Securing Lightning Channels against Rational Miners

Lukas Aumayr
TU Wien and Common Prefix, lukas.aumayr@gmail.com

Zeta Avarikioti
TU Wien and Common Prefix, georgia.avarikioti@tuwien.ac.at

Matteo Maffei
TU Wien and CDL-BOT, matteo.maffei@tuwien.ac.at

Subhra Mazumdar
Indian Institute of Technology Indore, subhra.mazumdar1993@gmail.com

Abstract

Payment channel networks (e.g., the Lightning Network in Bitcoin) constitute one of the most popular scalability solutions for blockchains. Their safety relies on parties being online to detect fraud attempts on-chain and being able to timely react by publishing certain transactions on-chain. However, a cheating party may bribe miners in order to censor those transactions, resulting in loss of funds for the cheated party: these attacks are known in the literature as timelock bribing attacks. In this work, we present the first channel construction that does not require parties to be online and, at the same time, is resistant to timelock bribing attacks.

We start by proving for the first time that Lightning channels are secure against timelock bribing attacks in the presence of rational channel parties under the assumption that these parties constantly monitor the mempool and never deplete the channel in one direction. The latter underscores the importance of keeping a coin reserve in each channel as implemented in the Lightning Network, albeit for different reasons. We show, however, that the security of the Lightning Network against Byzantine channel parties does not carry over to a setting in which miners are rational and accept timelock bribes.

Next, we introduce `CRAB`, the first Lightning-compatible channel construction that provides security against Byzantine channel parties and rational miners. `CRAB` leverages miners' incentives to safeguard the channel, thereby also forgoing the unrealistic assumption of channel parties constantly monitoring the mempool.

Finally, we show how our construction can be refined to eliminate the major assumption behind payment channels, i.e., the need for online participation. To that end, we present `Sleepy CRAB` the first provably secure channel construction under rational miners that enables participants to go offline indefinitely. We also provide a proof-of-concept implementation of `Sleepy CRAB` and evaluate its cost in Bitcoin, thereby demonstrating its practicality.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Bitcoin, Payment Channels, Lightning Network, Timelocked Bribing, Incentives

Related Version To appear in ACM CCS'24.
Full Version: `ia.cr/2024/826`

1 Introduction

Blockchains inherently suffer from a scalability problem, as nodes must store each transaction on-chain and validate them. The Bitcoin blockchain has exceeded 500GB in space, and its transaction throughput is around ten transactions per second, which is three orders of magnitude lower than that of traditional credit card networks. Blockchains can be classified into two fundamental categories: those with limited scripting capabilities (e.g., Bitcoin with

Workshop on Scalability and Interoperability of Blockchains 2024.
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

more than 50% of the cryptocurrency market share and privacy-oriented cryptocurrencies like Monero and Zcash) and those supporting Turing-complete scripting (Ethereum, Cardano, etc.). The former category features a reduced trusted computing base and is consequently much less prone to hacks and vulnerabilities, while the latter enables the design of more powerful smart contracts.

In this work, we focus on blockchains with limited scripting capabilities. In this context, Payment Channel Networks (PCNs) constitute the most widely deployed scalability solution (e.g., the Lightning Network in Bitcoin has a total value of around 200M USD locked). On a high level, a payment channel (PC) enables an arbitrary number of payments between users while only requiring two on-chain transactions. More precisely, a PC between Alice and Bob is created with a single on-chain transaction, where users lock some of their coins into a shared output controlled by both users (e.g., requiring a 2-of-2 multi-signature). Alice and Bob can pay each other arbitrarily many times by exchanging authenticated off-chain messages representing updates of their balance in the shared output. At any point in time, either of them can close the channel and retrieve their coins by posting the last channel balance on-chain. Should a party try to close the channel with an old balance on-chain, the other party has a certain amount of time to punish such misbehavior, thereby collecting all the channel coins. This punishment mechanism ensures the safety of the channel against Byzantine users, under the assumption that users can timely post punishment transactions on-chain. Finally, a PCN allows a payer to send money to any payee as long as the two are connected by a path of channels with sufficient capacity, updating the channel balances atomically.

1.1 Limitations of PCs

On a high level, current PC protocols for blockchains with limited scripting like Bitcoin suffer from at least one of two severe drawbacks that undermine their widespread deployment. The first one is a *system assumption*: in order to engage in the punishment mechanism, users are assumed to be online, either always [9, 38] or at a certain predefined time [13], which is hardly realistic. Alternatively, users have to rely on third parties, called watchtowers, that act on behalf of offline users; but the watchtowers must either be trusted [28, 34, 30, 16] or lock collateral for each monitored channel, which is financially infeasible [17, 15, 33]. The second one is a *security assumption*: users [9, 13] or watchtowers [28, 34, 30, 17] are assumed to be able to timely post transactions on-chain, which can be defeated in case miners[1] are subject to bribery and are willing to censor transactions if they have a profit in doing so [44, 46]. We summarize this comparison in Table 1, defer the reader to Appendix A for an in-depth comparison to related work. This state of the art leaves open the following research question: *is it possible to design a PC that is compatible with blockchains with limited scripting and does not suffer from either of the previous drawbacks, i.e., it allows users to safely go offline and it is secure against timelock bribing attacks?*

1.2 Our Contributions

We present the first PC construction that is secure against rational miners (AS2) even when a channel party is Byzantine, allows users to safely go offline (AS1), and is compatible with currencies with limited scripting capabilities like Bitcoin. Moreover, our construction is

---

[1] Throughout this work we use the term miners. We note that our protocol is agnostic to the underlying consensus protocol and the term can be replaced with block proposers.

Table 1 Comparison of bi-directional payment channel and watchtower constructions. Additional collateral refers to the total number of extra coins parties need to lock that cannot be utilized for payments. δ is a small, positive value (e.g., 1 or one dust), and v is the total capacity of the channel. All constructions except Sleepy [13], Suborn (DMC) [16], and DMC [23] have an unrestricted lifetime. All constructions except Suborn (DMC) [16] and DMC [23] have an unbounded number of payments. Unrestricted lifetime means the protocol does not require users to close the channel before a pre-specified time. Unbounded payments refer to channel users making any number of payments while the channel is open. In terms of scripts, DS refers to digital signatures, CLTV to absolute timelocks, and CSV to relative timelocks. The last six columns show balance security guarantees in the described settings, assuming different states of the attacker A (can be rational or byzantine), the miners M (can be honest or rational), and the victim V (can be online or offline). $\sim$ means the property holds just under one specific assumption.

	Additional collateral	Permissionless	Script requirements[1]	A: Either M: Honest V: Online	A: Either M: Honest V: Offline	A: Rational M: Rational V: Online	A: Rational M: Rational V: Offline	A: Byzantine M: Rational V: Online	A: Byzantine M: Rational V: Offline
DMC [23]	0	✓	DS + CLTV	✓	✗	$\sim^2$	✗	✗	✗
`LC` [38]	2δ	✓	DS + CSV	✓	✗	$\sim^2$	✗	✗	✗
`LC` + Suborn [16]	0	✓	DS + CSV	✓	✗	✓^{3}	✗	✗	✗
Suborn (DMC) [16]	0	✓	DS + CSV	✓	✗	✓^{3}	✗	✗	✗
`LC` + Monitors [1]/Outpost [28]	2δ	✓	DS + CSV	✓	✓	$\sim^2$	✗	✗	✗
Cerberus [17]	$2v$	✗	DS + CSV	✓	✓^{4}	$\sim^2$	✗	✗	✗
Sleepy [13]	$2v$	✓	DS + (optional) CLTV	✓	✓^{4}	$\sim^2$	✗	✗	✗
Brick [15]	$> 3v$	✗	Turing Complete	✓^{5}	✓^{5}	✓^{5}	✓^{5}	✗	✗
`CRAB`	$2v$ (resp. v)	✓	DS + CSV	✓	✗	✓	✗	✓ (resp. ✗)	✗
`Sleepy CRAB`	$2v$ (resp. v)	✓	DS + CSV	✓	✓	✓	✓	✓ (resp. ✗)	✓ (resp. ✗)

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

[2]: Only secure if parties constantly monitor the mempool. [3]: shows that the property holds within a specific parameter region (including collateral) but breaks otherwise. [4]: Requires honest nodes to come online once in a long time period. [5]: Requires a committee of $3f + 1$ nodes with at most f nodes Byzantine.

permissionless, as it does not depend on pre-defined entities to enforce security. Specifically, the contributions of this work can be summarized as follows:

- We prove for the first time that the Lightning Network is secure against timelock bribing attacks in the presence of rational channel parties, under the assumption that these (i) monitor the mempool and (ii) never deplete the channel in one direction. The former is a fairly unrealistic assumption, which is, however, required to protect from bribing attacks, whereas the latter is already implemented in Lightning, albeit for a different reason, as discussed in Section B.1. In particular, we prove that a small channel balance suffices to make the cheated party engage in a bribing war, which in turn causes the misbehaving party to lose more than it can gain. We formalize the aforementioned bribing war in a game-theoretic model and prove that the honest protocol execution is the Nash Equilibrium for rational parties. We show, however, that the security of the Lightning Network against Byzantine channel parties (i.e., parties willing to lose coins to let the counterparty incur a loss too) does not carry over to a setting in which miners are rational and accept bribes.
- Next, we introduce `CRAB`[2], a PC construction that leverages miners' incentives to safeguard the channel without requiring channel parties to constantly monitor the mempool. `CRAB` is the first channel primitive that is compatible with Lightning and preserves (Byzantine) security even against rational miners. We achieve this with the same collateral as solutions that provide weaker security guarantees, like Cerberus [17] or Sleepy [13]. Unlike previous watchtower-based solutions [13, 17, 33, 15], only channel parties lock collateral in `CRAB`, preserving its permissionless nature. We point out, that the collateral amount required is the minimum required to be secure against rational (or Byzantine) counterparties.

---

[2] `CRAB` is an acronym for Channel Resistant Against Bribery

- Finally, we refine `CRAB` to eliminate a major assumption behind payment channels, i.e., the need for online participation (AS2). Our construction, `Sleepy CRAB`, is the first PC that leverages miners' incentives to enable participants to go offline indefinitely without relying on watchtowers, committees, TEE, or limiting the channel lifetime, while maintaining balance security even when a channel party is Byzantine and miners are rational. Thereby `Sleepy CRAB` improves over all previous solutions, as demonstrated in Table 1.
- We evaluate the performance of `Sleepy CRAB` and our results show that the time and communication costs are in line with the highly efficient Lightning Network.

2 Background and Model

We adopt the notation for UTXO-based blockchains from [10]. Lightning channels (`LC`) consist of three phases. (i) *Open*, two users, Alice and Bob, lock a deposit $v' = v + 2\delta$ consisting of the actual value v and some (small) reserve δ in a multi-sig address, by publishing a funding transaction $\mathtt{tx}_{\langle \mathtt{fund}, C\rangle}$ on-chain. Before publishing it, they create the initial commitment transactions $\mathtt{tx}^{\mathtt{A},0}_{\langle \mathtt{commit}, C\rangle}$ and $\mathtt{tx}^{\mathtt{B},0}_{\langle \mathtt{commit}, C\rangle}$. Alice and Bob can perform payments if they (ii) *update* the channel, by exchanging new commitment transactions $\mathtt{tx}^{\mathtt{A},1}_{\langle \mathtt{commit}, C\rangle}$ and $\mathtt{tx}^{\mathtt{B},1}_{\langle \mathtt{commit}, C\rangle}$ (also known as the state of a channel), where they redistribute the channel balance. Finally, they can (iii) *close* the channel, by posting the latest commitment transaction on-chain. In order to prevent publishing an outdated commitment transaction, there is a punishment mechanism in place that requires the exchanging of secrets r_a^0 and r_b^0 for the previous state. In case someone posts an outdated state, the honest party has some time t to use this secret and use $\mathtt{tx}^{\mathtt{A},0}_{\langle \mathtt{revoke}, C\rangle}$ (or $\mathtt{tx}^{\mathtt{B},0}_{\langle \mathtt{revoke}, C\rangle}$ respectively) and take all the money of the cheating party. Timelock bribing occurs when the cheating party gives money to the miner if they censor this punishment transaction until the t expires and then include the old state instead. For brevity, we defer a more in-depth background section to Appendix B.

2.1 Model and Security Goals

System model and assumptions. We assume the existence of a blockchain $\mathbb{B}$, maintaining the coins currently associated with each address. All miners in $\mathbb{B}$ are considered rational, while each controls less than 50% of the total resources of the system. Miners are responsible for posting transactions in $\mathbb{B}$, thus they select the transactions to be included in a block. A miner selects the most profitable transactions from the mempool to maximize its profit; if it finds a transaction with an "*anyone can spend*" condition, the miner spends the output of that transaction. When miners have the option to achieve equal profit from two different execution branches of a protocol, they always prefer the one that awards them the profit sooner than the branch that offers the same profit later. Considering f the average fee of a blockchain transaction, a briber must thus offer a bribe higher than f to persuade miners to choose its preferred protocol execution branch, e.g., censor a transaction. We incorporate in our model the loss caused by delays in the transaction execution by considering a fixed opportunity cost for miners denoted ϵ.

We denote any channel instance discussed in this paper by C. We consider payment channel primitives consisting of two parties A and B, that may engage with the blockchain miners M to commit fraud. A and B operate their payment channel independently; the miners M do not (and in fact cannot) see or monitor channels or the inter-party communication. They act based on the information shared with them by the users, e.g., by posting transactions. We consider all players to be mutually distrusting rational agents, meaning that the two parties and the miners may deviate from the correct protocol execution if they are to increase their utility. The utility encapsulates the monetary profits of the players. We ignore the loss in opportunity cost for the channel parties.

Threat Model. We define the two different types of participants that we wish to defend against in PCs, rational and Byzantine. A participant's *strategy* refers to the possible actions they can take in a protocol.

▶ **Definition 1** (Rational Party). *A rational party chooses the strategy that maximizes its utility (e.g., monetary profit).*

▶ **Definition 2** (Byzantine Party). *A Byzantine party arbitrarily deviates from the protocol execution, possibly choosing strategies that may decrease its utility.*

Byzantine parties can also be modeled as rational parties with a fixed budget, who increase their utility when another party incurs financial loss (even if they lose funds themselves). For this reason, we often strive to design protocols that remain secure against Byzantine behavior, to capture all possible deviations from the honest protocol execution and, consequently, account for all types of utility functions. We stress, however, that a Byzantine adversary cannot utilize external (to the protocol) funds to increase its budget. As a result, Byzantine parties may only use the channel funds they can access (balance and their collateral) to bribe miners.

Desideratum. Two-party payment channel primitives must, in general, satisfy the following property, stating that no party involved in the channel should lose any coins.

▶ **Definition 3** (Balance Security). *At any time when an honest party* $P \in \{A, B\}$ *holds* α *coins in the latest state of the payment channel, they can claim at least* α *coins on the blockchain.*

3 Analysis for the Bitcoin `LC`

In this section, we model a two-party `LC` interacting with the blockchain miners as an Extensive Form Game (EFG) and demonstrate it is secure under the assumptions that (a) channel parties monitor the mempool and (b) the `LC` channel is never depleted in one direction. The latter assumption highlights the significance of the reserve of `LC`, which is already implemented albeit for protection against nothing-at-stake attacks (i.e., a party with no coins left in the latest update of the channel will always attempt to commit fraud as they have nothing to lose).

3.1 Lightning Channels Model and Analysis

A timelock bribing attack succeeds when the malicious party, say A, publishes an old state of the channel and manages to convince the miners to censor the corresponding revocation transaction. However, the success of such an attack is not straightforward as the cheated party – in this case, B – has also the ability to counter-bribe the miners to include its revocation transaction. This leads to a *bribing war* between the channel parties where rational miners will follow the strategy that awards them the highest payoff, i.e., a miner will publish the revocation transaction only if the bribe of B is higher than the bribe of A.

The core idea of our proof is that each channel primitive can be modeled as an EFG with Perfect Information (Definition 4) [37].

▶ **Definition 4** (Perfect Information Game). *A game in extensive form with perfect information can be formally represented as a tree and defined by the tuple* $(N, H, P, A_i, u_i), i \in N$, *where:*

- N *is a finite set of* n *players,* $N = 1, 2, ..., n$. *Each non-terminal choice node is labeled with the identifier of the player who makes the decision,* $i \in N$.
- H *is the set of histories, where each history* h *represents a sequence of actions that leads to a particular node in the game tree.* $Z \subseteq H$ *is the set of terminal histories representing the ends of all possible play sequences (the leaf nodes in the tree).*
- $P : H \setminus Z \to N$ *is the player function that maps each non-terminal history (or decision node) to the player who is to move at that history.*
- A_i *is a function that associates each player* i *and each history* h *with a set of actions* $A_i(h)$ *available after the history* h *has occurred, assuming player* i *is to move. Edges extending from a node represent the actions,* $A_i(h)$ *for each history* h*, available to the player* i *making the move at that particular point.*
- $u_i : Z \to R$ *is the payoff (or utility) function for each player* i*, which maps each terminal history (or outcome)* $z \in Z$ *to a real number representing player* i*'s payoff in case terminal history* z *is reached.*

We observe that the elements depicted in the EFG provide a comprehensive representation of the game, showing the sequence of decision-making, the set of feasible actions at each stage, and the consequent utilities for each player. Without loss of generality, we assume that the latest state of the `LC` is where A has transferred all the coins to B, but she tries to cheat by posting the initial state $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$. We thus present the punishment mechanism for `LC` in this form with $N = \{A, B, M\}$ illustrated as a game tree $\Gamma_{\mathtt{LC}}$ in Figure 1. The game starts with A, selecting either to post the old state $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$ or the latest state of the channel $\mathtt{tx}^{\mathtt{A},m}_{\langle\mathtt{commit},C\rangle}$. Next, B would punish A by posting $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ or remaining inactive. If B chooses to punish, A would follow up by either offering a bribe $p^a_1 : f < p^a_1 < v$ to the miners, or it would not bribe. If A offers a bribe to the miners, B would either choose to counterbribe with fee $p^b_1 : p^a_1 < p^b_1 < v$ so that miners select $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$, or it may remain inactive and allow A to succeed. If B chooses to counterbribe, A bribes with a fee $p^a_2 > p^b_1$. This bribing war goes on with A bidding p^a_i followed by B bidding p^b_i in the i^{th} round. A finally stops in the n^{th} round when the fee offered becomes $p^a_n = v$ and then B offer a fee $p^b_n = v$. Finally, miner M has to make a decision whether to include $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ or $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$ for mining. The payoffs are mentioned in the leaves of $\Gamma_{\mathtt{LC}}$. If M chooses to mine $A's$ transaction, it will get the fee after $+t$ has elapsed, hence the net payoff deducting the opportunity cost is $v - \epsilon$. On the other hand, if M chooses to mine $B's$ transaction, M gets the fee v instantly. We define a strategy profile in an EFG [37]:

▶ **Definition 5** (Strategy Profile). *A strategy profile in an extensive form game with perfect information specifies for each player* $i \in N$ *what action* $a \in A_i(h)$ *the player will take at every history* h *at which they are called to act. That is, for each player* $i \in N$*, a strategy* s_i *is a function from the set of histories* $H_i = h \in H : P(h) = i$ *to the set of actions* A_i*, such that* $s_i(h) \in A_i(h)$ *for each* $h \in H_i$. *A strategy profile is a list of strategies for all players,* $s = (s_1, s_2, ..., s_n)$.

Correct Protocol Execution as Nash Equilibrium. Equipped with this model, we can outline the desired strategy profile that encapsulates the 'correct protocol execution' (cf. Figure 1): When the channel closes, A chooses the *latest state* strategy. If A posts an old channel state to close the channel, B will choose to *punish* A. Following this, A will *bribe* the miners, and in response, B will offer a *counterbribe* to prevent A from succeeding.

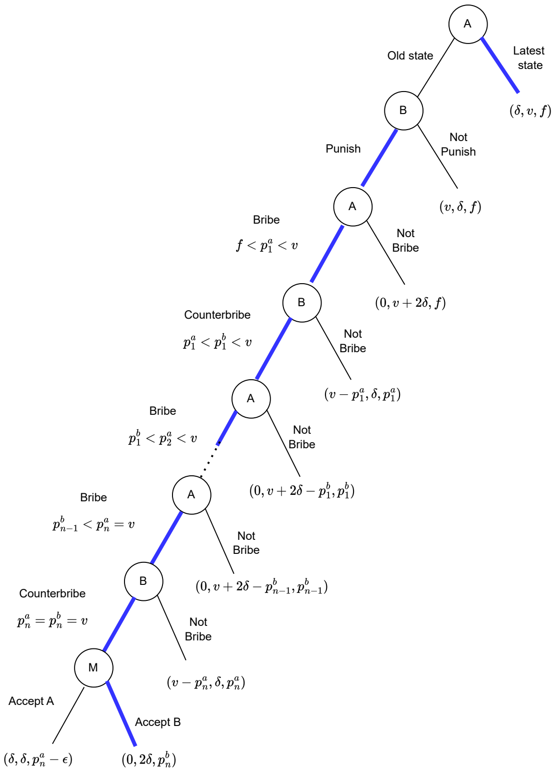

Figure 1 SPNE of Γ_{LC}

This situation leads to a bribing war, ensuring that M receives the maximum payoff, slightly higher than v.

The key point now is to demonstrate that utility-maximizing players will choose these actions at every step of the protocol execution. We do so by proving that the desired strategy profile constitutes a Subgame Perfect Nash Equilibrium (Definition 6) [37] of our game.

▶ **Definition 6** (Subgame Perfect Nash Equilibrium or SPNE). *A strategy profile $s^* = (s_1^*, s_2^*, ..., s_n^*)$ is a Subgame Perfect Nash Equilibrium if and only if, for every subgame G' of the original game G, and every player $i \in N$, the strategy s_i^* is the best response to the strategies of all other players in G'.*

Formally, let H' denote the set of all histories in subgame G'. For each player i, the strategy s_i^* is a best response in G' if:

$$u_i(s_i^*, s_{-i}^*; h) \geq u_i(s_i, s_{-i}^*; h),$$

for all strategies s_i available to player i in G', and for all $h \in H'$. Here, s_{-i}^* denotes the strategies of all players other than i in the SPNE, and $u_i(s_i, s_{-i}^*; h)$ denotes the payoff to player i when all players play according to the strategy profile (s_i, s_{-i}^*) in the subgame beginning at history h.

This condition must hold for all players and all subgames. In other words, a strategy profile is an SPNE if it induces a Nash Equilibrium in every subgame, including the game itself.

To determine the SPNE of a game, we employ a technique called backward induction. Backward induction is a method that starts at the end of a game, at the terminal nodes and moves backward through the extensive form game tree. At each decision node, it is assumed that the player will select the action leading to the highest possible payoff, given their knowledge of future play. This process continues until the beginning of the game is reached, resulting in a prediction of the game's outcome. This prediction, contingent on perfect information and rational behavior, is the SPNE.

▶ **Theorem 7.** *The strategy profile* $s^*(A, B, M)$= *((*latest state*, bribe* $f < p_1^a < v$*, bribe* $p_1^b < p_2^a < v, \ldots,$ *bribe* $p_n^a = v$ *), (*punish*, counterbribe* $p_1^a < p_1^b < v$*, counterbribe* $p_2^a < p_2^b < v, \ldots,$ *counterbribe* $p_n^b = v$*),* Accept B*) is a Subgame Perfect Nash Equilibrium for our game.*

Proof. We use backward induction on $\Gamma_{\texttt{LC}}$. If A *posts an old state*, she should ensure that M mines the transaction. A and B will counter-bribe M so that both A and B end up offering a fee v to M. With both transactions offering the same fee v, M will prefer *accept B* over *accept A* as this gives the payoff without incurring any opportunity cost. B proposes a bribe $p_n^b = v$. This implies that A had bid the same fee. A was provoked by B who had counterbribed an amount less than p_n^a. B was provoked by A and this goes on till A initiated the bribing attack. But before that, B chose to punish A when the latter posted an old state. Tracing the arrow marked in blue in Figure 1, we observe that if A had chosen *old state*, then B would choose to punish, leading to bribing war, so A earns a payoff 0. This is less than the payoff of the latest state, i.e., $\delta > 0$. Thus, A will post the *latest state* and earn δ rather than losing out by bribing M. If A always posts the latest state, B will earn v coins. This proves that (*latest state*, bribe $f < p_1^a < v$, bribe $p_1^b < p_2^a < v, \ldots,$ bribe $p_n^a = v$), (*punish*, counterbribe $p_1^a < p_1^b < v$, counterbribe $p_2^a < p_2^b < v, \ldots,$ counterbribe $p_n^b = v$), *Accept B*) is a subgame perfect Nash Equilibrium. ◀

Theorem 7 provides the desired security property for `LC` under rational participants, as any $P \in \{A, B\}$ closing the channel will always post the latest state. However, if B does not monitor the mempool or back off from the bribing war somewhere in between, A will win the bribing war by offering a bribe higher than the fee offered by B.

▶ **Corollary 8.** *Assuming rational miners and rational parties, balance security is satisfied in* `LC` *if and only if the parties monitor the mempool.*

Nonetheless, leveraging the bribing war to prove the security of `LC` is not ideal, as it relies on the unrealistic assumption that channel parties constantly monitor the mempool. As Bonneau points out in [18], such a strategy would considerably alter the security model of Bitcoin, necessitating all Bitcoin recipients to scan for potential bribery attacks and be prepared to counter them.

Moreover, if a channel party behaves maliciously (Byzantine) and is indifferent to losing their own funds to compromise the security of `LC`, the other party is left vulnerable. For example, if A is Byzantine and indifferent to loss of funds, she will instigate the bribing war as illustrated in Figure 1 and offer a bribe of $v + \delta$ coins. Should B decide to engage in this bribing war, A will force B to lose all v coins. Following the EFG $\Gamma_{\texttt{LC}}$, the miner will then choose to mine the punishment transaction for a fee $v + \delta$. As a result, B will win the bribing war but at the cost of losing its funds.

▶ **Corollary 9.** *Assuming rational miners and Byzantine parties, balance security is* not *satisfied in* `LC`*, despite the honest party monitoring the mempool.*

Modeling miners as single entity. Analyzing LC channels in a model where miners are seen as a single entity is an easy and straightforward way to derive positive results. It assumes that miners are always guaranteed a delayed payoff in the future, which gives them more money. A slightly weaker yet realistic modeling of the miners that considers the distribution of miners allows us to analyze the construction with tighter bounds because now there is a chance that the bribing war is won by the honest party, even if they only counter-bribe with a smaller amount.

Such an analysis is shown in Appendix C, showing that collateral of $c = v/2$ (which would be the channel reserve in LC channels) suffices to safeguard against the setting where there are at least two competing miners with a non-zero chance of mining a block, and no miner has more than 50% of the mining power. Nevertheless, modeling the miners as multiple entities (i) cannot alleviate the assumption that parties must monitor the mempool and (ii) will not help make this construction secure against Byzantine counterparties.

4 CRAB Protocol

In this section, we introduce a new channel construction that is secure against rational parties and miners even when parties are simply running light client verification protocols. We term this new construction CRAB and show that it is secure against Byzantine channel participants.

4.1 CRAB Design

We adapt LC until we arrive at our channel construction, CRAB. Contrarily to LC, an honest party of CRAB does not lose funds when its channel counterparty behaves maliciously and publishes an old state. This is achieved by leveraging the miners' incentives to enforce the correct protocol execution; now miners earn their fee by penalizing the malicious party for publishing an old state.

Incentivizing miners to punish. As a first step towards our solution, we try to incentivize miners by changing the *punishment* transaction $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ (resp. $\mathtt{tx}^{\mathtt{B},0}_{\langle\mathtt{revoke},C\rangle}$) so miners now get all the funds, i.e., $v+\delta$ coins. The rationale here is that A cannot bribe more than v coins from the old state since A gets at least δ in the new state, which is strictly more profitable for A. Miners ignore $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$ and instead include $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$, should A post an old state $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$. This course of action gives the miners $v+\delta$ coins, which is more than what A can offer. However, while this countermeasure ensures that miners post the punishment transaction, it does not ensure balance security for B as all its coins are lost. We thus strive for a channel construction where miners are incentivized to post the punishment transaction, and additionally, the miners' fee is borne by the malicious channel participant.

Collateralizing the channels. To shift the burden of the miners' fees on the cheating party, we require both channel parties to lock c coins as collateral each. The collateral is like the channel reserve δ and it is not part of the usable channel capacity. The usable channel capacity remains v but the total amount of coins needed to open the channel is $2\cdot c+v$. For example, if A provides the channel capacity when opening the channel, then A must lock $v+c$ coins in total while B must lock c coins.

We now modify the commitment transaction to alter the distribution of the channel balance and collateral. In particular, the output of $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$ is split into three parts: (i) B immediately spends the collateral c, (ii) the usable balance v can be either be spent by A after a relative timelock $+t$ or B can immediately spend it using the revocation secret r^0_a shared by A, and (iii) the remaining c coins can either be spend by A after relative timelock

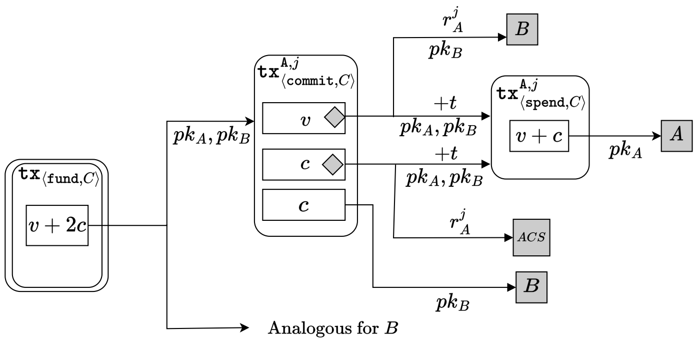

Figure 2 Transaction scheme of `CRAB`. ACS is shorthand for "anyone can spend", which in this case allows anyone, and in particular any miner, who knows r_A^j to claim the c coins.

$+t$, or by any miner (given *"anyone can spend"*) instantly, using secret r_a^0. Note that, in this design, the miners will learn r_a^0 from the revocation transaction posted by B, which contains this secret. There is no need for miners to monitor any communication outside the normal blockchain protocol.

The current design aims to encourage miners to automatically claim A's collateral c in case of fraud while ensuring B will retrieve (at least) its rightful funds. In detail, suppose A posts an old state on-chain and engages in a bribing war. The maximum bribe a rational A will offer for posting old-state $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$ will not exceed v. Thus, for $c > v$, miners will always choose to include the punishment transaction when a party commits fraud.

However, setting $c > v$ leads to using an excessive amount of collateral per channel, which in turn decreases the effective channel capital utilization. In Section C, we deduce the exact bounds of c with respect to v to ensure minimal collateralization of the channel while maintaining security for its participants.

4.2 Protocol Description

This section describes our `CRAB` protocol for realizing bi-directional payment channels. The transaction scheme is represented in Figure 2. We discuss the operations in `CRAB` and provide the pseudocode for each operation in Appendix C in Figure 5.

Opening of channel. A and B open a `CRAB` C by locking coins in a 2-of-2 multi-sig address $\mathtt{addr}_{\mathtt{fund},AB}$. We assume that the usable channel capacity is funded solely by A. Since the intended channel capacity is v, A has to lock $v + c$, and B has to lock just the collateral amount, i.e., c coins. Transaction $\mathtt{tx}_{\langle\mathtt{fund},C\rangle}$ sends $v + 2c$ coins from addresses of A and B to $\mathtt{addr}_{AB}$. Before publishing $\mathtt{tx}_{\langle\mathtt{fund},C\rangle}$, A and B create copies of initial commitment transaction $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$ and $\mathtt{tx}^{\mathtt{B},0}_{\langle\mathtt{commit},C\rangle}$ and exchange signatures on these transactions.

Channel Update. A and B want to update the channel to j^{th} state where A has net balance $v_a + c$ and B has net balance $v_b + c$ such that $v = v_a + v_b$. They generate two copies of the commitment transaction, $\mathtt{tx}^{\mathtt{A},j}_{\langle\mathtt{commit},C\rangle}$ and $\mathtt{tx}^{\mathtt{B},j}_{\langle\mathtt{commit},C\rangle}$, where $\mathtt{tx}^{\mathtt{A},j}_{\langle\mathtt{commit},C\rangle}$ is controlled by A and $\mathtt{tx}^{\mathtt{B},j}_{\langle\mathtt{commit},C\rangle}$ is controlled by B. We explain the transaction scheme with respect to $\mathtt{tx}^{\mathtt{A},j}_{\langle\mathtt{commit},C\rangle}$ having the following outputs:
(i) $v_b + c$ coins can be spent instantly by B.
(ii) $v_a + c$ coins are send to a 2-of-2 multisig address that serves as input of transaction $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$. This can be spent by A after a relative timelock $+t$.

Similarly, for B, the steps for updating the channel with respect to $\mathtt{tx}^{\mathtt{B},j}_{\langle\mathtt{commit},C\rangle}$ are analogous to the above description. Except here B has complete control but has to wait for a relative timelock t before publishing $\mathtt{tx}^{\mathtt{B},j}_{\langle\mathtt{spend},C\rangle}$ and spends $v_b + c$ coins. They invalidate the previous state of the channel by exchanging revocation secrets r_a^{j-1} and r_b^{j-1}.

Closing of channel. CRAB follows the same procedure of channel closure explained for LC in Appendix B.1. However, we describe the changes in the punishment mechanism upon fraudulent channel closure.

If A tries to close the channel by posting old state $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$, B creates revocation transactions $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle} = tx\Big(\mathtt{addr}_{rsmc0,AB}, \mathtt{pk}_{j,B}, 0\Big)$, $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle} = tx\Big(\mathtt{addr}_{rsmc0,AB}, \_, 0\Big)$. $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ allows B to spend v coins immediately provided they have the revocation secret r_a^0. $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ allows any miner with the revocation secret r_a^0 to spend c coins. Thus we put '_' in the place of the output address for $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$. The output of $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$ can also be spent by publishing transaction $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$ after $+t$ has elapsed. However, the relative timelock $+t$ on $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$ ensures that both $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ and $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$, have precedence over the former while spending. A similar procedure is followed by A who posts $\mathtt{tx}^{\phi\mathtt{B},m}_{\langle\mathtt{revoke},C\rangle}$ using secret r_b^0 to punish B for posting old channel state $\mathtt{tx}^{\mathtt{B},0}_{\langle\mathtt{commit},C\rangle}$ on-chain.

Security analysis. We prove the security of this construction in Appendix C, where we show that for a collateral $c = v/2$ this construction is secure against rational miners and rational counterparties and for $c = v$, secure against rational miners and Byzantine counterparties.

5 Extensions, Discussion, Limitations

Sleepy CRAB. Our construction of CRAB up to this point is secure contingent on both parties being online. If B is offline and A posts an old state $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$, B loses balance security since B cannot punish A. We adapt the construction of CRAB for Sleepy CRAB so that balance security is guaranteed even if honest channel participants remain offline. The main challenge is that honest parties are offline and miners need to post revoke transactions by themselves. If, e.g., B wants to go offline after the m^{th} state update, he sends all the revocation secrets $r_a^0, r_a^1, \ldots, r_a^{m-1}$ to the miners. This can be done on the network level, on a public bulletin board (PBB), or on the blockchain. The full protocol, along with some efficiency improvements, can be found in Appendix D.

Evaluation. We evaluate our construction by building a proof-of-concept implementation of LC, published anonymously on GitHub [6]. The cost for punishment is 649 bytes on-chain (around 1.22 USD) and 875 bytes on-chain for unilateral closing (around 1.64 USD), which is in-line with other state-of-the-art solutions. See Appendix E and Table 2 for the full results.

Removing timelocks. One may wonder whether our channel construction could achieve the sought-after goal of (bi-directional) payment channels needing only the signature verification script of the underlying blockchain. Such a channel construction could be adapted for other cryptocurrencies like Monero that do not support any timelock scripts. It turns out that we can indeed remove relative timelocks from CRAB and subsequently from Sleepy CRAB but at the cost of losing balance security in the presence of a Byzantine attacker. We analyze variants without timelocks of CRAB in Appendix G and Sleepy CRAB in Appendix G.2 and prove that, when removing timelocks, our channel constructions are secure only in the rational attacker setting. As our analysis of Appendix C relies on timelocks, we fall back to the analysis used for LC channels in Section 3.1.

Miner-Party Collusion. In our analysis in Appendix C, we assume that there are at least two distinct miners with non-zero mining power and competing interests, i.e., they do not collude with each other. All other miners are allowed to collude freely with each other. This is a very reasonable assumption, as it is the basis of every blockchain consensus. From Appendix C, we can see that having two miners with competing interests is already enough to ensure that every miner's best strategy is to not accept the bribe. Note that every miner can collude with the cheating party; this is already captured in the analysis, where we consider the cheating party to be Byzantine.

Interestingly, even if we relax our assumption and assume an unrealistically strong and irrational adversary, controlling miner(s) with a combined relative mining power of $0.5 < \lambda < 1$, who tries to actively include the bribe even though this is not rational (this is, in fact, equivalent to the counterparty having mining power), we can choose a timelock where the number of remaining blocks k is long enough, such that the non-colluding miner(s) will create a block within that timelock with overwhelming probability. Thus, even in this case, `CRAB` remains secure.

Perfect Information Game. In our game, the cheating party sends to the mempool the bribing transaction. We underscore that if the cheating party (say Alice) does not broadcast the bribe to all miners, any of the miners that win a block within the timelock and do not have the bribe transaction as motivation will simply include the revocation transaction of Bob. Therefore, the best strategy for Alice is to broadcast the transaction to all the miners (as in Bitcoin Alice cannot know the miners that will win the next k blocks.

Underlying Consensus Protocol. As mentioned, our construction is not restricted to Proof-of-Work (PoW) but also applies to other consensus mechanisms, such as Proof-of-Stake (PoS). We do need to differentiate between unpredictable block proposers and predictable ones, e.g., PoS public leader consensus protocols where the block proposers are known in advance. In the latter setting, the cheating party (say Alice) needs to bribe all the $l \leq k$ block proposers to censor Bob's transaction, which will require bribing each of the l block proposers more than c. This results in a total bribe of $l \cdot c$. Since Alice has at most v coins for her bribe, if $v \leq l \cdot c$ holds (which is the case for $c \geq \frac{v}{2}$ assuming there are at least 2 distinct block proposers), the construction is secure.

6 Conclusion

Payment channels like the Lightning Network in Bitcoin, are one of the most promising solutions to the scalability problem of cryptocurrencies. Lightning channels, however, assume that parties constantly monitor the blockchain and can timely post transactions on it. This makes them vulnerable to timelock bribing attacks, where a cheating party may bribe miners to censor valid transactions, resulting in loss of funds for the cheated party.

In this work, we show that Lightning channels are secure against timelock bribing when channel parties are rational and constantly monitor the mempool. However, Lightning channels are insecure when a channel party is Byzantine. We then present `CRAB`, the first PC construction that is secure against rational miners even when adversarial channel parties are Byzantine and is compatible with currencies with limited scripting capabilities like Bitcoin. We then refine `CRAB` to eliminate the major assumption behind payment channels, i.e., the need for online participation, yielding `Sleepy CRAB`. We provide a proof-of-concept implementation of `Sleepy CRAB`, and results demonstrate that our construction, besides being compatible with Lightning, is as efficient as Lightning channels.

As a future work, we intend to generalize our results to Layer-2 protocols building on payment channels, such as multi-hop payments [11, 32, 8], payment channel hubs [27, 41], virtual channels [24, 10, 12, 25], and so on. This requires non-trivial adjustments of the game-theoretic argumentation, possibly leading to additional refinements of such protocols.

Acknowledgements. This work was supported by the European Research Council (ERC) under the Horizon 2020 research (grant 771527-BROWSEC); by the Austrian Science Fund (FWF) through the SFB SpyCode project F8510-N and F8512-N, the project CoRaF (grant agreement ESP 68-N), the Bridge-1 project PR4DLT (grant 13808694), and the COMET K1 SBA and COMET K1 ABC; by CoBloX; by the Austrian 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 the WWTF through the project 10.47379/ICT22045.

References

1 Unlinkable outsourced channel monitoring, 2016. `https://diyhpl.us/wiki/transcripts/scalingbitcoin/milan/unlinkable-outsourced-channel-monitoring/`.

2 Txwithhold smart contracts by gleb naumenko, 2022. URL: `https://blog.bitmex.com/txwithhold-smart-contracts/`.

3 Bitcoin price in usd, October 2023. `https://coinmarketcap.com/`.

4 Bitcoin transaction fees per byte, October 2023. `https://privacypros.io/tools/bitcoin-fee-estimator/`.

5 Bolt #2: Peer protocol for channel management, 2023. URL: `https://github.com/lightning/bolts/blob/master/$02-peer-protocol.md$#rationale`.

6 Github repository of our evaluation proof-of-concept, October 2023. `https://github.com/crab-channels/evaluation`.

7 Lightning liquidity, 2024. URL: `https://bitcoin.design/guide/how-it-works/liquidity/#channel-reserve`.

8 Lukas Aumayr, Kasra Abbaszadeh, and Matteo Maffei. Thora: Atomic and privacy-preserving multi-channel updates. CCS '22, page 165–178, New York, NY, USA, 2022. Association for Computing Machinery. `doi:10.1145/3548606.3560556`.

9 Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized channels from limited blockchain scripts and adaptor signatures. In *Advances in Cryptology – ASIACRYPT 2021*, 2021.

10 Lukas Aumayr, Matteo Maffei, Oğuzhan Ersoy, Andreas Erwig, Sebastian Faust, Siavash Riahi, Kristina Hostáková, and Pedro Moreno-Sanchez. Bitcoin-compatible virtual channels. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 901–918. IEEE, 2021.

11 Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Blitz: Secure Multi-Hop Payments Without Two-Phase Commits. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

12 Lukas Aumayr, Pedro Moreno Sanchez, Aniket Kate, and Matteo Maffei. Breaking and Fixing Virtual Channels: Domino Attack and Donner. In *30th Annual Network and Distributed System Security Symposium, NDSS*, 2023.

13 Lukas Aumayr, Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, Pedro Moreno-Sanchez, and Matteo Maffei. Sleepy Channels: Bi-Directional Payment Channels without Watchtowers. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 179–192, New York, NY, USA, 2022. Association for Computing Machinery. `doi:10.1145/3548606.3559370`.

14 Georgia Avarikioti, Felix Laufenberg, Jakub Sliwinski, Yuyi Wang, and Roger Wattenhofer. Towards secure and efficient payment channels, 2018. `arXiv:1811.12740`.

15 Zeta Avarikioti, Eleftherios Kokoris-Kogias, Roger Wattenhofer, and Dionysis Zindros. Brick: Asynchronous incentive-compatible payment channels. In *Financial Cryptography and Data*

Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II, page 209–230, Berlin, Heidelberg, 2021. Springer-Verlag. `doi:10.1007/978-3-662-64331-0_11`.

16 Zeta Avarikioti and Orfeas Stefanos Thyfronitis Litos. Suborn channels: Incentives against timelock bribes. In *International Conference on Financial Cryptography and Data Security*, pages 488–511. Springer, 2022.

17 Zeta Avarikioti, Orfeas Stefanos Thyfronitis Litos, and Roger Wattenhofer. Cerberus channels: Incentivizing watchtowers for bitcoin. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security*, pages 346–366, Cham, 2020. Springer International Publishing.

18 Joseph Bonneau. Why buy when you can rent? In Jeremy Clark, Sarah Meiklejohn, Peter Y.A. Ryan, Dan Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, pages 19–26, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

19 Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 142–157. IEEE, 2019.

20 Hao Chung, Elisaweta Masserova, Elaine Shi, and Sri AravindaKrishnan Thyagarajan. Rapidash: Foundations of side-contract-resilient fair exchange. *Cryptology ePrint Archive*, 2022.

21 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 910–927. IEEE, 2020. `doi:10.1109/SP40000.2020.00040`.

22 Christian Decker and Rusty Russell. eltoo: A simple layer2 protocol for bitcoin, 2018. `https://blockstream.com/eltoo.pdf`.

23 Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In Andrzej Pelc and Alexander A. Schwarzmann, editors, *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings*, volume 9212 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015. `doi:10.1007/978-3-319-21741-3\_1`.

24 Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 106–123. IEEE, 2019.

25 Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 949–966, New York, NY, USA, 2018. Association for Computing Machinery. `doi:10.1145/3243734.3243856`.

26 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. *J. ACM*, apr 2024. Just Accepted. `doi:10.1145/3653445`.

27 Noemi Glaeser, Matteo Maffei, Giulio Malavolta, Pedro Moreno-Sanchez, Erkan Tairi, and Sri Aravinda Krishnan Thyagarajan. Foundations of coin mixing services. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 1259–1273, New York, NY, USA, 2022. Association for Computing Machinery. `doi:10.1145/3548606.3560637`.

28 Majid Khabbazian, Tejaswi Nadahalli, and Roger Wattenhofer. Outpost: A responsive lightweight watchtower. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, AFT '19, page 31–40, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3318041.3355464`.

29 Aggelos Kiayias and Orfeas Stefanos Thyfronitis Litos. A composable security treatment of the lightning network. In *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*, pages 334–349, 2020. `doi:10.1109/CSF49147.2020.00031`.

30 Marc Leinweber, Matthias Grundmann, Leonard Schönborn, and Hannes Hartenstein. Tee-based distributed watchtowers for fraud protection in the lightning network. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology: ESORICS 2019 International Workshops, DPM 2019 and CBT 2019, Luxembourg, September 26–27, 2019, Proceedings*, page 177–194, Berlin, Heidelberg, 2019. Springer-Verlag. `doi:10.1007/978-3-030-31500-9_11`.

31 Joshua Lind, Ittay Eyal, Peter R. Pietzuch, and Emin Gün Sirer. Teechan: Payment channels using trusted execution environments. *CoRR*, abs/1612.07766, 2016. URL: `http://arxiv.org/abs/1612.07766`, `arXiv:1612.07766`.

32 Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *Network and Distributed System Security Symposium, NDSS*, 2019.

33 Patrick McCorry, Surya Bakshi, Iddo Bentov, Sarah Meiklejohn, and Andrew Miller. Pisa: Arbitration outsourcing for state channels. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, AFT '19, page 16–30, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3318041.3355461`.

34 Arash Mirzaei, Amin Sakzad, Jiangshan Yu, and Ron Steinfeld. Fppw: A fair and privacy preserving watchtower for bitcoin. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021*, page 151–169, Berlin, Heidelberg, 2021. Springer-Verlag. `doi:10.1007/978-3-662-64331-0_8`.

35 Pedro Moreno-Sanchez, Arthur Blue, Duc V. Le, Sarang Noether, Brandon Goodell, and Aniket Kate. Dlsag: Non-interactive refund transactions for interoperable payment channels in monero. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security*, pages 325–345, Cham, 2020. Springer International Publishing.

36 Tejaswi Nadahalli, Majid Khabbazian, and Roger Wattenhofer. Timelocked bribing. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I 25*, pages 53–72. Springer, 2021.

37 Martin J Osborne and Ariel Rubinstein. *A course in game theory*. MIT press, 1994.

38 Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

39 Rusty Russell. [lightning-dev] splicing proposal: Feedback please!, 2018. `https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-October/001434.html`.

40 Jeremy Spillman. Spillman-style payment channels, 2013. `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html`.

41 E. Tairi, P. Moreno-Sanchez, and M. Maffei. A2l: Anonymous atomic locks for scalability in payment channel hubs. In *2021 2021 IEEE Symposium on Security and Privacy (SP)*, pages 1919–1936, Los Alamitos, CA, USA, may 2021. IEEE Computer Society. URL: `https://doi.ieeecomputersociety.org/10.1109/SP40001.2021.00111`, `doi:10.1109/SP40001.2021.00111`.

42 Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, Fritz Schmid, and Dominique Schröder. Verifiable timed linkable ring signatures for scalable payments for monero. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian D. Jensen, and Weizhi Meng, editors, *Computer Security – ESORICS 2022*, pages 467–486, Cham, 2022. Springer Nature Switzerland.

43 Peter Todd. Cltv-style payment channels, 2018. `https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki#Payment_Channels`.

44 Itay Tsabary, Matan Yechieli, Alex Manuskin, and Ittay Eyal. MAD-HTLC: because HTLC is crazy-cheap to attack. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1230–1248. IEEE, 2021.

45 Jo Van Bulck, David Oswald, Eduard Marin, Abdulla Aldoseri, Flavio D Garcia, and Frank Piessens. 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*, pages 1741–1758, 2019.

46 Sarisht Wadhwa, Jannis Stoeter, Fan Zhang, and Kartik Nayak. He-HTLC: Revisiting Incentives in HTLC. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023. URL: `https://www.ndss-symposium.org/ndss-paper/he-htlc-revisiting-incentives-in-htlc/`.

47 Guiyi Wei, Xiaohang Mao, Rongxing Lu, Jun Shao, Yunguo Guan, and Genhua Lu. Achieve space-efficient key management in lightning network. *Computer Networks*, 197:108346, 2021. URL: `https://www.sciencedirect.com/science/article/pii/S1389128621003406`, `doi:10.1016/j.comnet.2021.108346`.

A Related Work

Timelock bribing. Timelock bribing attacks, originally introduced for Hashed Time Lock Contracts (HTLCs) [36], leverage the vulnerabilities of timelocked contracts to censoring attacks. The core idea of timelock bribing attacks is that blockchain miners can be bribed to include a transaction on-chain, which is only valid in the future after a timelock expires, and meanwhile censor a conflicting but currently valid transaction. Applied to HTLCs, this attack may result in loss of funds, violating their security under the assumption of rational miners. Tracing such attacks is challenging as excluded transactions are not reported on-chain, and to date, the Bitcoin community has not reported any instances of such attacks on time-sensitive protocols. Nevertheless, BitMEX Research [2] has highlighted some practical approaches for implementing TxWithhold Smart Contracts. The objective of these contracts is to bribe miners to omit certain transactions from their blocks. These works identify that timelock bribing is a potential risk, especially for HTLCs which are commonly used, e.g., for routing payments in the Lightning Network [38].

To safeguard HTLCs against timelock bribing, Tsabary et al. proposed MAD-HTLC [44], which enables miners to extract the value locked should cheating occur. This is known as Maximal (or sometimes Miner) Extractable Value (MEV) [21]. While such optimizations are common in the Ethereum network, Bitcoin's default cryptocurrency client only offers basic optimization. The authors introduced a patch to the standard Bitcoin client to create Bitcoin-MEV infrastructure in order to implement MAD-HTLC. Soon after, a reverse-bribing attack on MAD-HTLC was discovered and mitigated by He-HTLC [46], based on the idea of burning part of the deposit of the dishonest participant. Concurrently, Rapidash [20] proposed a similar solution mainly focusing on atomic swaps. Nevertheless, none of these works discussed or addressed bribing attacks in payment channels. In contrast to HTLCs, where the burning of funds disincentivizes misbehavior, payment channels such as `LC` channels [38] can detect the cheating party, and thus have the theoretic potential to compensate honest parties, and therefore safeguard against Byzantine parties, as we elaborate below.

Payment Channels. The fundamental idea of payment channels (PCs) is that the transaction workload is lifted off-chain while the blockchain is used only in case of disputes. Hence, the on-chain settlement process of PCs is critical for their security. This process typically depends on one main premise: if a cheating party posts an old transaction, the cheated party can post some data (e.g., revocation transaction) on-chain within a pre-defined time period (timelock) in order to ensure it will not lose its PC funds. This premise, in turn, depends on two key assumptions: the channel parties (AS1) constantly monitor the blockchain to detect potential fraud with respect to their channel, i.e., cannot go offline for an arbitrarily long period, and (AS2) can timely post a transaction on the blockchain, i.e., they are not censored by the miners even if the miners are rational.

In the following, we review the main PC constructions and potential add-on solutions presented in the literature, pinpointing their exact assumptions and guarantees. We mainly focus on two assumptions, namely (AS1) online parties and (AS2) non-censoring miners, as mentioned above, as well as if the solutions are applicable to Bitcoin, which is the blockchain with the largest market cap and hosting the largest PCN, the Lightning Network. To do so, we evaluate if security holds under different system models considering the possible behavior of miners (honest/rational), attackers (rational/Byzantine), and victims (online/offline). A comprehensive comparison of the different solutions discussed here is illustrated in Table 1.

Unidirectional PCs: The first payment channel proposals (e.g., CLTV [43] and Spilman [40]) were unidirectional, meaning that one party is always the payer and the other party is always the payee. In this setting, only the payee can close the channel and there is no need to protect from attempts to finalize on-chain old channel states (i.e. balance distributions) since the payee always prefers the most recent state. In case the payee does not close the channel within a fixed time set upon the channel creation, the payer will be refunded. Other instances of unidirectional channels, such as Paymo [42] and DLSAG [35], support off-chain payments in Monero. Unidirectional PCs are, in general, safe against censoring from rational miners (AS2), and the parties can be offline for the lifetime of the channel (AS1). Since their lifetime is limited, these channels have to be closed and opened again after a predefined amount of time, which involves on-chain transactions. Furthermore, unidirectional channels are not capital-efficient as the locked coins can only flow in one direction; as such they were quickly replaced by bi-directional PCs.

Bi-directional PC: Duplex micropayment channels (DMC) [23] supported for the first time bi-directional payments, in which at any time, each party can play the role of payer as well as payee, at the cost of a bounded number of payments, after which the channel can no longer be used and has to be closed. Eltoo [22] also supports bi-directional payments but it is not compatible with Bitcoin due to special scripting requirements. Lightning channels [38], which are deployed in Bitcoin, are the de-facto standard today since they enable bidirectional payments as well as an unlimited channel lifetime. These bidirectional payment channels have effective punishment mechanisms to protect from attempts to finalize old channel states on-chain. In particular, if the malicious party posts an old channel state, the honest party can raise a dispute within a given time window, punishing the fraud attempt by collecting all the channel balances. All these constructions guarantee security only if channel parties are online (AS1) and miners are honest and do not censor transactions (AS2).

Rational miners: The only work that investigated the security of PCs under rational miners (addressing AS2), and considered timelock bribing attacks within the context of payment channels is [16]. There, Avarikioti et al. proposed a modification of DMC channels, introducing a new channel primitive termed *Suborn*, that enabled miners to claim the coins of the briber upon the honest party posting the punishment transaction. Suborn channels, although secure against timelock bribing attacks, still suffer from the DMC drawbacks: only a limited number of transactions is feasible coupled with a bounded channel lifetime. Additionally in [16], the parameter region in which bribes are effective in Lightning channels was examined, and the authors proposed the use of an increased fee in the revocation transaction, depending on the value of each transaction, to increase the secure region. However, this work only limits the parameter region in Lightning in which timelock bribes are effective. Beyond this region, the channel design is not secure against bribing attacks. Most importantly, both proposals in [16] are insecure when parties are offline.

Offline parties: There are several works addressing the requirement for online participation in PCs (AS1). The most common approach entails utilizing third parties, the so-called watchtowers, to punish malicious channel parties on behalf of the offline counterparty. This approach was originally introduced with Monitors [1], some special nodes in the Bitcoin network that were deemed responsible for monitoring the mempool and punishing fraud attempts. Monitors, however, are not properly incentivized to provide this service in the first place because they do not get paid unless fraud happens. DCWC [14] is another watchtower proposal suffering from the same weaknesses. Later, Outpost [28], Pisa [33], and Cerberus [17], solved this problem by granting a fee to watchtowers for each channel update. Although all these proposals alleviate (but do not eliminate) the demand for online participation, they assume watchtowers can timely post transactions on-chain and do not consider rational miners that may be bribed to censor such transactions. Therefore, they still suffer from timelock bribing attacks and remain secure only when miners are honest (AS2).

A similar approach to watchtowers, relying instead on a trusted execution environment (TEE), was proposed in Teechan [31]. Teechan guarantees security when honest parties go offline but it assumes that transactions can be timely posted on-chain (AS2), similar to watchtowers. Moreover, the security of TEEs is, in general, questionable given the number of discovered vulnerabilities [19, 45], besides constituting a strong system assumption.

Taking a different approach to tackle the online participation assumption without the use of watchtowers or TEEs, Aumayr et al. recently proposed a new Bitcoin-compatible PC, called Sleepy Channels [13]. Sleepy channels are yet another proposal that is insecure against timelock bribing attacks, as their security depends on parties timely posting transactions on-chain in case of fraud. Additionally, Sleepy channels require a limited channel lifetime.

The only PC proposal that has successfully addressed both the online participation (AS1) and remains secure under rational miners that may engage in censoring (AS2) is Brick [15]. Brick employs a pre-selected committee of watchtowers within the channel itself and restricts the settlement process of the channel to either occur in collaboration with the counterparty or the committee, thereby achieving security in asynchrony without the use of timelocks. Nevertheless, Brick suffers from several limitations: (i) it loses security when a channel party is Byzantine, meaning they are willing to lose coins in order to inflict loss to its counterparty, (ii) it needs a Turing complete scripting language that makes it incompatible with blockchains like Bitcoin, (iii) it requires a prohibitively high collateral (at least three times the channel balance), (iv) it is not permissionless since it relies on a predefined committee that is registered during the channel opening and collateralizes the channel for security.

B Preliminaries

UTXO model. We adopt the notation for UTXO-based blockchains from [10]. Coins are held in outputs of transactions in the UTXO model. The output θ is a tuple $(\theta.\texttt{cash}, \theta.\psi)$, where $\theta.\texttt{cash}$ denotes the amount of coins associated with the output and $\theta.\psi$ denotes the conditions that need to be satisfied to spend the output. In general, $\theta.\psi$ contains the scripts with specific operations supported by the underlying blockchain. In this paper, we focus on Bitcoin, which, among others, allows for signature verification (single and multi-sig), absolute and relative timelocks, hashlocks, and logical $\wedge$ and $\vee$. A user P controls or owns an output θ if $\theta.\psi$ contains only a signature verification with respect to the public key of P.

A transaction in the UTXO model maps one or more existing unspent outputs to a list of new outputs. A transaction `tx` consists of the following attributes (`tx.txid`, `tx.Input`, `tx.Output`, `tx.TimeLock`, `tx.Witness`). $\texttt{tx.txid} \in \{0,1\}^*$, called the identifier of the transaction, is calculated as $\texttt{tx.txid} := \mathcal{H}([\texttt{tx}])$, where $\mathcal{H}$ is a hash function that is

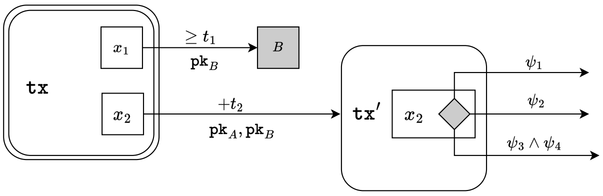

Figure 3 Illustration of our transaction chart notation: Transaction `tx` is on-chain (double-bordered) and has two outputs (boxes), whose spending conditions are specified by arrows: the first output has value x_1 that can be spent by party B with a transaction signed with $\mathtt{pk}_B$ at or after round t_1 (absolute timelock); the other one has value x_2 that can be spent by a transaction signed by $\mathtt{pk}_A$ and $\mathtt{pk}_B$ (multisig) but only if at least t_2 rounds passed since `tx` was posted on the blockchain (relative timelock). Transaction $\mathtt{tx}'$ is off-chain (single-bordered), has one input, which is the second output of `tx` containing x_2 coins, and has only one output, which is of value x_2 and can be spent by a transaction whose witness satisfies the output condition $\psi_1 \vee \psi_2 \vee (\psi_3 \wedge \psi_4)$. $\psi_1 := r$ would denote a hashlock, which can be satisfied if a witness x is given, such that $x = \mathcal{H}(r)$. The input of `tx` is not shown.

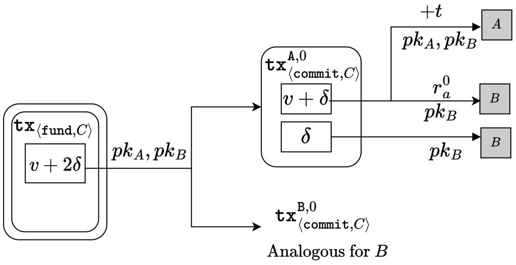

Figure 4 Transaction scheme of an instance of `LC` between A and B. It shows the state of lightning channel C when the initial state of the channel has been updated.

modeled as a random oracle. $[\mathtt{tx}]$ is the body of the transaction defined as $[\mathtt{tx}] := (\mathtt{tx.Input}, \mathtt{tx.Output}, \mathtt{tx.TimeLock})$. `tx.Input` is a vector of strings $[addr_1, addr_2, \ldots, addr_n]$ which identify the inputs of `tx`, where each $addr_i, i \in [1, n]$ are the source addresses. Similarly, `tx.Output` is the output of `tx`, comprising vector of new output addresses $[addr'_1, addr'_2, \ldots, addr'_m]$. $\mathtt{tx.TimeLock} \in \mathbb{N} \cup \{0\}$ denotes the absolute (or relative) timelock of the transaction. It denotes that `tx` will not be accepted by the blockchain before the round defined by `tx.TimeLock`. If the timelock is 0, then `tx` can be spent immediately. Lastly, $\mathtt{tx.Witness} \in \{0, 1\}^*$, called the transaction's witness, contains the witness of the transaction that is required to spend the transaction inputs. For readability, we use a transaction chart notation, which we illustrate and explain in Figure 3.

B.1 Lightning Channels

Architecture. Operating a lightning channel (LC) consists of the following phases: open, update, and close. Throughout the paper, we refer to an instance of `LC` as C.

(a) Channel Open: Suppose Alice (A) and Bob (B) decide to establish a Lightning channel with an initial deposit of $v' = v + 2\delta$, contributed by A, where v is the transferable value and δ the (small) channel reserve. To do so, they agree on a funding transaction $\mathtt{tx}_{\langle \mathtt{fund}, C \rangle}$, that spends two outputs, one controlled by A and one by B, holding a total of v' coins. $\mathtt{tx}_{\langle \mathtt{fund}, C \rangle}$ then transfers these coins to a new output requiring both signatures of A

and B, known as a multi-sig address. Note that typically in LC, one party – in this case, Alice – provides the entire funding amount v'.

Before publishing the funding transaction on-chain, both parties create, sign, and exchange their own copy of the initial commitment transaction, $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$ for A, and $\mathtt{tx}^{\mathtt{B},0}_{\langle\mathtt{commit},C\rangle}$ for B. These transactions spend the output of $\mathtt{tx}_{\langle\mathtt{fund},C\rangle}$ and distribute the funds of the channel to their initial contributors (here, A gets back v' coins) after a relative timelock $+t$ expires. This timelock is to prevent cheating by allowing the revocation of old states; more on this below. Exchanging the initial commitments before opening the channel on-chain is critical for security as it ensures that parties cannot hold their counterparty hostage in the channel, upon its creation.

Once $\mathtt{tx}_{\langle\mathtt{fund},C\rangle}$ is added to the blockchain, the payment channel between A and B is effectively *open*. We illustrate the transaction flow of C in Figure 4, where parties A and B lock up some coins in C via the funding transaction $\mathtt{tx}_{\langle\mathtt{fund},C\rangle}$.

(b) Channel Update: If A and B wish to make an off-chain payment, they need to update the channel state, i.e., the distribution of the v' coins among A and B. To do so, the two parties sign and exchange new commitment transactions, $\mathtt{tx}^{\mathtt{A},1}_{\langle\mathtt{commit},C\rangle}$ and $\mathtt{tx}^{\mathtt{B},1}_{\langle\mathtt{commit},C\rangle}$, and the revocation secrets for the previous commitment transaction r^0_a (of A) and r^0_b (of B). The new commitment transactions validate that both parties agreed on the new channel state and depict the new coin distribution after the payment; they only differ in that they enforce a relative timelock $+t$ on the output of the party that holds it, e.g., $\mathtt{tx}^{\mathtt{A},1}_{\langle\mathtt{commit},C\rangle}$ enforces a timelock on A' output. The revocation secrets ensure that the previous commitment transaction can get invalidated if it appears on-chain, and the corresponding party is penalized.

During the update phase where payments are executed off-chain within a channel C, it is recommended that each party maintains a reserve δ ideally equal to 1% of the total channel capacity. This reserve is a specified amount of coins that each participant should retain in their channel balance and not use for transactions. The intention behind introducing the channel reserve is to make it less beneficial for a cheating party to close the channel at an old state [7]. Now, out of the total channel capacity $v' = v + 2\delta$, only v is usable, with A and B each maintaining a channel reserve of δ [5]. If one party does not have the channel reserve initially (but instead, e.g., 0 coins), the reserve is ensured as soon as that party receives money.

(c) Channel Close: A payment channel can be closed either (i) *co-operatively* or (ii) *unilaterally.*

(i) A and B may mutually agree to *co-operatively close* the channel. In this case, they sign and post on-chain a transaction that spends the output of the funding transaction $\mathtt{tx}_{\langle\mathtt{fund},C\rangle}$ and distributes to each party its coins as agreed in the latest update of the channel.

(ii) If one of the parties is not responsive, say B, the counterpart A may close the channel *unilaterally* without the cooperation of B. To do so, A publishes on-chain the last commitment transaction. B recovers its funds immediately while A can spend her funds only after the relative timelock t expires. For the rest of this work, we denote by $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$ and $\mathtt{tx}^{\mathtt{B},0}_{\langle\mathtt{spend},C\rangle}$ the transactions spending the outputs of $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$ and $\mathtt{tx}^{\mathtt{B},0}_{\langle\mathtt{commit},C\rangle}$ respectively.

In case a party posts an old commitment transaction in an attempt to close the channel in a more beneficial state for themselves, the revocation secrets come into play. Specifically, if A posts the old state $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$ on-chain to close the channel, she can access her funds only after the relative timelock $+t$, B can spend them knowing r^0_a. Thus, B employs the secret r^0_a to create a revocation transaction $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$. The revocation transaction invalidates the previous commitment transaction, and grants control over all the channel funds to the party who submits the revocation on-chain. Note that the validity of the revocation transaction is contingent on a party publishing on-chain the corresponding old commitment, as it spends the timelocked output of the old commitment. For example, B can utilize $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{revoke},C\rangle}$ with secret r_a^0 to access the funds from $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{commit},C\rangle}$ within time t of its publication *only if* A has posted $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{commit},C\rangle}$ on-chain. Therefore, to ensure the safety of payment channels, it is critical for parties involved to vigilantly monitor the blockchain in order to detect and revoke potential fraud attempts.

Implementing the revocation. There are multiple ways of implementing revocation. In [29], *combined signatures* are used, a two-party scheme that allows the *signer* to construct the signing key only if the *secret holder* shares secret information. This protocol enables the efficient exchange of revocation secrets. However, as pointed out in other work, e.g. [9, 13], this revocation functionality can be implemented also by simply hashing a secret, adaptor signatures, or using a 2-of-2 multi-signature. For example, the spending condition for A's coins in $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{commit},C\rangle}$ (or B's coins in $\texttt{tx}^{\texttt{B},0}_{\langle\texttt{commit},C\rangle}$) can be the hashlock $\mathcal{H}(r_a^0)$ (or $\mathcal{H}(r_b^0)$).

Timelock bribing attack in Lightning Channels. We revisit here the timelock bribing attack, specifically in the context of Lightning Channels, which was initially examined in [16]. After updating the channel state, A can maliciously post $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{commit},C\rangle}$ where she holds the full channel capacity. Thereby, B has to post the corresponding revocation transaction using the secret r_a^0, before t expires. Given that the blockchain miners are assumed to be honest and do not censor transactions, even when bribed by A, the punishment mechanism of `LC` is secure in this setting. However, miners are, in principle, rational agents and thus choose to mine the transaction with a higher fee. Hence, the miners may censor an honest party's revocation transaction and allow the malicious party to publish its old commitment transaction if the latter comes with a higher fee. Specifically in our example, suppose A publishes $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{commit},C\rangle}$ and B publishes $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{revoke},C\rangle}$ with fee f_b. Now A publishes $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{spend},C\rangle}$ with fee $f_a : f_a > f_b$. Miners may now censor B's transaction until t expires to get the larger fee f_a instead of f_b. Thus, the revocation mechanism of `LC` is susceptible to timelock bribing attacks.

C `CRAB` Analysis and Pseudocode

We present the full protocol pseudocode in Figure 5. In our analysis of `CRAB`, it is essential to revisit its core goals. Designed to eliminate the necessity for parties to constantly watch the mempool and engage in active counterbribing, `CRAB` integrates a pre-determined collateral, c. This collateral serves both as a penalty for cheating and an implicit counterbribe to miners. We stress that such collateral is unavoidable, as it is necessary to counter-effect the bribe of the cheating party to the miners. The key challenge here is setting the collateral amount in advance while keeping it minimal to ensure the construction's efficacy.

It is possible to analyze `CRAB` channels in the same way as `LC` channels in Section 3. However, the analysis yields imperfect results: (i) a demand for higher collateral of $c \geq v$ where v is the total capacity of the channel, and (ii) no security against Byzantine counterparties.

Therefore, we defer this analysis to Appendix F.1 and instead opt for a more in-depth analysis here, which, in addition to the collateral, takes timelocks into account and considers multiple (> 1) distinct miners where at least one is not colluding, instead of the miners as a single entity (cf. Section 5). This assumption is the basis of every blockchain consensus and something that holds in practice [16, 20, 46].

Our findings suggest that even with rational and miners, a collateral of $c \geq \frac{v}{2}$ can secure against rational counterparties and $c \geq v$ against Byzantine counterparties. Note that this

Parties A and B each have funding address (also public keys) $\mathtt{pk}_{\mathtt{fund},A}$ and $\mathtt{pk}_{\mathtt{fund},B}$ respectively. The corresponding secret keys of these addresses[a] are $\mathtt{sk}_{\mathtt{fund},A}$ and $\mathtt{sk}_{\mathtt{fund},B}$. Both A and B have sufficient balance in the funding address to fund a `CRAB` C of capacity $v + 2c$ where $v + c$ are locked by A and c coins are locked by B. The transactions can be broadcasted on the ledger $\mathbb{B}$ parameterized by $(\Delta, \Sigma, \mathcal{V})$. Δ is the time after which a valid transaction is appended to the ledger, a signature scheme Σ, and a set $\mathcal{V}$, defining valid spending conditions, including signature verification under Σ, supporting absolute and relative timelocks.

Opening of channel

(1) Parties use $\mathtt{KGen}(1^\lambda)$ for generating the following keys: A generates $(\mathtt{pk}_{\mathtt{comm0},A}, \mathtt{sk}_{\mathtt{comm0},A}), (\mathtt{pk}_{\mathtt{rsmc0},A}, \mathtt{sk}_{\mathtt{rsmc0},A})$ and B generates $(\mathtt{pk}_{\mathtt{comm0},B}, \mathtt{sk}_{\mathtt{comm0},B}), (\mathtt{pk}_{\mathtt{rsmc0},B}, \mathtt{sk}_{\mathtt{rsmc0},B})$. A and B jointly generate 2-of-2 multi-sig addresses $\mathtt{addr}_{\mathtt{fund},AB}$, $\mathtt{addr}_{rsmc0,AB}$, $\mathtt{addr'}_{rsmc0,AB}$ and $\mathtt{addr}_{\mathtt{comm0},AB}$
(2) The following transactions are generated:

- *Funding transaction*: $\mathtt{tx}_{\langle\mathtt{fund},C\rangle} = tx\Big([\mathtt{pk}_{\mathtt{fund},A}, \mathtt{pk}_{\mathtt{fund},B}], \mathtt{addr}_{\mathtt{fund},AB}, 0\Big)$
- *Initial commitment transaction*: $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle} = tx\Big(\mathtt{addr}_{\mathtt{fund},AB}, [\mathtt{addr}_{rsmc0,AB}, \mathtt{pk}_{\mathtt{comm0},B}], 0\Big)$, $\mathtt{tx}^{\mathtt{B},0}_{\langle\mathtt{commit},C\rangle} = tx\Big(\mathtt{addr}_{\mathtt{fund},AB}, [\mathtt{pk}_{\mathtt{comm0},A}, \mathtt{addr'}_{rsmc0,AB}], 0\Big)$, $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle} = tx\Big(\mathtt{addr}_{rsmc0,AB}, \mathtt{pk}_{\mathtt{rsmc0},A}, +t\Big)$, and $\mathtt{tx}^{\mathtt{B},0}_{\langle\mathtt{spend},C\rangle} = tx\Big(\mathtt{addr'}_{rsmc0,AB}, \mathtt{pk}_{\mathtt{rsmc0},B}, +t\Big)$.

(3) A and B exchanges $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$ and $\mathtt{tx}^{\mathtt{B},0}_{\langle\mathtt{commit},C\rangle}$ with each other. B signs $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$, sends the signature $\sigma_{\mathtt{comm0},B}$ to A, and A signs $\mathtt{tx}^{\mathtt{B},0}_{\langle\mathtt{commit},C\rangle}$, sends the signature $\sigma_{\mathtt{comm0},A}$ to B. Note that $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$ (resp. $\mathtt{tx}^{\mathtt{B},0}_{\langle\mathtt{commit},C\rangle}$) spends from a multi-sig address $\mathtt{addr}_{\mathtt{fund},AB}$ so it would need signature of B (resp. A) as well. Next, A and B sign transaction $\mathtt{tx}_{\langle\mathtt{fund},C\rangle}$ individually, with A generating $\sigma_{\mathtt{fund},A}$, and B generating $\sigma_{\mathtt{fund},B}$. They exchange these signatures with each other. Either A or B posts $\mathtt{tx}_{\langle\mathtt{fund},C\rangle}$ on $\mathbb{B}$.

Channel Update

For a j^{th} channel update where v_a and v_b are the channel balances of A and B respectively:
(1) Parties use $\mathtt{KGen}(1^\lambda)$ for generating the following keys: A generates $(\mathtt{pk}_{\mathtt{commj},A}, \mathtt{sk}_{\mathtt{commj},A}), (\mathtt{pk}_{\mathtt{rsmcj},A}, \mathtt{sk}_{\mathtt{rsmcj},A})$ and B generates $(\mathtt{pk}_{\mathtt{commj},B}, \mathtt{sk}_{\mathtt{commj},B}), (\mathtt{pk}_{\mathtt{rsmcj},B}, \mathtt{sk}_{\mathtt{rsmcj},B})$. A and B jointly generate a 2-of-2 multi-sig addresses $\mathtt{addr}_{\mathtt{commj},AB}$
(2) *Generate j^{th} commitment transaction*: $\mathtt{tx}^{\mathtt{A},j}_{\langle\mathtt{commit},C\rangle} = tx\Big(\mathtt{addr}_{\mathtt{fund},AB}, [\mathtt{addr}_{rsmcj,AB}, \mathtt{pk}_{\mathtt{commj},B}], 0\Big)$, $\mathtt{tx}^{\mathtt{B},j}_{\langle\mathtt{commit},C\rangle} = tx\Big(\mathtt{addr}_{\mathtt{fund},AB}, [\mathtt{pk}_{\mathtt{commj},A}, \mathtt{addr'}_{rsmcj,AB}], 0\Big)$, $\mathtt{tx}^{\mathtt{A},j}_{\langle\mathtt{spend},C\rangle} = tx\Big(\mathtt{addr}_{rsmcj,AB}, \mathtt{pk}_{\mathtt{rsmcj},A}, +t\Big)$, and $\mathtt{tx}^{\mathtt{B},j}_{\langle\mathtt{spend},C\rangle} = tx\Big(\mathtt{addr'}_{rsmcj,AB}, \mathtt{pk}_{\mathtt{rsmcj},B}, +t\Big)$.
(3) A and B exchanges $\mathtt{tx}^{\mathtt{A},j}_{\langle\mathtt{commit},C\rangle}$ and $\mathtt{tx}^{\mathtt{B},j}_{\langle\mathtt{commit},C\rangle}$ with each other. B signs $\mathtt{tx}^{\mathtt{A},j}_{\langle\mathtt{commit},C\rangle}$, sends signature $\sigma_{\mathtt{commj},B}$ to A, and A signs $\mathtt{tx}^{\mathtt{B},j}_{\langle\mathtt{commit},C\rangle}$, sends signature $\sigma_{\mathtt{commj},A}$ to B. Next, A shares revocation secret r_a^{j-1} with B, and B shares revocation secret r_b^{j-1} with A to invalidate the $(j-1)^{th}$ state of the channel.

Channel Closing

Each party can close the channel at j^{th} unrevoked state:
(1) *If A and B mutually decide to close the channel*: Revoke transactions $\mathtt{tx}^{\mathtt{A},j}_{\langle\mathtt{commit},C\rangle}$ and $\mathtt{tx}^{\mathtt{B},j}_{\langle\mathtt{commit},C\rangle}$ and create one transaction $\mathtt{tx}_{\langle\mathtt{close},C\rangle} = tx\Big(\mathtt{addr}_{\mathtt{fund},AB}, [\mathtt{pk}_{\mathtt{commj},A}, \mathtt{pk}_{\mathtt{commj},B}], 0\Big)$. Publish $\mathtt{tx}_{\langle\mathtt{close},C\rangle}$ on-chain.
(2) *If A (resp. B) unilaterally closes the channel*: Publish $\mathtt{tx}^{\mathtt{A},j}_{\langle\mathtt{commit},C\rangle}$ (resp. $\mathtt{tx}^{\mathtt{B},j}_{\langle\mathtt{commit},C\rangle}$) and $\mathtt{tx}^{\mathtt{A},j}_{\langle\mathtt{spend},C\rangle}$ (resp. $\mathtt{tx}^{\mathtt{B},j}_{\langle\mathtt{spend},C\rangle}$) on-chain.
(3) *If A publishes an old state:*

(a) B generates the address $\mathtt{pk}_{j,B}$ and also the following transactions - $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle} = tx\Big(\mathtt{addr}_{rsmc0,AB}, \mathtt{pk}_{j,B}, 0\Big)$, $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle} = tx\Big(\mathtt{addr}_{rsmc0,AB}, \_, 0\Big)$.

(b) B can post $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ using secret r_a^0 on $\mathbb{B}$ before $+t$ elapses. Miners uses the secret r_a^0 to post $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ on $\mathbb{B}$. So the secret r_a^0 allows B to immediately spend the output of $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$ before A spends the coins via transaction $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$.

---

[a] Hash of the public key is used as addresses, but we ignore such details for a simplified explanation.

Figure 5 Pseudocode for `CRAB`

in-depth analysis yields similar bounds for `LC` channels (albeit necessitating a channel reserve of $\frac{v}{2}$). However, due to the lack of collateral, `LC` channels cannot be secure against Byzantine counterparties as an attacker can simply bribe the full channel amount he owns. Also, recall that `LC` channels cannot be secure against rational counterparties and miners without monitoring the mempool.

Recall the setting we used for LC where A tries to close the channel by publishing the old state $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$. Before the relative timelock $+t$ expires, only $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ and $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ can be published. Let us look at the conditions under which including $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ in the blocks becomes the dominant strategy for the miners in the presence of a rational attacker. The fee offered for $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$ will not exceed v as a rational attacker will choose not to lose the collateral c.

Let M be any miner. We say that M has a mining power λ, expressed as the percentage of the total mining power. We analyze any point in time between posting $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$ and the timelock expiring. We represent the time period $+t$ in terms of number of blocks, denoted as k. One must wait for block height to increase by k blocks after $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$ is posted on-chain, only then $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$ becomes valid. Further, we say that F is the maximum fee earned for a block without either $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ and $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$. If we replace one normal transaction in the block with $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$, then $F_c := F - f + c$ is the maximum amount of fees earned for mining a block containing $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$. Similarly, on replacing a normal transaction in the block with $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$, $F_v := F - f + v$ is the fee earned for a block containing $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$.

If $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ has already been included; this means that B will get back v coins by posting $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$, i.e., balance security holds. Similarly, if there are other miners whose strategy is to include $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ in these upcoming k blocks, B is compensated and balance security ensured. We thus focus on the corner case where no other miner will include $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$. We compute the expected payoff of not including $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ and instead try to include $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$ in the first block after the timelock expires. For any miner M, the expected number of blocks mined until the timeout is $k\lambda$ of the k remaining blocks. Thus, the expected payoff is $k\lambda F + \lambda F_v$. To see what is the dominant strategy, we compare this to the expected payoff of including $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$. For this, we consider the following two cases.

Case 1: $k\lambda \geq 1$. M has mining power such that it is expected to mine at least one block in the k remaining slots until the timelock expires. Because we know that $k\lambda \geq 1$, the expected payoff for including $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ is $F_c + (k\lambda - 1)F + \lambda F$. Any such miner M will include the punishment if the following inequality holds.

$$F_c + (k\lambda - 1)F + \lambda F > k\lambda F + \lambda F_v \implies c - f > \lambda(v - f) \tag{1}$$

Since the fee f is negligible compared to v and c, we can rewrite the inequality $c > \lambda v$. We observe that the collateral c must exceed M's proportionate share of the total value v, such that it is more profitable for M to include $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$. Since we consider the underlying blockchain secure, we know that $\lambda < 0.5$ holds for any M. Thus, if $c = \frac{v}{2}$, the dominant strategy for *any* miner with $k\lambda \geq 1$ is to include $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$.

Case 2: $k\lambda < 1$. M's mining power is such that it is expected to mine fewer than one block in the k remaining slots. The expected payoff for including $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ is $k\lambda F_c + \lambda F$. Again, such a miner M will include the punishment if the following inequality holds.

$$k\lambda F_c + \lambda F > k\lambda F + \lambda F_v \implies c - f > \frac{v - f}{k} \tag{2}$$

From case 1, we observed that setting $c = \frac{v}{2}$ would be enough for miners to choose the punishment transaction $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ over $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$. Given that $c = \frac{v}{2}$ and fee f is negligible, setting $k > 2$ ensures that $c > \frac{v}{k}$. We can merge case 1 and case 2 and write $c > max\left(\lambda v, \frac{v}{k}\right)$.

Since the least value of k is 3, and the strongest miner may have mining power more than $\frac{1}{3}$, setting $c = \frac{v}{2}$ is sufficient collateral to disincentivize cheating in both the cases.

To make matters worse, however, the strongest miner can announce a feather-forking attack for $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$, disincentivizing every other miner from including $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$. But then the strongest miner's mining power does not exceed 0.5, so the expected payoff of the strongest miner will be strictly less than $\frac{v}{2}$ upon choosing $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$. Thus $c = \frac{v}{2}$ is a tight bound on the collateral when the participants and the miners are rational.

▶ **Corollary 10.** *Assuming rational miners and rational parties, balance security is satisfied in `CRAB`, if the honest party is online, and the collateral locked by each party is equal to half the channel capacity $c = v/2$.*

If the attacker is Byzantine, the maximum amount she can bribe is $v + c$. Ignoring fee f, if we replace v by $v + c$ in Equation (1) and in Equation (2), we get $c > max\Big(\lambda(v + c), \frac{v+c}{k}\Big)$. Given $max(\lambda, \frac{1}{k}) < 0.5$, a collateral $c = v$ is necessary to prevent timelock bribing if A is malicious and miners are rational.

▶ **Corollary 11.** *Assuming rational miners and Byzantine parties, balance security is satisfied in `CRAB`, if the honest party is online, and collateral locked by each party is equal to the channel capacity $c = v$.*

Corollary 10 and Corollary 11 further imply that balance security holds without parties monitoring the mempool. Further, as we have pointed out that $v/2$ and v are the lower bounds for the settings where counterparties are rational and Byzantine, respectively, our construction is collateral optimal.

D Sleepy `CRAB`

D.1 Protocol Description

The channel design is the same as `CRAB`. The only difference here is that the honest party is offline and miners need to post revoke transactions by themselves. If B wants to go offline after the m^{th} state update, he puts all the revocation secrets $r_a^0, r_a^1, \ldots, r_a^{m-1}$ on a public bulletin board (PBB). If A posts any of the old states after B has gone offline, then the miner selects the appropriate revocation secret from the bulletin board and publishes the revocation transaction to claim A's collateral. Later, when B comes online, he can post his revocation transaction to claim $A's$ deposit. To improve efficiency, we discuss how users can safely go offline without dumping all the revocation secrets into PBB. This can be achieved through posting a minimum amount of information on the blockchain. Since there could be multiple channel participants who might want to go offline at the same time, their individual channel's revocation secret can be aggregated and put into one single transaction.

Using secret derivation. To achieve constant storage cost for channels, we should guarantee that anyone with the current revocation secret can derive the previous revocation secrets but should not be able to generate any future revocation secret. There exist techniques from the payment channel and watchtower literature to store revocation secrets efficiently. Trapdoor one-way functions are used in [47] to implement a scheme that allows for constant storage of secrets per channel. The construction does not require any modification on the core of the current Bitcoin system or Lightning Network. The trapdoor one-way functions are easy to compute but hard to invert without the knowledge of the secret or *trapdoor td*. We define the function as f_{td} where $y \leftarrow f_{td}(x)$. y could be derived from x. If a person has the knowledge of td, then he or she can compute $x \leftarrow f_{td}^{-1}(td, y)$.

A channel participant who wishes to go offline will post the revocation secret of the last revoked state. No one except him can derive the future revocation secret from this information.

We define the interface for revocation secret generation and derivations in `Sleepy CRAB`:
(a) `GenerateRevokeSecret`(y, td, i): Given the revocation secret y for the current channel state, and the knowledge of trapdoor td, the revocation secret for i^{th} state, we define $y_j \leftarrow f_{td}^{-1}(td, y_{j-1})$ for $1 \leq j \leq i$ where $y_0 = y$.
(b) `DeriveRevokeSecret`(x, k, i): Given the revocation secret x of channel state k, to derive the revocation secret of the i^{th} channel state where $0 \leq i < k$, we define $y_{j-1} \leftarrow f_{td}(y_j)$ for $i+1 \leq j \leq k$ where $y_k = x$.
The authors have used RSA cryptosystem in [47], one of the famous trapdoor one-way functions. A party must post the RSA public key and the revocation secret of the last revoked state on-chain before going offline. Given that the size of the RSA modulus is 2048 bits (256 bytes), as per the experimental results shown in [47], the estimated storage overhead for storing the public key and revocation secret is close to 600 bytes. If we take the Bitcoin transaction fee of 7 satoshi per byte [4] and a current price of roughly 26.9k USD/BTC [3], then the fee for storing this information would be 1.13 USD.

Aggregating revocation secrets and posting it on-chain. Let us now more efficiently utilize the blockchain on which the payment channels are deployed, and thus, a blockchain that we know that miners are reading. For instance, this can be implemented in Bitcoin by posting a balance-neutral transaction (i.e., A transferring coins to herself), which has an additional zero-value output with `OP_RETURN` storing the revocation secret. To make it easily identifiable to miners, A can add an identifier marking this transaction as holding such information and possibly identifying the channel's funding transaction.

Clearly, it is not desirable to post an on-chain transaction and thus the associated fees every time one wishes to go offline. We therefore propose the following two improvements. Multiple users can create a joint transaction, which, instead of holding the secret of one channel participant, holds the secret of multiple channel participants. This can be implemented easily, using an *untrusted* centralized service. Note that this service does not need to be trusted since a user can check if her secret appears on the blockchain before going offline. We mentioned previously that the storage overhead of one secret is close to 600 bytes. Assuming a transaction size limit of 400kb, up to roughly 600 users can put their secrets in a single transaction, splitting the fee among themselves and avoiding overhead which would be present if there were 600 individual transactions. Again, note that one secret per channel is enough to cover the whole channel history and users only need to post the secret when they wish to go offline.

Using the blockchain's network layer. It is important to highlight that posting the revocation information on-chain is a way to ensure that miners are aware of it. It suffices, however, to choose any mechanism that transfers this information to the miners, e.g., posting it online in a forum or using the blockchain's network layer. The security of consensus protocols, e.g., of Bitcoin or Ethereum, typically relies on a synchrony assumption, i.e., messages are delivered in a timely manner [26]. This synchrony, which in practice is realized through flooding in Bitcoin, suffices to ensure that miners see this information when users post it to the network, therefore ensuring this construction. The compensation of miners for storing this information is less straightforward than when posting the information on-chain, but this is an orthogonal and known problem in the watchtower literature, e.g., [34, 17, 1].

D.2 Analysis of `Sleepy CRAB`

This construction is the same as `CRAB`, except for the derivation of revocation secrets. Thus, the analysis of Appendix C transfers to `Sleepy CRAB`. We have the same security guarantee as `CRAB` for rational and Byzantine attackers but without assuming the honest party is online.

▶ **Corollary 12.** *Assuming rational miners and rational parties, balance security is satisfied in `Sleepy CRAB`, even when parties are offline if the collateral locked by each party is equal to half the channel capacity* $c = v/2$.

▶ **Corollary 13.** *Assuming rational miners and Byzantine parties, balance security is satisfied in `Sleepy CRAB`, even when parties are offline if the collateral locked by each party is equal to the channel capacity* $c = v$.

D.3 Interplay with Lightning channels

`Sleepy CRAB` can be used alongside Lightning channels in an agile way. Users can use Lightning channels, until they wish to go offline, at which point they simply change to `Sleepy CRAB`, using a technique known as *splicing* [39]. Splicing allows users to increase or decrease the channel capacity with an on-chain transaction. This can be thought of as closing the old and simultaneously opening a new channel, with a different capacity. Indeed, we can use this technique to change the nature of the channel to `Sleepy CRAB`, by adding the necessary collateral and logic (or else change it back to Lightning). We discuss the construction in Section G.2 of the Appendix.

E Evaluation

To evaluate our construction and show its practical feasibility, we build a proof-of-concept implementation of `CRAB`. Since `Sleepy CRAB` and `CRAB` are the same except for the derivation of revocation secret, the implementation holds true for `Sleepy CRAB`, and from here onwards, we refer to it merely as the evaluation for `Sleepy CRAB`. This implementation creates the necessary transactions for deploying our construction with the following goals in mind: (i) measure the overhead both on-chain and off-chain, (ii) compare it with existing constructions, and (iii) demonstrate its compatibility with Bitcoin by publishing the transactions on the Bitcoin testnet. More concretely, we compare our results with Lightning Network (LN) channels [38], Generalized channels (GC) [9], and Sleepy channels [13]. The code of our implementation can be found in a public GitHub repository [6].

We evaluate the following phases: open, update, punish, unilateral close, and cooperative close. The update phase happens completely off-chain, for the other phases we also estimate on-chain costs. For this, we take a current Bitcoin transaction fee of 7 satoshi per byte [4] and a current price of roughly 26.9k USD/BTC [3]. This allows us to accurately compute the current estimated on-chain fees in USD. The funding transaction and, therefore, the (on-chain part of the) opening phase is the same for all of these constructions, essentially a transaction with two inputs and one output. The off-chain part of the opening phase is analogous to the update phase. It has a size of 338 bytes which results in approximately 0.64 USD in on-chain fees. Similarly, the cooperative closure phase is the same for all constructions, spending the funding transaction's output and generating two new outputs. It has a size of 225 bytes, which is approximately 0.42 USD in on-chain fees.

For the other three phases, we show our results and comparison in Table 2. We take the numbers for LN, GC, and Sleepy from the evaluation in [13, 9]. For `Sleepy CRAB`, we investigate the following transactions. The funding transaction has 338 bytes. The commitment transaction has 457 bytes. The punish transaction has 192 bytes. Finally, the payment transaction has 418 bytes. To carry out an update, we require exchanging two commitment transactions, as well as the pre-signed payment transactions. This results in 4 transactions or 1750 bytes exchanged. Note that additionally, we need to exchange the revocation key (32 bytes). We omit this in the table for all constructions, since we focus on the transactions themselves. In practice, two of these keys, but also some other messages specific to how the protocol is implemented, need to be exchanged.

For a punishment, one user needs to post a commitment transaction, and the other user needs to publish a punishment transaction. This totals 649 bytes or 1.22 USD in on-chain fees. For the unilateral close, one user also needs to publish a commitment transaction, followed by a payment transaction, totaling 875 bytes or 1.64 USD.

From these results, we can see that `Sleepy CRAB` is a very practical scheme. Its on-chain overhead is comparable to the other channel constructions, both for punishing and unilateral closure. The off-chain communication overhead is higher than [38] or [9], but lower than [13]. All in all, `Sleepy CRAB` is cheap to deploy and as we show, compatible with the current Bitcoin implementation, which implies that it is also compatible with other cryptocurrencies which have limited scripting capabilities.

Table 2 Results of our evaluation and comparison to existing schemes: Lightning Network (LN), Generalized (GC), and Sleepy channels.

	update		punish			unilateral close		
	# txs	bytes	# txs	bytes	USD	# txs	bytes	USD
LN	2	706	2	513	0.97	2	511	0.96
GC	2	695	2	663	1.25	2	695	1.31
Sleepy (fast)	10	2408	2	450	0.85	2 (3)	449 (823)	0.85 (1.55)
`Sleepy CRAB`	4	1750	2	649	1.22	2	875	1.64

F Analysis of CRAB and Sleepy CRAB with relative timelocks

We use the single miner assumption for analysis of CRAB and sleepy CRAB with relative timelocks.

F.1 Rational Analysis of `CRAB`

We represent `CRAB` as an extensive form game with $N = \{A, B, M\}$ illustrated as a game tree $\Gamma_{\mathtt{CRAB},T}$ in Figure 6. The action set of the players is as follows: player A selects her action from S_A ={*latest state*, *old state with bribe* $f < p < c$, *old state with bribe* $p = c$, *old state with bribe* $p = c + \delta$}, where $\delta > \epsilon$, and ϵ is the opportunity cost. B selects his action from S_B ={*punish, not punish*} and miner M selects its actions from {accept, reject}. The game starts with A, selecting an action s from set S_A. Next, B can choose to *punish* A and reveal the revocation secret r_a^0, or *not punish* A. If B chooses to *punish* A, the latter will offer a bribe p for mining $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$. In the next step, M decides whether to *accept* or *reject* the bribe offered by A. We observe that the elements depicted in the extensive form game provide a comprehensive representation of the game, showing the sequence of decision-making, the set of feasible actions at each stage, and the consequent utilities for each player.

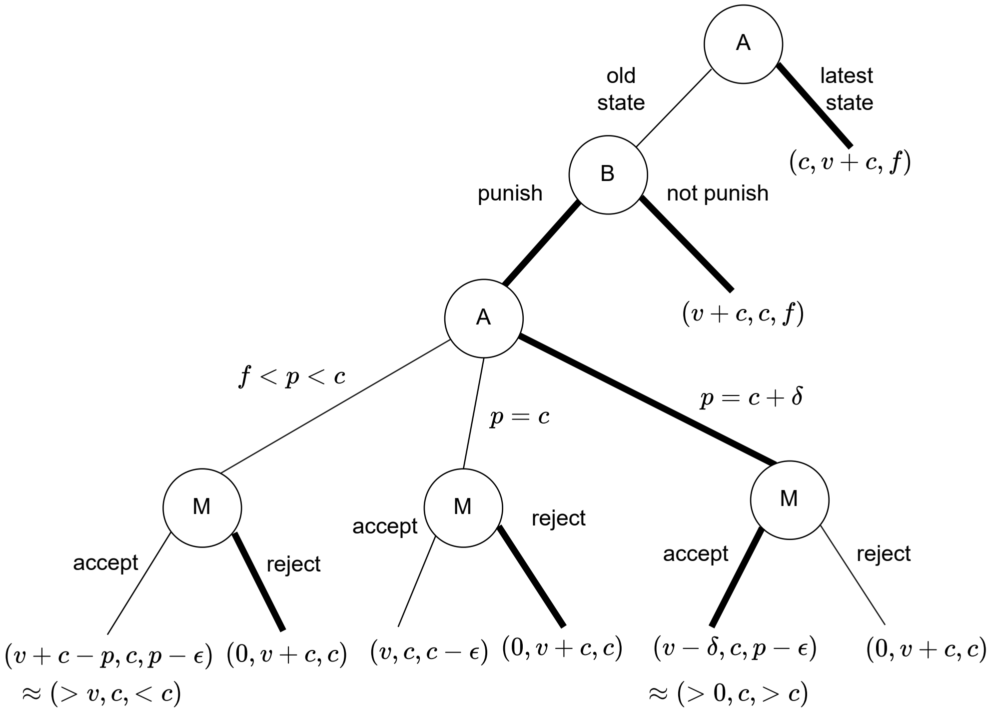

Figure 6 SPNE upon applying backward induction on $\Gamma_{\mathtt{CRAB},T}$

Payoff Structure. We explain the payoff as illustrated in Figure 6:

(i) If A publishes the old state $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{commit},C\rangle}$, then the following situation arises:

(a) B punishes A by publishing $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$: A bribes miners so that $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$ is selected. We analyze the following cases:

  - A offers a bribe $f < p < c$: If M chooses to accept then it gets a fee less than c but if M rejects the bribe and mines $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$, it gets payoff u_M((old state, bribe f < p < c)$, punish, reject) = c$, and B gets u_B((old state, bribe $f < p < c$), punish, reject) $= v + c$.
  - A offers a bribe $p = c$: If M chooses to accept then it gets a fee less than c, due to loss of opportunity cost. If M rejects the bribe, the payoff is u_M((old state, bribe $p = c$), punish,reject)=c. Payoff of A and B are as follows: u_A((old state, bribe $p = c$), punish,accept)=v, u_B((old state, bribe $p = c$), punish,accept)=c, and u_A((old state, bribe $p = c$), punish,reject)=0, u_B((old state, bribe $p = c$), punish,reject)=$c + v$.
  - A offers a bribe $p = c + \delta$: If M accepts the bribe, it gets payoff more than c and A earns a payoff $v - \delta$. If M rejects the bribe, A earns payoff 0

(b) B does not punish A: u_A(old state, not punish) $= v + c$, u_B(old state,not punish) $= c$ and u_M(old state, not punish) $= f$.

(ii) If A publishes the latest state, u_A(latest state) $= c$, u_B(latest state) $= v + c$ and u_M(latest state) $= f$.

Desired Protocol Execution. Our desired protocol execution is A chooses to publish *latest state* on-chain, and B chooses to *punish* A when it posts an old channel state. Equipped with this model, we will prove that our intended protocol execution is a subgame perfect Nash Equilibrium (SPNE). Subgame Perfect Nash Equilibrium (SPNE) is a refinement of the concept of Nash Equilibrium for extensive form games where players act sequentially.

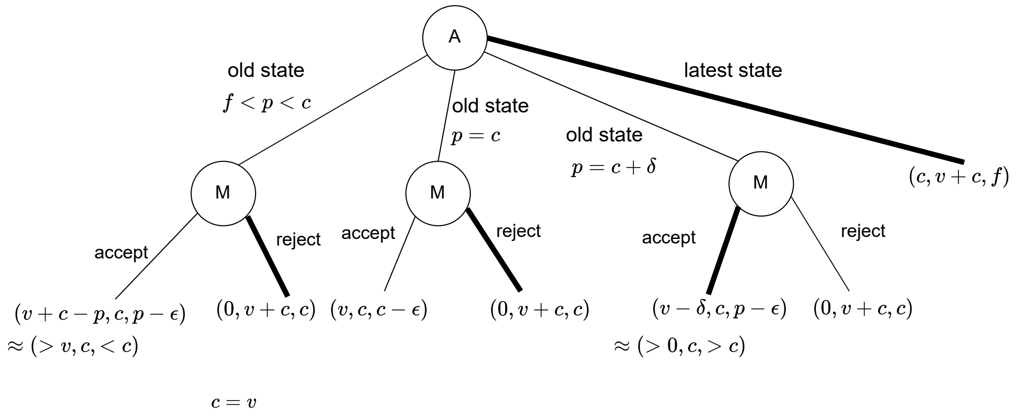

Figure 7 SPNE upon applying backward induction on $\Gamma_{\texttt{Sleepy CRAB},T}$

We assume that B can choose to punish A with probability q or not to punish with probability $1 - q$, where $q \in [0, 1]$.

▶ **Theorem 14.** *Given that $c = \frac{v}{q}$, the strategy profile $s^*(A, B, M)$= ((*latest state*, bribe $p = c + \delta$), (*punish with probability $q \in [0, 1]$, not punish with probability $1 - q$*), (reject, reject, accept)) is a Subgame Perfect Nash Equilibrium for our game.*

Proof. We prove that strategy profile $s^*(A, B, M)$ is SPNE using backward induction on $\Gamma_{\texttt{CRAB}}$. If A *posts an old state*, she should ensure that M mines the transaction. She will offer a fee $p = c + \epsilon$ and miners will choose to accept the fee as it is more than c. When the fee is less than c, the miners will choose to reject over accept. If $p = c$, M rejects the bribe as it gets a fee c instantly rather than waiting and losing the opportunity cost. When $p = c + \delta$ where $\delta > \epsilon$, M gets a payoff $c + \delta - \epsilon$ which is greater than c, so M will accept the bribe. A will offer a bribe $p = c + \delta$ and she gets the payoff $v - \delta$. If the miner chooses to accept the bribe and mines $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{spend},C\rangle}$, then B gets a payoff of c. If B chooses *not to punish* A, he still gets a payoff of c. So B remains indifferent between choosing to punish and not punish. A believes that B has probability q of choosing *punish* (and with probability $1 - q$ he will choose not to punish), so her payoff will be $q(v - \delta) + (1 - q)(v + c) = v + (1 - q)c - q\delta$. If we want A to choose *latest state* over the old state then $v + (1 - q)c - q\delta < c$. In other words, $c > \frac{v}{q} - \delta$, so if we set $c = \frac{v}{q}$ then we can say the strategy profile $s^*(A, B, M)$= ((*latest state*, bribe $p = c + \delta$), (*punish with probability* $q \in [0, 1]$, *not punish with probability* $1 - q$), (reject, reject, accept)) is a Subgame Perfect Nash Equilibrium for our game. The selected strategies are shown using black arrow in Figure 6 on the tree $\Gamma_{\texttt{CRAB},T}$. ◀

F.2 Rational Analysis of `Sleepy CRAB`

We represent `Sleepy CRAB` as an extensive form game with $N = \{A, M\}$ illustrated as a game tree $\Gamma_{\texttt{Sleepy CRAB}}$ in Figure 7. The action set of the players is as follows: player A selects her action from S_A =\{*latest state, old state with bribe $f < p < c$, old state with bribe $p = c$, old state with bribe $p = c + \delta$*\}, and miner M select its action from \{*accept, reject*\}. The game starts with A, selecting an action s from set S_A. Next, M can choose to *accept* the bribe from A and mine $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{spend},C\rangle}$, or *reject* the bribe and mine $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{revoke},C\rangle}$. Since B is offline, it has no role in the game.

Payoff Structure. We explain the payoff as illustrated in Figure 11:

(i) If A publishes the old state $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{commit},C\rangle}$, then the following situation arises:

- A offers a bribe $f < p < c$: If M chooses to accept then it gets a fee less than c but if M rejects the bribe and mines $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$, miner gets payoff $u_M((\text{old state, bribe } f < p < c), \text{reject}) = c$, B gets payoff $v + c$, and A gets 0.
- A offers a bribe $p = c$: If M accepts the bribe, it earns a payoff less than c. If M rejects the bribe, it gets the payoff is $u_M((\text{old state, bribe } p = c), \text{reject})=c$. Payoff of A are as follows: $u_A((\text{old state, bribe } p = c), \textit{accept}) = v$, $u_B((\text{old state,bribe } p = c), \text{accept}) = c$, and $u_A((\text{old state, bribe } p = c),\text{reject})=0$, $u_B((\text{old state, bribe } p = c), \text{reject}) = c + v$.
- A offers a bribe $p = c + \delta$: If M accepts the bribe, it gets payoff more than c and A earns a payoff $v - \epsilon$. If M rejects the bribe, A earns payoff 0.

(ii) If A posts the latest state, $u_A(\text{latest state}) = c$, $u_B(\text{latest state}) = c + v$, and $u_M(\text{latest state}) = f$.

Desired Protocol Execution. Our desired protocol execution is A choosing the strategy *latest state* upon channel closure, and M decides to *punish* when A publishes the old state and offers a bribe less than c. We will prove our intended protocol execution is a subgame perfect Nash Equilibrium (SPNE).

▶ **Theorem 15.** *Given that $c = v$, the strategy profile $s^*(A, M)= ($*latest state*, (reject, reject, accept)) is a Subgame Perfect Nash Equilibrium for our game.*

Proof. We use backward induction on $\Gamma_{\text{Sleepy }\mathtt{CRAB},T}$ as shown in Figure 7. If A *posts an old state* and offers a bribe less than c coins, miners will reject the bribe, mine $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$ and earn the collateral c. If A offered a bribe of more than c coins, then M will accept the bribe from A. If the bribe offered is c, then M will choose to punish A and reject the bribe. When M decides to mine $\mathtt{tx}^{\phi\mathtt{A},0}_{\langle\mathtt{revoke},C\rangle}$, A earns payoff 0. The only time M decides not to punish A is when it gets a fee $c + \delta - \epsilon$ coins. However, A would earn a payoff of at most $v - \delta$ coins. The payoffs for both cases are less than the payoff A would get if she chooses the latest state and gets back her collateral c, if $c = v$.

◀

G Analysis after removal of relative timelocks from `CRAB` and `Sleepy CRAB`

Cryptocurrencies like Monero do not possess the capability for relative timelock in their script. To adapt `CRAB` for a wide range of cryptocurrencies supporting only signatures, we can get rid of the timelocks and rely on the miners to mine the most profitable transactions. Except for no relative timelock on the spending transaction, the transaction scheme remains the same as shown in Figure 2. Since we have no timelocks in this construction, we cannot use the analysis of Appendix C, and instead use the (weaker) single miner assumption and EFG-based analysis of Section 3.1.

G.1 Rational Analysis of `CRAB`

We represent `CRAB` as an extensive form game with $N = \{A, B, M\}$ illustrated as a game tree $\Gamma_{\mathtt{CRAB}}$ in Figure 9. The action set of the players is as follows: player A selects her action from $S_A =\{$*latest state, old state with bribe $f < p < c$, old state with bribe $p = c$, old state with bribe $p = c + \epsilon$*$\}$, B selects his action from $S_B =\{$*punish, not punish*$\}$ and M selects its actions from $\{$accept, reject$\}$. The game starts with A, selecting an action s from set S_A. Next, B can choose to *punish* A and reveal the revocation secret r_a^0, or *not punish* A. If B chooses to *punish* A, the latter will offer a bribe p for mining $\mathtt{tx}^{\mathtt{A},0}_{\langle\mathtt{spend},C\rangle}$. In the next step,

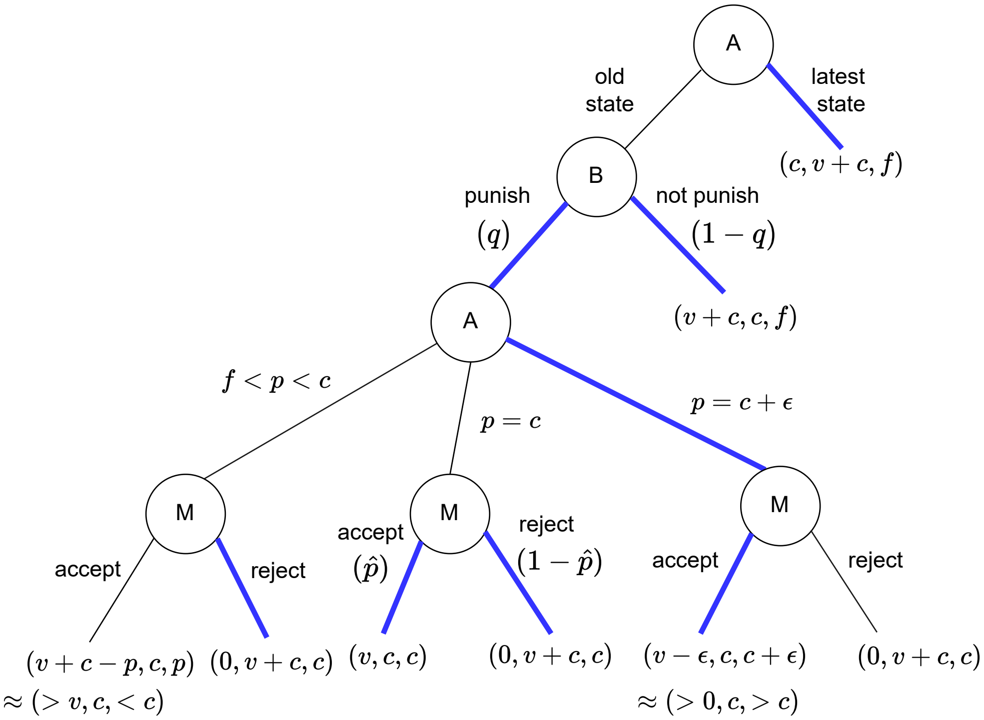

Figure 8 SPNE upon applying backward induction on $\Gamma_{\texttt{CRAB}}$ (in absence of relative timelock)

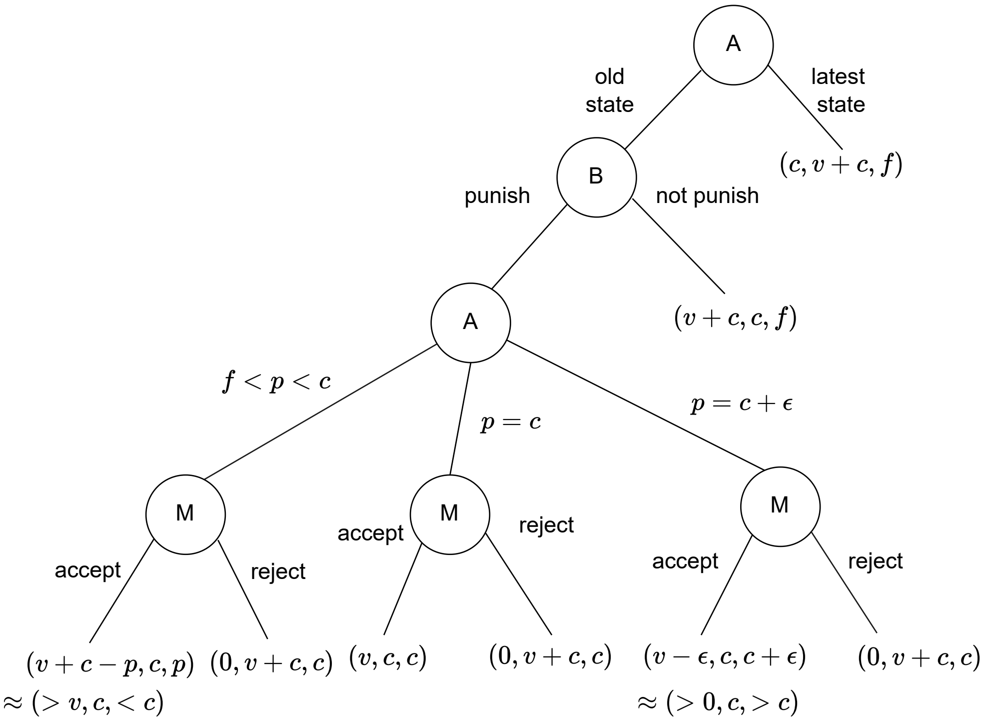

Figure 9 `CRAB` as EFG $\Gamma_{\texttt{CRAB}}$

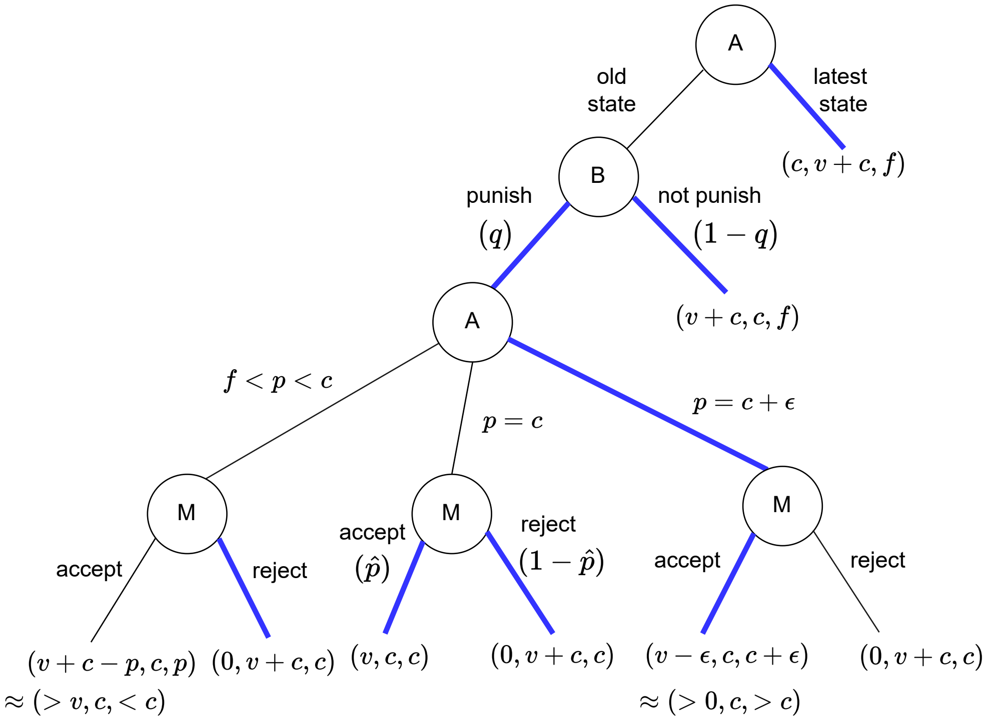

Figure 10 SPNE upon applying backward induction on $\Gamma_{\texttt{CRAB}}$ (in absence of relative timelock)

M decides whether to *accept* or *reject* the bribe offered by A. We observe that the elements depicted in the extensive form game provide a comprehensive representation of the game, showing the sequence of decision-making, the set of feasible actions at each stage, and the consequent utilities for each player.

Payoff Structure. We explain the payoff as illustrated in Figure 9:

(i) If A publishes the old state $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{commit},C\rangle}$, then the following situation arises:

(a) B punishes A by publishing $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{revoke},C\rangle}$: A bribes miners so that $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{spend},C\rangle}$ is selected. We analyze the following cases:

- A offers a bribe $f < p < c$: If M chooses to accept then it gets a fee less than c but if M rejects the bribe and mines $\texttt{tx}^{\phi\texttt{A},0}_{\langle\texttt{revoke},C\rangle}$, it gets payoff $u_M((oldstate, bribef < \text{p} < \text{c}), punish, reject) = c$, and B gets $u_B((\text{old state, bribe } f < p < c), \text{punish, reject}) = v + c$.
- A offers a bribe $p = c$: M can now choose to accept or reject the bribe as there is no relative timelock on spending $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{spend},C\rangle}$. In both the cases the payoff is $u_M((\text{old state, bribe } p = c), \text{punish,accept}) = u_M((\text{old state, bribe } p = c), \text{punish,reject}) = c$. Payoff of A and B are as follows: $u_A((\text{old state, bribe } p = c), \text{punish,accept}) = v$, $u_B((\text{old state, bribe } p = c), \text{punish,accept}) = c$, and $u_A((\text{old state, bribe } p = c), \text{punish,reject}) = 0$, $u_B((\text{old state, bribe } p = c), \text{punish,reject}) = c + v$.
- A offers a bribe $p = c + \epsilon$: If M accepts the bribe, it gets payoff more than c and A earns a payoff $v - \epsilon$. If M rejects the bribe, A earns payoff 0

(b) B does not punish A: $u_A(\text{old state, not punish}) = v + c$, $u_B(\text{old state,not punish}) = c$ and $u_M(\text{old state, not punish}) = f$.

(ii) If A publishes the latest state, $u_A(\text{latest state}) = c$, $u_B(\text{latest state}) = v + c$ and $u_M(\text{latest state}) = f$.

Desired Protocol Execution. Our desired protocol execution is A chooses to publish *latest state* on-chain, and B chooses to *punish* A when it posts an old channel state. Equipped with this model, we will prove that our intended protocol execution is a subgame perfect Nash Equilibrium (SPNE). Subgame Perfect Nash Equilibrium (SPNE) is a refinement of the concept of Nash Equilibrium for extensive form games where players act sequentially.

If there is no relative timelock on spending $\mathtt{tx}^{\mathtt{A},0}_{\langle \mathtt{spend},C\rangle}$ then M can choose to either accept or reject $\mathtt{tx}^{\mathtt{A},0}_{\langle \mathtt{spend},C\rangle}$ if bribe $p = c$. We assume that a miner accepts $\mathtt{tx}^{\mathtt{A},0}_{\langle \mathtt{spend},C\rangle}$ with probability $\hat{p} \in [0,1]$ and rejects it with probability $1-\hat{p}$. We additionally assume that B can choose to punish A with probability q or not to punish with probability $1-q$, where $q \in [0,1]$.

▶ **Theorem 16.** *Given that* $c = \frac{v}{q}$*, the strategy profile* $s^*(A,B,M)$*= ((*latest state*, bribe* $p = c + \epsilon$ *), (*punish with probability $q \in [0,1]$, not punish with probability $1-q$*), (reject, accept with probability* $\hat{p} \in [0,1]$*, reject with probability* $1-\hat{p}$*, accept)) is a Subgame Perfect Nash Equilibrium for our game, provided there is no relative timelock.*

Proof. We prove that strategy profile $s^*(A,B,M)$ is SPNE using backward induction on $\Gamma_{\mathtt{CRAB}}$. If A *posts an old state*, she should ensure that M mines the transaction. She will offer a fee $p = c + \epsilon$ and miners will choose to accept the fee as it is more than c. When the fee is less than c, the miners will choose to reject over accept. If $p = c$, M can now either choose to either accept $\mathtt{tx}^{\mathtt{A},0}_{\langle \mathtt{spend},C\rangle}$ with probability $\hat{p}$ or reject the bribe from A with probability $1-\hat{p}$. Though we consider $\hat{p}$ to lie in the range 0 and 1, this information is not known to A, hence she would get a payoff $\hat{p}v$ upon selecting branch $p = c$. The payoffs of branch $p = c$ and $p = c + \epsilon$ are equal if $\hat{p}v = v - \epsilon$ or $\hat{p} = \frac{v-\epsilon}{v}$. As ϵ is negligible, both the brances will have equal payoff when $\hat{p} \approx 1$. Since A is not aware of M's behavior, she assumes $\hat{p}v < v - \epsilon$, and chooses $p = c + \epsilon$ to be sure that she gets the payoff $v - \epsilon$. If the miner chooses to accept the bribe and mines $\mathtt{tx}^{\mathtt{A},0}_{\langle \mathtt{spend},C\rangle}$, then B gets a payoff of c. If B chooses *not to punish* A, he gets a payoff of c. So B remains indifferent between choosing to punish and not punish. A believes that B has probability q of choosing *punish* (and with probability $1-q$ he will choose not to punish), so her payoff will be $q(v-\epsilon) + (1-q)(v+c) = v + (1-q)c - q\epsilon$. If we want A to choose *latest state* over the old state then $v + (1-q)c - q\epsilon < c$. In other words, $c > \frac{v}{q} - \epsilon$, so if we set $c = \frac{v}{q}$ then we can say the strategy profile $s^*(A,B,M)$= ((*latest state*, bribe $p = c + \epsilon$), (*punish with probability* $q \in [0,1]$, *not punish with probability* $1-q$), (reject, accept with probability $\hat{p} \in [0,1]$, reject with probability $1-\hat{p}$, accept)) is a Subgame Perfect Nash Equilibrium for our game. The selected strategies are shown using blue arrow in Figure 10 on the tree $\Gamma_{\mathtt{CRAB}}$. ◀

Since both A and B need to lock equal collateral, both would stick to choosing a collateral equal to v so that $c > v - \epsilon$.

▶ **Corollary 17.** *Assuming all participants are rational and mutually distrusting, parties opening a channel need to lock collateral as large as the channel balance to realize an* `CRAB` *if there is no relative timelock.*

G.2 Rational Analysis of `Sleepy CRAB`

We represent `Sleepy CRAB` as an extensive form game with $N = \{A, M\}$ illustrated as a game tree $\Gamma_{\text{Sleepy }\mathtt{CRAB}}$ in Figure 11. The action set of the players is as follows: player A selects her action from S_A =\{*latest state, old state with bribe* $f < p < c$, *old state with bribe* $p = c$, *old state with bribe* $p = c + \epsilon$\}, and miner M select its action from \{*accept,* *reject*}. The game starts with A, selecting an action s from set S_A. Next, M can choose to *accept* the bribe from A and mine $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{spend},C\rangle}$, or *reject* the bribe and mine $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{revoke},C\rangle}$. Since B is offline, it has no role in the game. We assume that there is no relative timelock on $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{spend},C\rangle}$.

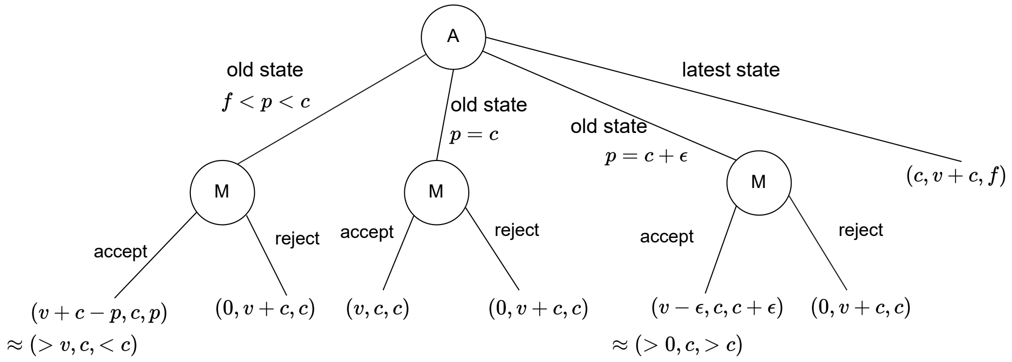

Figure 11 `Sleepy CRAB` as an EFG $\Gamma_{\texttt{Sleepy CRAB}}$

Payoff Structure. We explain the payoff as illustrated in Figure 11:
(i) If A publishes the old state $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{commit},C\rangle}$, then the following situation arises:

- A offers a bribe $f < p < c$: If M chooses to accept then it gets a fee less than c but if M rejects the bribe and mines $\texttt{tx}^{\phi\texttt{A},0}_{\langle\texttt{revoke},C\rangle}$, miner gets payoff u_M((old state, bribe $f < p < c$), reject) $= c$, B gets payoff $v + c$, and A gets 0.
- A offers a bribe $p = c$: M can now choose to accept or reject the bribe as there is no relative timelock on spending $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{spend},C\rangle}$. In both the cases the payoff is u_M((old state, bribe $p = c$), accept) $= u_M$((old state, bribe $p = c$), reject) $= c$. Payoff of A are as follows: u_A((old state, bribe $p = c$), accept) $= v$, u_B((old state,bribe $p = c$), accept) $= c$, and u_A((old state, bribe $p = c$),reject) $= 0$, u_B((old state, bribe $p = c$), reject) $= c + v$.
- A offers a bribe $p = c + \epsilon$: If M accepts the bribe, it gets payoff more than c and A earns a payoff $v - \epsilon$. If M rejects the bribe, A earns payoff 0.

(ii) If A posts the latest state, u_A(latest state) $= c$, u_B(latest state) $= c + v$, and u_M(latest state) $= f$.

Desired Protocol Execution. Our desired protocol execution is A choosing the strategy *latest state* upon channel closure, and M decides to *punish* when A publishes the old state and offers a bribe less than c. If A has posted an old state, M will choose to *punish* A when the bribe offered is more than v but less than c, or *not punish* when the bribe provided is more than c. If A offers a fee c, then M can select *punish* or *not punish* with equal probability. We will prove our intended protocol execution is a subgame perfect Nash Equilibrium (SPNE).

Given there is no relative timelock on spending $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{spend},C\rangle}$ then M can choose to either accept or reject $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{spend},C\rangle}$ if bribe $p = c$. We assume that a miner accepts $\texttt{tx}^{\texttt{A},0}_{\langle\texttt{spend},C\rangle}$ with probability $\hat{p} \in [0, 1]$ and rejects it with probability $1 - \hat{p}$.

▶ **Theorem 18.** *Given that $c = v + \epsilon$ and there is no relative timelock, the strategy profile $s^*(A, M) =$ (*latest state*, (reject, accept with probability $\hat{p}$, reject with probability $1 - \hat{p}$, accept)) is a Subgame Perfect Nash Equilibrium for our game.*

Proof. We use backward induction on $\Gamma_{\texttt{Sleepy CRAB}}$ as shown in Figure 12. If A *posts an old state* and offers a bribe less than c coins, miners will reject the bribe, mine $\texttt{tx}^{\phi\texttt{A},0}_{\langle\texttt{revoke},C\rangle}$ and earn the collateral c. If A offered a bribe of more than c coins, then M will accept the bribe

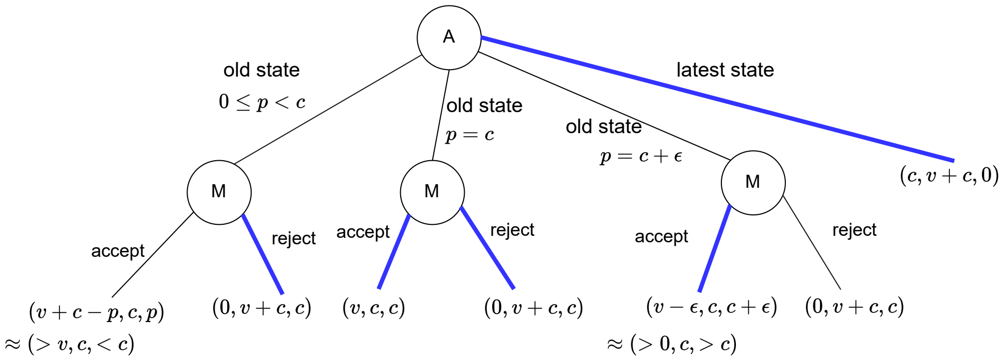

Figure 12 SPNE for $\Gamma_{\text{Sleepy }\texttt{CRAB}}$ (without relative timelock)

from A. If the bribe offered is c, then M has no preference and can choose to punish A or not to punish. When M decides to mine $\texttt{tx}^{\phi\texttt{A},0}_{\langle\texttt{revoke},C\rangle}$, A earns payoff 0. The only time M decides not to punish A is when it gets a fee $c + \epsilon$ coins. However, A would earn a payoff of at most $v - \epsilon$ coins. The payoffs for both cases are less than the payoff A would get if she chooses the latest state and gets back her collateral c.

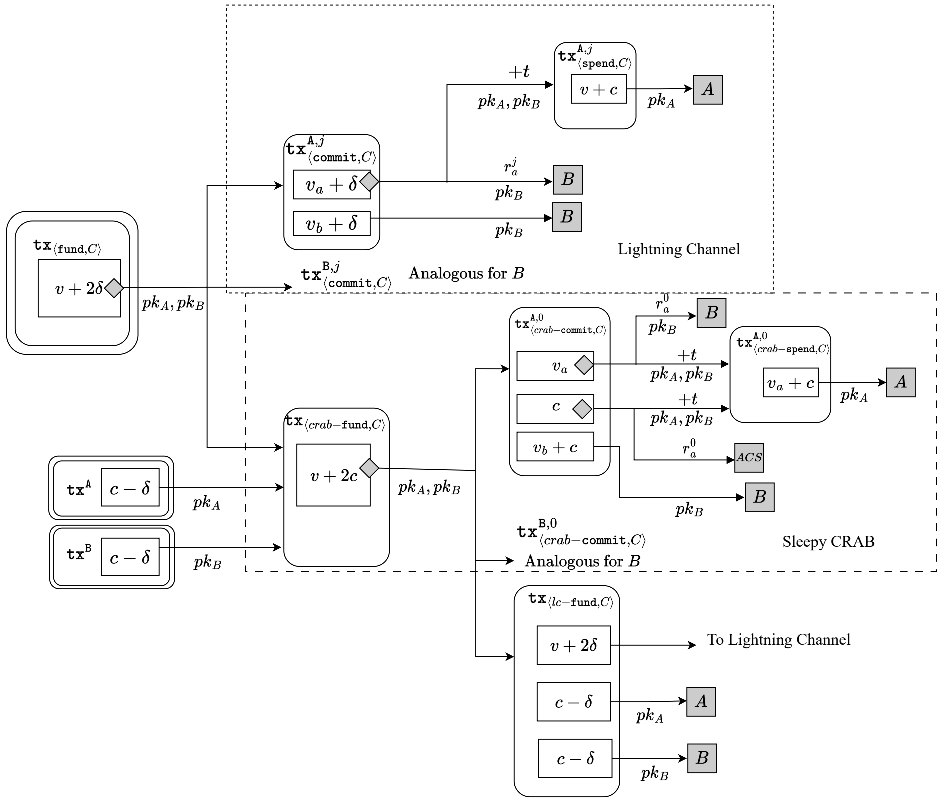

Figure 13 Transaction scheme for Splicing

We choose the collateral $c = v + \epsilon$, i.e., slightly higher than v to get the intended protocol execution. If the collateral c was equal to v coins, then A could offer $c = v$ coins to miners for mining the old state and keep v coins. There is a non-zero probability with which M

might choose the old state, and B ends up getting a payoff of 0.

◀

From Theorem 18, we derive the desired property for `Sleepy CRAB` under rational participants.

▶ **Corollary 19.** *Assuming rational parties and miners, with one participant remaining offline, balance security is satisfied in* `Sleepy CRAB`*.*

H Interplay of `Sleepy CRAB` with `LC`

`Sleepy CRAB` can be used alongside Lightning channels in an agile way. Users can use Lightning channels, until they wish to go offline, at which point they simply change to `Sleepy CRAB`, using a technique known as *splicing* [39]. Splicing allows users to increase or decrease the channel capacity with an on-chain transaction, which can be thought of as closing the old and simultaneously opening a new channel, with a different capacity. Indeed, we can use this technique to change the nature of the channel to `Sleepy CRAB`, by adding the necessary collateral and logic (or else change it back to Lightning).

We illustrate splicing in Figure 13. The funding transaction $\mathtt{tx}_{\langle \mathtt{fund}, C\rangle}$ is used for opening a `LC`, where A has a balance $v+\delta$ coins and B has a balance δ coins. A and B continue performing off-chain payments using this lightning channel C. A and B update C to the j^{th} state update, where A has a balance $v_a+\delta$ and B has a balance $v_b+\delta$. If one of the participants wants to go offline, he or she informs the other channel participant. A and B mutually agrees to open a `Sleepy CRAB`, where $\mathtt{tx}_{\langle \mathtt{fund}, C\rangle}$ is used to fund the funding transaction of `Sleepy CRAB`. Additional input of $c-\delta$ coins each would be required for the collateral from both A and B respectively. The funding transaction $\mathtt{tx}_{\langle crab-\mathtt{fund}, C\rangle}$ is used to open the `Sleepy CRAB` C, where A has balance v_a+c coins and B has a balance v_b+c coins. Once $\mathtt{tx}_{\langle crab-\mathtt{fund}, C\rangle}$ is posted on-chain, the lightning channel ceases to exist. Neither A can post $\mathtt{tx}^{\mathtt{A},j}_{\langle \mathtt{commit}, C\rangle}$ nor B can post $\mathtt{tx}^{\mathtt{B},j}_{\langle \mathtt{commit}, C\rangle}$ on-chain.

A and B continue using the `Sleepy CRAB`, and B goes offline for a certain period of time, after the k^{th} channel update. Let balance of A and B be v'_a+c and v'_b+c. He has to post the secret r_b^{k-1} on-chain before going offline. If A misbehaves when B is offline, miners will punish A. Once B becomes active, he can request A to close the `Sleepy CRAB` and switch back to `LC` by withdrawing the collateral c. In the Figure 13, we show a third arrow going out of $\mathtt{tx}_{\langle crab-\mathtt{fund}, C\rangle}$. It shows that the output of $\mathtt{tx}_{\langle crab-\mathtt{fund}, C\rangle}$ serves as the input of the funding transaction $\mathtt{tx}_{\langle lc-\mathtt{fund}, C\rangle}$ for the new `LC` between A and B. Only $v+2\delta$ coins are used for funding the channel, rest $2c-2\delta$ coins are divided equally between A and B. The initial commitment transaction of this new channel will have output distributed as per the k^{th} state of `Sleepy CRAB` C.