channels. *IACR Cryptol. ePrint Arch.* 2020 (2020), 299.

[23] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. 2019. SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)*. 142–157. https://doi.org/10.1109/EuroSP.2019.00020

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

[25] Christian Decker and Roger Wattenhofer. 2015. 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 (Lecture Notes in Computer Science, Vol. 9212)*, Andrzej Pelc and Alexander A. Schwarzmann (Eds.). Springer, 3–18. https://doi.org/10.1007/978-3-319-21741-3_1

[26] LN developers. [n. d.]. BOLT #2: Peer Protocol for Channel Management. https://github.com/lightning/bolts/blob/master/02-peer-protocol.md.

[27] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. 2019. Perun: Virtual Payment Hubs over Cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 106–123. https://doi.org/10.1109/SP.2019.00020

[28] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. 2018. General State Channel Networks. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 949–966. https://doi.org/10.1145/3243734.3243856

[29] Andreas Erwig, Sebastian Faust, Kristina Hostáková, Monosij Maitra, and Siavash Riahi. 2021. Two-Party Adaptor Signatures from Identification Schemes. In *PKC 2021, Part I (LNCS, Vol. 12710)*, Juan Garay (Ed.). Springer, Heidelberg, 451–480. https://doi.org/10.1007/978-3-030-75245-3_17

[30] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In *EUROCRYPT'99 (LNCS, Vol. 1592)*, Jacques Stern (Ed.). Springer, Heidelberg, 295–310. https://doi.org/10.1007/3-540-48910-X_21

[31] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. 1988. A Digital Signature Scheme Secure Against Adaptive Chosen-message Attacks. *SIAM J. Comput.* 17, 2 (April 1988), 281–308.

[32] Majid Khabbazian, Tejaswi Nadahalli, and Roger Wattenhofer. 2019. Outpost: A Responsive Lightweight Watchtower. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies* (Zurich, Switzerland) *(AFT '19)*. Association for Computing Machinery, New York, NY, USA, 31–40. https://doi.org/10.1145/3318041.3355464

[33] Joshua Lind, Ittay Eyal, Peter R. Pietzuch, and Emin Gün Sirer. 2016. Teechan: Payment Channels Using Trusted Execution Environments. *CoRR* abs/1612.07766 (2016). arXiv:1612.07766 http://arxiv.org/abs/1612.07766

[34] Yehuda Lindell. 2017. Fast Secure Two-Party ECDSA Signing. In *CRYPTO 2017, Part II (LNCS, Vol. 10402)*, Jonathan Katz and Hovav Shacham (Eds.). Springer, Heidelberg, 613–644. https://doi.org/10.1007/978-3-319-63715-0_21

[35] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. 2017. Concurrency and Privacy with Payment-Channel Networks. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 455–471. https://doi.org/10.1145/3133956.3134096

[36] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. 2019. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In *NDSS 2019*. The Internet Society.

[37] Patrick McCorry, Surya Bakshi, Iddo Bentov, Sarah Meiklejohn, and Andrew Miller. 2019. Pisa: Arbitration Outsourcing for State Channels. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies* (Zurich, Switzerland) *(AFT '19)*. Association for Computing Machinery, New York, NY, USA, 16–30. https://doi.org/10.1145/3318041.3355461

[38] Arash Mirzaei, Amin Sakzad, Jiangshan Yu, and Ron Steinfeld. 2021. FPPW: A Fair and Privacy Preserving Watchtower For Bitcoin. Cryptology ePrint Archive, Report 2021/117. https://ia.cr/2021/117.

[39] Pedro Moreno-Sanchez, Arthur Blue, Duc V. Le, Sarang Noether, Brandon Goodell, and Aniket Kate. 2020. DLSAG: Non-interactive Refund Transactions for Interoperable Payment Channels in Monero. In *Financial Cryptography and Data Security*, Joseph Bonneau and Nadia Heninger (Eds.). Springer International Publishing, Cham, 325–345.

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

[41] R. L. Rivest, A. Shamir, and D. A. Wagner. 1996. Time-lock Puzzles and Timed-release Crypto.

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

[43] Jeremy Spillman. [n. d.]. Spillman-style payment channels. https://tinyurl.com/uwzfb2tu.

[44] Sri Aravinda Krishnan Thyagarajan, Adithya Bhat, Giulio Malavolta, Nico Döttling, Aniket Kate, and Dominique Schröder. 2020. Verifiable Timed Signatures Made Practical. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 1733–1750. https://doi.org/10.1145/3372297.3417263

[45] Sri Aravinda Krishnan Thyagarajan, Tiantian Gong, Adithya Bhat, Aniket Kate, and Dominique Schröder. 2021. OpenSquare: Decentralized Repeated Modular Squaring Service. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 3447–3464. https://doi.org/10.1145/3460120.3484809

[46] Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, Fritz Schmidt, and Dominique Schröder. 2020. PayMo: Payment Channels For Monero. Cryptology ePrint Archive, Report 2020/1441. https://eprint.iacr.org/2020/1441.

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

[48] Jo Van Bulck, David Oswald, Eduard Marin, Abdulla Aldoseri, Flavio D. Garcia, and Frank Piessens. 2019. 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* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 1741–1758. 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 [13], 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

Create

Party A upon $(\texttt{CREATE}, \text{id}, \gamma, tid_A) \xleftarrow{t_0} \mathcal{E}$:

(1) Generate $\left(pk_{\text{CPay},A}, sk_{\text{CPay},A}\right)$, $(pk_{\text{pun},A}, sk_{\text{pun},A})$, $(pk_{\text{fp},A}, sk_{\text{fp},A})$ and $(pk_{\text{ffp},A}, sk_{\text{ffp},A})$. Let $pkey^A_{set}$ be the set of public keys of these key pairs.
(2) Extract $v_{A,0}$ and $v_{B,0}$ from $\gamma.\text{st}$, and $c := \gamma.c$
(3) Send $(\text{createInfo}, \text{id}, tid_A, pkey^A_{set}) \xhookrightarrow{t_0} B$.
(4) If $(\text{createInfo}, \text{id}, tid_B, pkey^B_{set}) \xleftarrow{t_0+1} B$, continue. Else, go idle.
(5) Using $pkey^A_{set}$ and $pkey^B_{set}$, A together with B runs $\mathcal{F}_{\text{JKGen}}$ to generate the following set of shared addresses: $addr_{set} := \{\text{Ch}_{AB}, \text{SleepyCh}_A, \text{SleepyCh}_B, \text{ExitCh}_A, \text{ExitCh}_B, \text{aux}_A, \text{aux}_B\}$ which takes t_g rounds. In case of failure, abort.
(6) Generate $tx_f := tx([tid_A, tid_B], [\text{Ch}_{AB}], [2 \cdot c + v_{A,0} + v_{B,0}])$
(7) Let $tx_{set_0} \leftarrow \texttt{GenerateTxs}(addr_{set}, pkey^A_{set}, pkey^B_{set}, c, v_{A_i}, v_{B_i})$
(8) Let $sig_{set_0}{}^A \leftarrow \texttt{SignTxs}^A(tx_{set0}, addr_{set}, pkey^A_{set} \cup pkey^B_{set})$
(9) A generates a signature σ_{tid_A} for the output tid_A and sends $(\text{createFund}, \text{id}, \sigma_{tid_A}) \xhookrightarrow{t_0+1+t_g+t_s} A$.
(10) If $(\text{createFund}, \text{id}, \sigma_{tid_B}) \xleftarrow{t_0+2+t_g+t_s} B$, 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 $\Gamma^A(\text{id}) := (tx_F, tx_{set0}, sig_{set_0}{}^A, addr_{set}, pkey^A_{set}, pkey^B_{set})$ and $(\texttt{CREATED}, \text{id}) \xhookrightarrow{t_1} \mathcal{E}$.

Update

Party A upon $(\texttt{UPDATE}, \text{id}, \overrightarrow{\theta}, t_{\text{stp}}) \xhookleftarrow{t_0} \mathcal{E}$

(1) $(\text{updateReq}, \text{id}, \overrightarrow{\theta}, t_{\text{stp}}) \xhookrightarrow{t_0} B$

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

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

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

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

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

(9) $(\texttt{SETUP-OK}, \text{id}) \xhookrightarrow{\tau_1} \mathcal{E}$
(10) If not $(\texttt{UPDATE-OK}, \text{id}) \xleftarrow{\tau_1} \mathcal{E}$, go idle.
(11) $\texttt{SignTxs}^A(tx_{set\,i}, addr_{set}, pkey^A_{set} \cup pkey^B_{set})$

Party A in round $t_1 + 1 + t_s$

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

Party B in round $\tau_1 + t_s$

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

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

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

Close

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

(1) Extract $(tx_F, tx_{set\,i}, sig_{set}{}^A_i, addr_{set}, pkey^A_{set}, pkey^B_{set})$ from $\Gamma^A(\text{id})$.
(2) Extract $v_{A,i}$ and $v_{B,i}$ from $tx^A_{\text{Pay},j} \in tx_{set\,i}$, and c from tx_F
(3) Create transaction $tx_c := tx(\text{Ch}_{AB}, \{pk_A, pk_B\}, \{v_{A,i} + c, v_{B,i} + c\})$, 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 $\mathcal{F}_{\text{Sign}}$ 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 ForceClose(id).
(6) If tx_c appears on $\mathbb{B}$ in round $t_1 \leq t_0 + t_r + \Delta$, set $\Theta^A(\text{id}) := \bot$, $\Gamma^A(\text{id}) := \bot$ and send $(\text{CLOSED}, \text{id}) \overset{t_2}{\hookrightarrow} \mathcal{E}$.

Punish

Party A upon $\text{PUNISH} \overset{t_0}{\hookleftarrow} \mathcal{E}$:

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

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

Subprotocols

ForceClose(id):

Let t_0 be the current round

(1) Extract $(tx_F, tx_{set0}, sig_{set_0}^A, addr_{set}, pkey^A_{set}, pkey^B_{set})$ from $\Gamma^A(\text{id})$ and extract $tx^A_{\text{Pay},j}$ from tx_{set} and $\sigma^A_{\text{Pay},j}$ and sig_{set}.
(2) Party A posts $\left(tx^A_{\text{Pay},j}, \sigma^A_{\text{Pay},j}\right)$ on $\mathbb{B}$
(3) Let $t_1 \leq t_0 + \Delta$ be the round in which $tx^A_{\text{Pay},j}$ is accepted by $\mathbb{B}$.
(4) If $tx^{A,B}_{\text{Fpay},i}$ appears on $\mathbb{B}$ at or after round $t_2 \leq t_1 + \Delta$ and before $\mathbf{T}$, post $\left(tx^A_{\text{Pay},j}, \sigma^A_{\text{Pay},j}\right)$ and send $(\text{CLOSED}, \text{id}) \overset{t_3 \leq t_2 + \Delta}{\longrightarrow} \mathcal{E}$. Otherwise, post $\left(tx^{A,A}_{\text{Fpay},i}, \sigma^{A,A}_{\text{Fpay},i}\right)$ after $\mathbf{T}$ and send $(\text{CLOSED}, \text{id}) \overset{t_4 \leq \text{T} + \Delta}{\longrightarrow} \mathcal{E}$.
(5) Set $\Gamma^P(\text{id}) := \bot$, $\Theta^P(\text{id}) := \bot$.

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 $tx^A_{\text{Pay},i} := tx(\text{Ch}_{AB}, [pk_{\text{CPay},A}, \text{SleepyCh}_A, \text{ExitCh}_B], [c, v_{A,i}, v_{B,i} + c])$
(3) Generate $tx^B_{\text{Pay},i} := tx(\text{Ch}_{AB}, [pk_{\text{CPay},B}, \text{SleepyCh}_B, \text{ExitCh}_A] [c, v_{B,i}, v_{A,i} + c])$
(4) Generate punishment transactions $tx^A_{\text{Pnsh},i} := tx(\text{SleepyCh}_A, pk_{\text{pun},B}, v_{A,i})$ and $tx^B_{\text{Pnsh},i} := tx(\text{SleepyCh}_B, pk_{\text{pun},A}, v_{B,i})$
(5) Generate finish-pay transactions $tx^{A,A}_{\text{Fpay},i} := tx(\text{SleepyCh}_A, pk_{\text{fp},A}, v_{A,i})$ and $tx^{B,B}_{\text{Fpay},i} := tx(\text{SleepyCh}_B, pk_{\text{fp},B}, v_{B,i})$ both timelocked until time $\mathbf{T}$.
(6) Generate a set of faster finish-pay transactions $tx^{A,B}_{\text{Fpay},i} := tx(\text{ExitCh}_A, [pk_{\text{ffp},B}, \text{aux}_A], [v_{B,i} + c - \epsilon, \epsilon])$ and $tx^{B,A}_{\text{Fpay},i} := tx(\text{ExitCh}_B, [pk_{\text{ffp},A}, \text{aux}_B], [v_{A,i} + c - \epsilon, \epsilon])$.
(7) Generate a set of enabler transactions $tx^{A*}_{\text{Fpay},i} := tx([\text{SleepyCh}_A, \text{aux}_A], pk_{\text{fp},A}, v_{A,i} + \epsilon)$ and $tx^{B*}_{\text{Fpay},i} := tx([\text{SleepyCh}_B, \text{aux}_B], pk_{\text{fp},B}, v_{B,i} + \epsilon)$ that enable a faster finish-payment.
(8) Return $tx_{set} := \{tx^A_{\text{Pay},i}, tx^B_{\text{Pay},i}, tx^A_{\text{Pay},i}, tx^A_{\text{Pnsh},i}, tx^B_{\text{Pnsh},i}, tx^{A,A}_{\text{Fpay},i}, tx^{B,B}_{\text{Fpay},i}, tx^{A,B}_{\text{Fpay},i}, tx^{B,A}_{\text{Fpay},i}, tx^{A*}_{\text{Fpay},i}, tx^{B*}_{\text{Fpay},i}\}$

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 $\mathcal{F}_{\text{Sign}}$ to sign the transactions as follows.

(1) Party A receives signature $\sigma^{A,A}_{\text{Fpay},i}$ on transaction $tx^{A,A}_{\text{Fpay},i}$ under the shared key SleepyCh_A.
(2) Party B receives signature $\sigma^{B,B}_{\text{Fpay},i}$ on transaction $tx^{B,B}_{\text{Fpay},i}$ under the shared key SleepyCh_B.
(3) Party A receives signatures $\left(\sigma_{\text{SleepyCh},A}, \sigma_{\text{aux},A}\right)$ on the transaction $tx^{A*}_{\text{Fpay},i}$ with respect to the shared keys SleepyCh_A and aux_A, respectively.
(4) Party B receives signatures $\left(\sigma_{\text{SleepyCh},B}, \sigma_{\text{aux},B}\right)$ on the transaction $tx^{B*}_{\text{Fpay},i}$ with respect to the shared keys SleepyCh_B and aux_B, respectively.
(5) Party A receives signature $\sigma^{A,B}_{\text{Fpay},i}$ on the transaction $tx^{A,B}_{\text{Fpay},i}$ under the shared key ExitCh_B.
(6) Party B receives signature $\sigma^{B,A}_{\text{Fpay},i}$ on the transaction $tx^{B,A}_{\text{Fpay},i}$ under the shared key ExitCh_A.
(7) Party A receives signature $\sigma^A_{\text{Pay},i}$ on the transaction $tx^A_{\text{Pay},i}$ under the shared key Ch_{AB}.
(8) Party B receives signature $\sigma^B_{\text{Pay},i}$ on the transaction $tx^B_{\text{Pay},i}$ 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 $sig_{set_i}^A := \left\{\sigma^{A,A}_{\text{Fpay},i}, \left(\sigma_{\text{SleepyCh},A}, \sigma_{\text{aux},A}\right), \sigma^{B,A}_{\text{Fpay},i}, \sigma^A_{\text{Pay},i}\right\}$ and to B $sig_{set_i}^B := \left\{\sigma^{B,B}_{\text{Fpay},i}, \left(\sigma_{\text{SleepyCh},B}, \sigma_{\text{aux},B}\right), \sigma^{A,B}_{\text{Fpay},i}, \sigma^B_{\text{Pay},i}\right\}$

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 $\mathcal{F}_{\text{JKGen}}$ and (ii) the 2-party signing protocol Γ_{Sign} for a signature scheme Π_{DS} with an idealized version $\mathcal{F}_{\text{Sign}}$. For the UC formulations we refer the reader to [17, 20]. 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 $\mathcal{F}_{\text{JKGen}}$. 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 $\mathcal{F}_{\text{JKGen}}$, $\mathcal{R}$ using $\mathcal{A}$ can be used to distinguish a keyshare of Γ_{JKGen} from the data received in $\mathcal{F}_{\text{JKGen}}$, 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 $\mathcal{F}_{\text{Sign}}$, 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 $\mathtt{SignTxs}^P$ 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

Case A is honest and B is corrupted

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

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

Do the following:

(1) Set $\text{id} := \gamma.\text{id}$, generate $\left(pk_{\mathsf{CPay},A}, sk_{\mathsf{CPay},A}\right)$, $(pk_{\mathsf{pun},A}, sk_{\mathsf{pun},A})$, $(pk_{\mathsf{fp},A}, sk_{\mathsf{fp},A})$ and $(pk_{\mathsf{ffp},A}, sk_{\mathsf{ffp},A})$. Let $pkey^A_{set}$ be the set of public keys of these key pairs. Send $(\text{createInfo}, \text{id}, tid_A, pkey^A_{set}) \overset{\tau_0}{\hookrightarrow} B$.
(2) If you receive $(\text{createInfo}, \text{id}, tid_B, pkey^B_{set}) \overset{\tau_0+1}{\hookleftarrow} B$, 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 $tx_{set_0} \leftarrow \mathtt{GenerateTxs}(addr_{set}, pkey^A_{set}, pkey^B_{set}, c, v_{A_i}, v_{B_i})$
(6) Let $sig^A_{set_0} \leftarrow \mathtt{SignTxs}^{\mathcal{A}}(tx_{set\,0}, addr_{set}, pkey^A_{set} \cup pkey^B_{set})$
(7) Generates a signature on behalf of A, σ_{tid_A}, for the output tid_A and send $(\text{createFund}, \text{id}, \sigma_{tid_A}) \overset{t_0+1+t_g+t_s}{\hookrightarrow} A$.
(8) If you $(\text{createFund}, \text{id}, \sigma_{tid_B}) \overset{\tau_0+2+t_g+t_s}{\hookleftarrow} B$, 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 $\Gamma^A(\text{id}) := (tx_F, tx_{set0}, sig^A_{set_0}, addr_{set}, pkey^A_{set}, pkey^B_{set})$.

Simulator for Update

Case A is honest and B is corrupted

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

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

Case B is honest and A is corrupted

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

(1) Upon $(\text{updateReq}, \text{id}, \overrightarrow{\theta}, t_{\text{stp}}) \overset{\tau_0}{\hookleftarrow} A$, do the following
(2) Retrieve $(tx_F, tx_{set\,i-1}, sig^B_{set\,i-1}, addr_{set}, pkey^A_{set}, pkey^B_{set}) = \Gamma^B(\text{id})$
(3) Extract $v_{A,i}$ and $v_{B,i}$ from $\overrightarrow{\theta}$, and c from tx_F
(4) Let $tx_{set\,i} \leftarrow \mathtt{GenerateTxs}(addr_{set}, pkey^A_{set}, pkey^B_{set}, c, v_{A_i}, v_{B_i})$
(5) Let $\overrightarrow{tid} := (tx^A_{\mathsf{Pay},i}.\text{id}, tx^B_{\mathsf{Pay},i}.\text{id})$ be a tuple of the transaction ids of transaction $tx^A_{\mathsf{Pay},i}$ and $tx^B_{\mathsf{Pay},i}$. Inform $\mathcal{F}$ of $\overrightarrow{tid}$.
(6) $(\text{updateInfo}, \text{id}) \overset{\tau_0}{\hookrightarrow} A$
(7) Upon A sending $(\text{updateCom}, \text{id}) \overset{\tau_0+1+t_{\text{stp}}}{\hookleftarrow} B$, send $(\mathtt{SETUP\text{-}OK}, \text{id}) \overset{\tau_1}{\hookrightarrow} \mathcal{F}$ on behalf of A.
(8) Receive $(\text{updateCom}, \text{id}) \overset{\tau_1 \leq \tau_0+2+t_{\text{stp}}}{\hookleftarrow} A$
(9) If B sends $(\mathtt{UPDATE\text{-}OK}, \text{id}) \overset{\tau_1}{\hookrightarrow} \mathcal{F}$, $\mathtt{SignTxs}^{\mathcal{A}}(tx_{set\,i}, addr_{set}, pkey^A_{set} \cup pkey^B_{set})$

(10) If $sig_{set_i}^{B}$ is not returned from $\texttt{SignTxs}^{A}$ in round $\tau_1 + t_s$, execute $\texttt{ForceClose}^{B}(\text{id})$ and go idle.
(11) If A starts the $\mathcal{F}_{\text{Sign}}$ in round $\tau_1 + t_s$ to generate $\sigma^{A}_{\text{Pnsh},i}$, send $(\texttt{REVOKE}, \text{id}) \xhookrightarrow{\tau_1+t_s} \mathcal{F}$ on behalf of A. Participate in the signing on behalf of B.
(12) Upon $(\text{revoke}, \text{id}, \sigma^{A}_{\text{Pnsh},i}) \xhookrightarrow{\tau_1+1+t_s+t_r} A$, continue. Else, execute $\texttt{ForceClose}^{B}(\text{id})$ and go idle.
(13) If B does not send $(\texttt{REVOKE}, \text{id}) \xhookrightarrow{\tau_1+1+t_s+t_r} \mathcal{F}$, go idle.
(14) $\mathcal{S}$ on behalf of B together with A runs the interactive protocol $\mathcal{F}_{\text{Sign}}$ to generate the following signature. $\sigma^{B}_{\text{Pnsh},i}$ on the punishment transaction $tx^{B}_{\text{Pnsh},i}$. Party B receives $\sigma^{B}_{\text{Pnsh},i}$ as output after t_r. In case of failure, execute $\texttt{ForceClose}^{B}(\text{id})$.
(15) $(\text{revoke}, \text{id}, \sigma^{B}_{\text{Pnsh},i}) \xhookrightarrow{\tau_1+1+t_s+2t_r} A$
(16) $\Theta^{B}(\text{id}) := \Theta^{B} \cup \left\{(tx_{set\,i-1}, sig^{B}_{set\,i-1}, \sigma^{B}_{\text{Pnsh},i-1})\right\}$
(17) $\Gamma^{B}(\text{id}) := (tx_F, tx_{set\,i}, sig^{B}_{set_i}, addr_{set}, pkey^{A}_{set}, pkey^{B}_{set})$

Simulator for Close

Case A is honest and B is corrupted

Upon A sending $(\texttt{CLOSE}, \text{id}) \xhookrightarrow{t_0} \mathcal{F}$, do the following.

(1) Extract $(tx_F, tx_{set\,i}, sig^{A}_{set_i}, addr_{set}, pkey^{A}_{set}, pkey^{B}_{set})$ from $\Gamma^{A}(\text{id})$.
(2) Extract $v_{A,i}$ and $v_{B,i}$ from $tx^{A}_{\text{Pay},j} \in tx_{set\,i}$, and c from tx_F
(3) Create transaction $tx_c := tx(\text{Ch}_{AB}, \{pk_A, pk_B\}, \{v_{A,i} + c, v_{B,i} + c\})$, 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 interactive protocol $\mathcal{F}_{\text{Sign}}$ 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 $(\texttt{CLOSE}, \text{id}) \xhookrightarrow{t_0+t_r} \mathcal{F}$ on behalf of B. Else, execute $\texttt{ForceClose}^{A}(\text{id})$.
(6) If tx_c appears on $\mathbb{B}$ in round $t_1 \le t_0 + t_r + \Delta$, set $\Theta^{A}(\text{id}) := \bot$, $\Gamma^{A}(\text{id}) := \bot$.

Simulator for Punish

Case A is honest and B is corrupted

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

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

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

Let τ_0 be the current round

(1) Extract $(tx_F, tx_{set0}, sig^{A}_{set_0}, addr_{set}, pkey^{A}_{set}, pkey^{B}_{set})$ from $\Gamma^{A}(\text{id})$ and extract $tx^{A}_{\text{Pay},j}$ from tx_{set} and $\sigma^{A}_{\text{Pay},j}$ and sig_{set}.
(2) Post $\left(tx^{A}_{\text{Pay},j}, \sigma^{A}_{\text{Pay},j}\right)$ on $\mathbb{B}$
(3) Let $t_2 \le t_1 + \Delta$ be the round in which $tx^{A}_{\text{Pay},j}$ is accepted by $\mathbb{B}$.
(4) If $tx^{A,B}_{\text{Fpay},i}$ appears on $\mathbb{B}$ at or after round $t_3 \le t_2 + \Delta$ and before $\mathbf{T}$, post $\left(tx^{A}_{\text{Pay},j}, \sigma^{A}_{\text{Pay},j}\right)$. Otherwise, post $\left(tx^{A,A}_{\text{Fpay},i}, \sigma^{A,A}_{\text{Fpay},i}\right)$ after $\mathbf{T}$. Set $\Gamma^{P}(\text{id}) := \bot$, $\Theta^{P}(\text{id}) := \bot$.

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 $\text{obsSet}(\text{action}, \tau)$ 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., 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 CREATE in round t_0, A sends message 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 $t_1 \le t_0 + 2 + t_g + t_s + \Delta$, finally A will output CREATED. Thus, the execution ensemble is $EXEC^{\text{create}}_{\Pi,\mathcal{A},\mathcal{E}} := \{\text{createFund}[t_0 + 1], \text{obsSet}(a_0, t_0 + 1), \text{obsSet}(a_1, t_0 + 1 + t_g), \text{createFund}[t_0 + 2 + t_g + t_s], \text{obsSet}(a_2, t_0 + 2 + t_g + t_s), \texttt{CREATED}[t_1]\}$.

[6] The necessity for time-locks can be dropped when using verifiable timed signatures (VTS) as discussed in Section 5.2, 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 $EXEC^{\text{create}}_{\mathcal{F},\mathcal{S},\mathcal{E}} := \{\text{createFund}[t_0+1], \text{obsSet}(a_0, t_0+1), \text{obsSet}(a_1, t_0+1+t_g), \text{createFund}[t_0+2+t_g+t_s], \text{obsSet}(a_2, t_0+2+t_g+t_s), \text{CREATED}[t_1]\}$ □

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 $\left(tx^A_{\text{Pay},j}, \sigma^A_{\text{Pay},j}\right)$ 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 $tx^{A,B}_{\text{Fpay},i}$ appears on $\mathbb{B}$ in round $t_2 \leq t_1+\Delta$ and before $\mathbf{T}$. In this case, A posts $\left(tx^A_{\text{Pay},j}, \sigma^A_{\text{Pay},j}\right)$, 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 $\left(tx^{A,A}_{\text{Fpay},i}, \sigma^{A,A}_{\text{Fpay},i}\right)$ 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 $EXEC^{\text{forceclose}}_{\Pi,\mathcal{A},\mathcal{E}} := \{\text{obsSet}(a_0, t_0), o \in \{\text{obsSet}(a_1, t_2), \text{obsSet}(a_2, \mathbf{T})\}, \text{CLOSED}[t_m]\}$.

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 $tx^{A,B}_{\text{Fpay},i}$ appears on $\mathbb{B}$ in round $t_2 \leq t_1+\Delta$ and before $\mathbf{T}$. In this case, the simulator posts $\left(tx^A_{\text{Pay},j}, \sigma^A_{\text{Pay},j}\right)$, which we denote as action a_1. Otherwise, (ii) the simulator posts $\left(tx^{A,A}_{\text{Fpay},i}, \sigma^{A,A}_{\text{Fpay},i}\right)$ 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 $EXEC^{\text{forceclose}}_{\mathcal{F},\mathcal{S},\mathcal{E}} := \{\text{obsSet}(a_0, t_0), o \in \{\text{obsSet}(a_1, t_2), \text{obsSet}(a_2, \mathbf{T})\}, \text{CLOSED}[t_m]\}$. □

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 $t_1 \leq t_0+2+t_{\text{stp}}$(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 $t_1 \leq t_0+2+t_{\text{stp}}$ (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 $\tau_1+2+t_s+2t_r$

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

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 $\tau_1 + t_s$ (if previous signing was successful) ($\mathcal{S}$)
- REVOKE−REQ to $\mathcal{E}$ in round $\tau_1 + 1 + t_s$ (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 $EXEC^{\text{close}}_{\Pi,\mathcal{A},\mathcal{E}} := \{\text{obsSet}(a_0, t_0), \text{obsSet}(a_1, t_0 + t_r), \text{CLOSED}[t_1]\}$ or $EXEC^{\text{close}}_{\Pi,\mathcal{A},\mathcal{E}} := \{\text{obsSet}(a_0, t_0), \text{obsSet}(a_2, t_2)\}$.

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 $EXEC^{\text{close}}_{\mathcal{F},\mathcal{S},\mathcal{E}} := \{\text{obsSet}(a_0, t_0), \text{obsSet}(a_1, t_0 + t_r), \text{CLOSED}[t_1]\}$ or $EXEC^{\text{close}}_{\mathcal{F},\mathcal{S},\mathcal{E}} := \{\text{obsSet}(a_0, t_0), \text{obsSet}(a_2, t_2)\}$.

□

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.

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 $EXEC^{\text{punish}}_{\Pi,\mathcal{A},\mathcal{E}} := \{\text{obsSet}(a_0, t_0), \text{obsSet}(a_1, t_1), \text{PUNISHED}[t_2]\}$.

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 $EXEC^{\text{punish}}_{\mathcal{F},\mathcal{S},\mathcal{E}} := \{\text{obsSet}(a_0, t_0), \text{obsSet}(a_1, t_1), \text{PUNISHED}[t_2]\}$.

□

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. □

[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

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.