Zeus: Defending against Fee Stealing and Griefing Attacks in Multi-Hop Payments

Jingyu Liu[1], Yingjie Xue[2], Di Wu[3], Jian Liu[3], and Xuechao Wang[1]

[1]The Hong Kong University of Science and Technology (Guangzhou)
[2]University of Science and Technology of China
[3]Zhejiang University
[1] *jliu514@connect.hkust-gz.edu.cn, yjxue@ustc.edu.cn, {wu.di, liujian2411}@zju.edu.cn, xuechaowang@hkust-gz.edu.cn*

Abstract

Payment Channel Networks (PCNs) are the most scalable and trust-minimized solution to Bitcoin's scalability challenges. Within PCNs, connected payer and payee can make arbitrary off-chain transactions through multi-hop payments (MHPs), while intermediate relays charge relay fees by providing liquidity. However, current MHPs face critical security threats including fee-stealing attacks and griefing attacks. In this paper, we first identify new fee-stealing attacks targeting most existing MHP protocols. Then we demonstrate the impossibility of griefing resistance by reducing the griefing resistance MHP to a fair secret exchange problem. Finally, we introduce *Zeus*, the first Bitcoin-compatible MHP that is secure against fee-stealing attacks and offers bounded griefing protection against k-cost-sensitive adversaries—those who only launch griefing attacks when the damage exceeds a k fraction of their cost. These guarantees are established through rigorous proofs in the Global Universal Composability (GUC) framework. Our comprehensive evaluation demonstrates that *Zeus* reduces worst-case griefing damage to 28% and 75% compared to MHP schemes such as AMHL (NDSS'19) and Blitz (USENIX SEC'21), respectively. Our results further show that, even under the most adverse configurations in the Lightning Network, *Zeus* ensures attackers incur costs at least 10x higher than potential damage.

1 Introduction

Payment channel networks (PCNs), exemplified by Lightning Network (LN), are one of the most promising solutions to solve the scalability problem of Bitcoin. Compared to other solutions, such as sidechains, rollups, PCNs stand out by not introducing additional security assumptions [43] (e.g., reliance on external committees), rendering them the trust-minimized and widely-adopted solutions for scaling Bitcoin.

In PCNs, transactions can be conducted off-chain. Given a payment channel, two users can securely conduct numerous payments, while only burdening the blockchain with two transactions (opening and closing) [42]. Creating direct channels with every potential user is inefficient due to the need to lock BTC on mainnet. Consequently, multi-hop payment (MHP) protocols emerge [5, 32, 33, 42], enabling payments between indirect payer and payee through intermediate relays. Most MHPs adopt a lock-resolve paradigm to ensure atomic channel updates [5, 32, 33, 35, 42]. For ease of exposition, we use the most widely adopted Hashed Timelock Contract (HTLC) based MHP[1] to show how to realize MHPs. Other solutions share same basic structures as HTLC-based MHP.

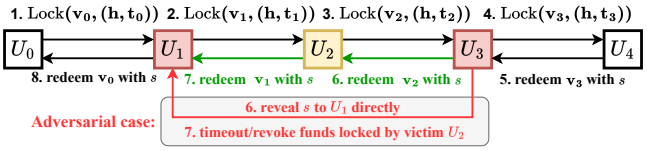


Figure 1: HTLC-based MHP protocol and wormhole attack.

As illustrated in Fig. 1, HTLC-based MHP follows a *lock-resolve paradigm*: in the *lock phase*, starting from the payer U_0, each left user U_i makes a conditional payment to its right neighbor U_{i+1}, with the amount $v_i = x + \sum_{j=i+1}^{n} f_j$ through the intermediate channel γ_i, where x is the payment amount and f_j is the relay fee towards relay U_j. The unlock condition of U_i's payment is: once the preimage of a hash h (denoted by s) is revealed before t_i[2], the locked fund is given to U_{i+1}; otherwise, it is refunded to U_i. Once all funds are locked, payee U_{n+1} enters the *resolve phase*, gives the preimage s to U_n and asks U_n to update channel γ_n, where the new state directly gives the locked fund v_n to U_{n+1}. If U_n refuses, U_{n+1} can always close channel and redeem v_n on-chain. Subsequently, U_{i+1} redeems the payment from U_i, until U_0's payment gets redeemed.

[1]It supports LN with $\sim$ 42M USD TVL as of August 2025.

[2]$t_0 = 4\delta$, and $t_{i+1} = t_i - \delta$, δ is number of blocks a node has to settle a payment before it could potentially lose money. In LN, δ is at least 18 Bitcoin blocks [1]. For comparison, the block confirmation time Δ is 6 blocks. The Bitcoin block time is 10 minutes.

1.1 Challenges

The lock-resolve paradigm introduces vulnerabilities for achieving secure and efficient multi-hop payment over payment channel networks, as we demonstrate below.

Fee Stealing Attack. MHPs face *fee-stealing attacks* where the payee receives payments while relays go unpaid, exemplified by the wormhole attack [33]. Incremental-lock based protocols (Fulgor [32], AMHL [33], EAMHL+ [54], Astrape [13]) enforce atomicity via a trusted payer setup (secrets generation), assuming payer honesty [33, 45]. However, we observe that the payer can profitably deviate (detailed in Section 3), violating incentive compatibility. These protocols fail to ensure that rational payers behave honestly.

Griefing Attack. Griefing attacks in MHPs allow a malicious payer to initiate doomed payments (including loop attacks [48]), locking relay funds until timeout expiration [36, 40, 46]. These attacks can isolate critical nodes and disrupt network operations. Existing defenses—path restrictions [46], rerouting [15], and reputation/fee systems [26, 50]—remain insufficient. As discussed in Section 1.3, these solutions are either hindered by the permissionless natural of PCNs, or introduce unfairness for honest participants. Moreover, in most MHPs, such as the HTLC-based MHP [42], Fulgor [32], and AMHL [33], the timeout set for funds locked at each channel γ_i must be progressively longer than that of the subsequent relay U_{i+1} to guarantee the balance security [33]. For example, in Fig.1, the timeout t_i increases by one δ at each hop from right to left, where δ is a safe timelock gap ($\geq$ 18 Bitcoin blocks in LN [1]). Given a n-hop payment with value x, an adversary can inflict a disproportionately larger *griefing damage*—defined as the total fund times their locking time for innocent intermediaries—amounting to $\Theta(n^2 \cdot \delta \cdot x)$. Based on the current LN topology, griefing a single 200 USD MHP can equate to immobilizing 11,878 USD-hours of liquidity for relays. Payment rejection by the payee enables an infinite griefing factor (damage-to-cost ratio [19]) at zero cost.

Recent works (Sprites [35], AMCU [14], Blitz [5]) use global on-chain events to enforce payment atomicity, achieving constant timeout per hop, reducing the griefing damage to $\Theta(n \cdot \delta \cdot x)$. Blitz further forces malicious payers to post on-chain transactions, raising attack cost. However, it fails to prevent the zero-cost griefing launched by the payee and remains vulnerable to fee-stealing attacks (Section 3).

Research Question. *Is it possible to design a Bitcoin-compatible MHP protocol that simultaneously achieves both fee-stealing resistance and griefing resistance?*

1.2 Contributions

Identification of Fee Stealing Vulnerabilities within Existing Bitcoin-Compatible MHPs. We identify that, for incremental-lock based MHPs (e.g., Fulgor [32], AMHL [33], and Astrape [13]) requiring a payer-side trusted setup, it is not incentive compatible for payers to stay honest after the setup phase. The key insight is that payers can engage in the illicit sale of setup secrets to malicious relays, a collusive action which increase the payoff for both the payer and the complicit malicious relay. For Blitz, we discover that colluding relays can steal relay fees by exploiting its fast revoke mechanism (see Section 3). Our empirical analysis in Section 8 underscores the acute vulnerability to fee-stealing attacks. Remarkably, it is shown that the engagement of less than 0.6% of total relays in fee-stealing activities allows an adversary to amass roughly 50% of all distributed relay fees.

Impossibility of Griefing Resistant MHPs. We prove that no lock-resolve MHP with multiple relays can achieve *griefing resistance* against malicious adversaries. Our proof reduces the griefing resistant MHPs to fair secret exchanges (FSE). We then prove that solving FSE with public blockchain is "not easier" than solving FSE in a world with solely a random beacon [44]. In the random beacon world, all existing FSE protocols [31, 44] have a communication complexity of $O(1/p)$ to ensure fairness with probability $\geq 1-p$. This result provides theoretical insights with direct implications for scenarios like atomic swaps and Multi-party Secure Computation, which may be of independent interest.

***Zeus*: A Secure MHP Protocol.** We propose *Zeus*, the first Bitcoin-compatible MHP simultaneously achieves:

1. Fee-Stealing Resistance. We identify that fast-revoke (design to improve the efficiency) is the root of the fee stealing risks. To prevent this, we introduce a secure *refundable deposit* that the payer locks before the payment begins, maintaining the same level of efficiency as Blitz.

2. Griefing Resistance against k-Cost-Sensitive Adversaries. Zeus defends against griefing by (i) lowering the griefing damage via constant timelocks for all channel funds (payment-length agnostic) and (ii) increasing the griefing cost for any adversary. For (ii), we first disincentivize payee-side griefing by letting the payee post a small *one-time on-chain stake* (a few USD). This stake is then effectively augmented by off-chain *premiums* pledged by the payee, scaled to the potential griefing damage, thereby increasing the payee's griefing cost. Second, to prevent payer-side griefing, the payer will place a small refundable off-chain deposit to the payee (routed by relays). Should the payer attempt to grief, this deposit is forfeited to compensate the payee and relays. Last, even in scenarios involving colluding payer and payee, griefing the MHP necessitates at least two distinct on-chain transactions. Our evaluation demonstrates that, even in the worst-case setting, *Zeus* maintains a griefing factor below 0.1. For typical payment scenarios (e.g., payment $\leq$ \$500), the griefing factor drops significantly to below $0.01 \ll 1$.

Formal Security Analysis and Evaluation. We provide a comprehensive security analysis of *Zeus* using the global universal composability (GUC) framework [11]. We then implement *Zeus* with Bitcoinjs-lib [2] and evaluate it using real LN topology captured in March 2025. Our evaluation

shows that *Zeus* achieves a griefing factor lower than 0.1 for any adversaries. For griefing damage, compared to the state-of-the-art solution Blitz, *Zeus* reduces griefing damage by 25%, and achieves a 72% reduction relative to widely adopted HTLC and AMHL. *Zeus* further decreases dispute resolution cost by 10.3% compared to HTLC and AMHL.

1.3 Related Works

MHPs. Existing MHP protocols have made significant strides, yet each faces distinct limitations, as summarized in Table 1. Beginning with Fulgor [32], several protocols (e.g., AMHL [33], EAMHL+ [54], and Astrape [13]) have emerged to address the *wormhole attack*. These protocols, which we term *incremental-lock* based MHPs, are widely considered in industry as the replacement to vulnerable HTLC-based approaches. They employ different cryptographic primitives (e.g., zero knowledge proofs, homomorphic one-way functions) to ensure *strong atomicity* - guaranteeing that U_i's payment can be redeemed only if all preceding sub-payments have been redeemed. Rain et al. [45] validate that these MHPs can effectively prevent the wormhole attack through game-theoretic analysis. However, these protocols rely on trusted setup performed by payers, and require payers to stay honest during the following execution. Moreover, these MHPs inherit the *linear collateral* inefficiency and griefing risks from HTLC. Linear collateral means the timelock along the MHP increases along the payment, thereby leading to a griefing damage with quadratic complexity w.r.t. the path length. To reduce the collateral, Sprites [35] pioneered the use of global state through an on-chain preimage manager. While innovative, this approach requires smart contract beyond Bitcoin's capabilities. Subsequent efforts to adapt these concepts to Bitcoin's UTXO model include AMCU [14], which was later proven insecure by Payment Tree [24]. Although Payment Tree achieved logarithmic collateral, it remains vulnerable to the *Domino attack* [6], where a party can force close all involved channels on-chain. Blitz [5] first securely achieves constant-collateral but only partially mitigates griefing attacks. Blitz introduced the "pay-or-revoke" paradigm, which has been widely adopted in protocols like Thora [3], Donner [6] and X-Transfer [29]. Section 3 will demonstrate that Blitz remains vulnerable to fee stealing.

Protocols	Fee Security	Collateral	Griefing Factor	Script
HTLC-based [42]	✗	Linear	∞	BTC
Sprites [35]	✗	Constant	∞	SC
Fulgor [32], AMHL [33]	honest payer	Linear	∞	BTC
Payment Tree [24]	honest payer & payee	Logarithmic	∞	BTC
HTLC-GP [34]	honest payee	Linear	∞	SC
Blitz [5]	✗	Constant	∞	BTC
Zeus (this work)	✓	Constant	$\leq 0.1^*$	BTC

Table 1: Comparison of MHP protocols. BTC: Bitcoin script support, SC: Require Smart Contract. For typical payments ranging from \$10 to \$500, griefing factors drop to below $0.01 \ll 1$.

Griefing Mitigation. To mitigate griefing in MHP, several works [15, 26, 46, 50] introduce different mitigations. Path restrictions [46] and reputation systems [50] are fundamentally limited by the permissionless nature of PCNs, making enforcement and reliable identity tracking infeasible. Fee-based [15, 26, 34] approaches either bring unfairness or require additional assumptions. For instance, Kumble et al. [26] require endpoints to pre-pay non-refundable fees regardless of payment success, discouraging honest participation. Rerouting [15] shifts additional cost to victim relays, exacerbating fairness issues and failing to deter determined adversaries. HTLC-GP [34] requires an "honest payee" setup and utilized smart contract capabilities to achieve dynamic fee rates.

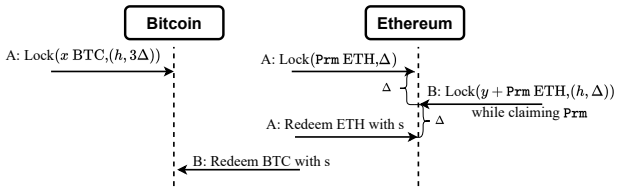


Figure 2: Atomic swap preventing Alice from griefing Bob.

In cross-chain swaps [23], premiums are a common technique to mitigate griefing attacks [20, 37, 51, 52]. Consider a two-party swap where Alice wants to exchange her x BTC for Bob's y ETH. To disincentivize Alice from aborting mid-way, as depicted in Fig. 2, Alice needs to pay premium for the liquidity locked by Bob. Specifically, Alice first locks x BTC in an HTLC on Bitcoin network and also locks a *premium* Prm on Ethereum. 2) Bob can only claim this premium by locking $y + \texttt{Prm}$ ETH in a pre-agreed HTLC. After premium redemption, if Alice tries to grief this swap, Alice will lost its premium permanently. Based this idea, Xue and Herlihy [52] showed this griefing prevention can extended to both swap participants, and the griefing damage can be reduced to a negligible level through multiple rounds of premiums. However, this approach (i) requires linear increasing timelocks to handle premium dispute; and (ii) is infeasible for MHPs due to the dust limit—the minimum transferable value in Bitcoin-compatible PCNs. Grief-free Swap [37] claims to deterministically eliminate griefing risk by binding the premium and principal locking within a single transaction. Nevertheless, one participant can refuse to sign the transaction, thereby implicitly griefing the principal reserved by its counter-party.

2 Background and Model

Digital Signature. A digital signature scheme is a tuple $\Sigma := (\text{KeyGen}, \text{Sign}, \text{Vrfy})$ satisfying EUF-CMA security [10]. We use (sk_i, pk_i) to denote the key-pair of a user U_i, and $Sig(U_i)$ to denote the signature of U_i.

UTXO Ledger and Payment Channel Networks. Bitcoin is an *Unspent Transaction Outputs* (UTXO) ledger, modeled by an ideal functionality $\mathcal{G}_{\text{L}}$ [4]. For a transaction

$tx := (\text{id}, \text{Ins}, \text{ScriptSigs}, \text{Outs})$, tx.id is the unique identifier, tx.Ins $:= (\mu_1, \ldots, \mu_n)$ is the list of inputs, where each μ_i contains a reference to a UTXO, identified by the source transaction's id μ_i.id and an index μ_i.idx. tx.ScriptSigs $:= (\lambda_1, \ldots, \lambda_n)$ is the list of scripts that unlock inputs, and tx.Outs $:= (\theta_1, \ldots, \theta_m)$ is the list of outputs. For each output θ_i, θ_i.amt is the amount of coins, and $\theta_i.\phi$ is the locking script of this output. $\mathcal{G}_{\text{L}}$ maintains a block list, with each block b containing a list of transactions b.TXs. A valid transaction tx submitted to $\mathcal{G}_{\text{L}}$ at block height h is considered confirmed after Δ consecutive blocks are mined (i.e. added to block $h+1$ when block $h+\Delta$ is mined).

A PCN can be modeled as an ideal functionality $\mathcal{F}_{Channels}$ [4], that supports channel opening, updating, and closing. Each channel γ has a unique identifier γ.id, left and right users ($\gamma.lu$, $\gamma.ru$), a current state γ.state that contains a list of outputs $\theta_1, \ldots, \theta_n$. We highlight following interfaces: 1) UPDATE(γ.id, tx_{state}): Update channel γ to the new state tx_{state}, co-signed by $\gamma.lu$ and $\gamma.ru$. $\mathcal{F}_{Channels}$ confirms within t_{update} blocks. 2) CLOSE(γ.id): Close γ when called by a user $\in \gamma.lu/ru$. A transaction containing the latest state of γ will be finalize in $\mathcal{G}_{\text{L}}$ within t_{close} blocks. The formal definitions of $\mathcal{F}_{Channels}$ and $\mathcal{G}_{\text{L}}$ can be found in Appendix C

Communication Model. We assume 1) a round-based synchronous network, where time is modeled by a public global clock functionality [25]. 2) authenticated, private communication channels [10] with bounded one-round delay. $(m) \overset{\tau}{\hookleftarrow} \mathcal{Q}$ denotes a message m sent from $\mathcal{Q}$ is received at round τ.

MHP Model. Let a n-hop MHP be defined as a tuple $\mathcal{M} = (U, \Gamma, x, \text{Fees}, \mathcal{S})$ where $U = (U_0, U_1, ..., U_n, U_{n+1})$ is a sequence of $n+2$ parties with payer U_0, payee U_{n+1}, and relays U_1 through U_n. $\Gamma = (\gamma_0, ..., \gamma_n)$ denotes the payment channels, where U_i establishes channel γ_i with U_{i+1}. Payer U_0 makes a payment with amount x to the payee U_{n+1}. We denote the payment x as *principal* in the following context. Fees $= (f_1, ..., f_n)$ represents the sequence of relay fees where f_i is the fee charged by the relay U_i. We denote the cost of the transaction on the chain as F. For each γ_i, $\mathcal{S}_i \in \mathcal{S}$, $\mathcal{S}_i$ defines all the possible states derived from an initial `idle` state. For simplicity, we assume every channel γ_i have the same `idle` state, with U_i owning lb and U_{i+1} owning rb. Each state specifies a list of token distributions mapped with spending conditions (signatures or specific constraints). Here we define *Lock-Resolve MHPs*, capturing the essence of most MHPs.

Definition 1 (Lock-Resolve MHPs)**.** *A Lock-Resolve MHP, Π_{MHP}, satisfies (i) $\forall \gamma_{i\in[0,n]}$ has following possible states;*

$$\texttt{locked}: (\{lb - v_i : U_i\}, \{rb : U_{i+1}\}, \{v_i : \text{cond}_i\}),$$
$$\texttt{paid}: (\{lb - v_i : U_i\}, \{rb + v_i : U_{i+1}\}),$$
$$\texttt{unpaid}: (\{lb : U_i\}, \{rb : U_{i+1}\}),$$

and (ii) executes in the following two phases:

- *Lock Phase: For each $\gamma_{i\in[0,n]}$, U_i needs to commit the liquidity used in this payment by sending a lock signature m^i_{lock} (i.e., U_i's signature over the `locked` state) to U_{i+1}, updating γ_i to `locked` state. γ_i can be updated to locked state ($\gamma_i \rightarrow$ `locked`) if only if m^i_{lock} is received.*
- *Resolve Phase: Before a predefined deadline N, participants invoke the resolution process, updating each channel γ_i to `paid` state if the spending condition cond_i met. Otherwise, reset γ_i to `unpaid`. γ_i can be updated to `paid` only if γ_i has enter the `locked` state.*

Here we define the well-established security requirements for MHP protocols [5, 16, 32, 33]. Our definition is directly derived from [16]. Let Π_{MHP} be a protocol for $\mathcal{M}$ and $\mathcal{A}$ be a probabilistic polynomial time (PPT) adversary controlling all but honest parties. Any Π_{MHP} must satisfy:

Definition 2 (Balance Security)**.** *When Π_{MHP} terminates: 1) the balance of any honest relay $U_{i\in[1,n]}$ will not decrease compared with its initial balance, and 2) if payer U_0 and payee U_{n+1} are controlled by the same (rational) entity, U_0 pays U_1 in the first channel γ_0 only if U_{n+1} gets paid in channel γ_n.*

Definition 3 (Correctness)**.** *If all channels have sufficient balances and the parties are honest, then the payment will be settled off-chain, and each relay U_i gets the relay fee f_i.*

Definition 4 (Coin Availability)**.** *For any honest $U_{i\in[0,n+1]}$, its funds must be resolved before the deadline N. Formally, if $\gamma_i \rightarrow$ `locked`, then γ_i can be resolved to `paid` or `unpaid` within a finite time N. If γ_i does not enter `locked` state, then the MHP protocol must terminate within a finite time N.*

Fee Security. These properties fail to regulate sophisticated fee stealing attacks or the griefing attacks. So we first strengthen MHP security by formally introducing *fee security*. This ensures that once a payment is settled off-chain, all honest relays are guaranteed to receive their fees.

Definition 5 (Fee Security)**.** *Given any honest $U_{i\in[1,n]}$, if the payee U_{n+1} receives the payment off-chain, then U_i must receive the relay fee f_i if it has forwarded this payment.*

Griefing Resistance. Compared with fee stealing, griefing attacks are usually performed by endpoints, and the attackers do not care whether payments are successful or not. We formalize griefing resistance in two different settings: 1) *griefing resistance against malicious adversaries* (strong griefing resistance): Guarantees honest relays cannot be grieved against any adversary controlling any set of participants:

Definition 6 (Griefing resistance against malicious adversaries)**.** *For any honest $U_{i\in[1,n]}$ and PPT adversary $\mathcal{A}$ controlling any other parties, if U_i locks funds, then U_i gets the corresponding relay fee f_i.*

2) *Griefing resistance against cost-sensitive adversaries*: In a more realistic setting, griefing attacks are initiated by cost-sensitive endpoints, who can further corrupt relays to

increase the griefing damage. We denote the possible griefing coalition as $\mathcal{S}_A$, which includes at least one endpoint (payer or payee) plus some relays. Formally, $\mathcal{S}_A := \mathcal{A}_{\text{endpoints}} \cup \mathcal{A}_{\text{relays}}$, where $\emptyset \neq \mathcal{A}_{\text{endpoints}} \subseteq \{U_0, U_{n+1}\}$ and $\mathcal{A}_{\text{relays}} \subseteq \{U_1, \ldots, U_n\}$. We define the *grieved relay set* as S_G. A relay $U_r \in S_G$ if U_r locks funds but does not get the corresponding relay fee.

For each $U_t \in S_G$, we denotes its *griefing damage* as GD_t. GD_t quantifies the liquidity loss from U_t's locked collateral. Formally, $GD_t := r \cdot (fl_{t-1} \cdot tl_{t-1} + fl_t \cdot tl_t)$, where r is the effective interest rate for the locked funds (e.g., opportunity cost of capital), fl_{t-1} and fl_t are the fund amounts locked by U_t in its adjacent channels γ_{t-1} and γ_t respectively, and fl_{t-1} and TL_t are the corresponding lock durations. Then the griefing factor λ is defined as $\lambda := (\sum_{U_t \in S_G} GD_t)/c$, where c is the griefing cost incurred by the adversary $\mathcal{A}$. Here, we ensure griefing attacks are *economically irrational* by capping the griefing factor λ lower than k, where $k < 1$.

Definition 7 (Griefing resistance against k-cost-sensitive adversaries)**.** *For a PPT adversary $\mathcal{A}$ controlling any $\mathcal{S}_A$ and $\forall k \in (0,1)$, the griefing factor $\lambda \leq k$.*

Pay-or-Revoke Paradigm. The "pay-or-revoke" paradigm, introduced in Blitz [5], enables atomic synchronization of otherwise independent UTXOs. This is achieved by making their settlement contingent on a single global on-chain event. Consider the example in Fig. 3, where two separate UTXOs, $\mathbf{Out}_A$ (controlled by Alice and Bob) and $\mathbf{Out}_B$ (controlled by Bob and Caro), are time-locked. After a timeout T, Bob can claim $\mathbf{Out}_A$ and Caro can claim $\mathbf{Out}_B$. To allow Alice and Caro to atomically revoke their funds before this timeout, a global transaction, tx_G, is prepared.

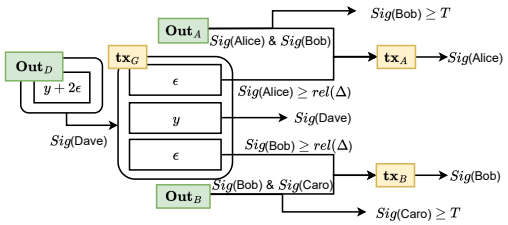


Figure 3: The pay-or-revoke paradigm.

This tx_G creates two small "trigger" outputs. Alice's revocation transaction, tx_A, is constructed to require one of these triggers as an input to reclaim her funds from $\mathbf{Out}_A$. Similarly, Caro's revocation transaction, tx_C, requires the other trigger to reclaim her funds from $\mathbf{Out}_B$. This dependency ensures that neither revocation can occur until tx_G is confirmed on-chain. Furthermore, the triggers in tx_G are encumbered with a relative timelock Δ, preventing one party from front-running.

3 Exploiting Bitcoin-Compatible MHPs

Incremental-Lock Based MHPs. As instanced by AMHL [33], incremental-lock based MHPs prevent wormhole attack [33] via a trusted setup: the payer generates secrets $s_0, \ldots, s_n$ (relay $U_{i \in [1,n]}$ receives s_i, the payee U_{n+1} gets $\bar{s}_n = \sum_{j=0}^{n} s_j$) and uses a homomorphic one-way function g [27] with $g(\sum s_j) = \prod g(s_j)$. Compared to HTLC, each channel γ_i replaces hashlock with $h_i = \prod_{j=0}^{i} g(s_j)$, requiring $\bar{s}_i = \sum_{j=0}^{i} s_j$ for redemption. As long as the payer keeps the secrets private, U_i can redeem the payment from U_{i-1} only if all subsequent $\{U_{i+1}, \ldots, U_{n+1}\}$ does, enforcing strong atomicity by induction (Fig. 4).

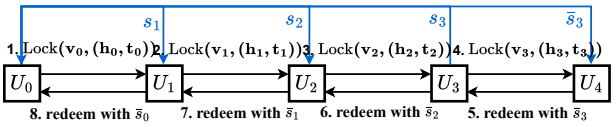


Figure 4: Workflow of AMHL.

Exploiting Incremental-Lock Based MHPs. We start with a simple observation: once U_0 confirms that the last relay U_n has locked funds in the channel γ_n, payee U_{n+1} can always redeem the payment by sending $\bar{s}_n$ to U_n in time. Since U_0 possesses all setup secrets necessary to unlock any locked fund, this creates perverse incentives: U_0 *can minimize costs by selling these secrets to malicious relays while still ensuring payment settlement.* Such *secret resale attack* works as follow:

Once U_n confirms $\gamma_n \rightarrow$ `locked`, U_0 can publish the identity of the payment (e.g., lock conditions) on various off-chain communication channels to orchestrate fee theft. Malicious relays watches those off-chain communication channel. Upon recognition, malicious relays can negotiate secret acquisition through secure exchange protocols. A simple HTLC can settle this acquisition, ensuring fairness for both party. Once the adversaries (e.g., U_0 and U_3 in Fig. 4) obtain the secrets $\{s_0, \ldots, s_n\}$, they can construct the unlock key $\bar{s}_1 = s_1$ for channel γ_0. Therefore, similar to the wormhole attack in HTLC (Fig. 1), U_3 can timeout honest U_2, while capturing its intended relay fees directly from γ_0.

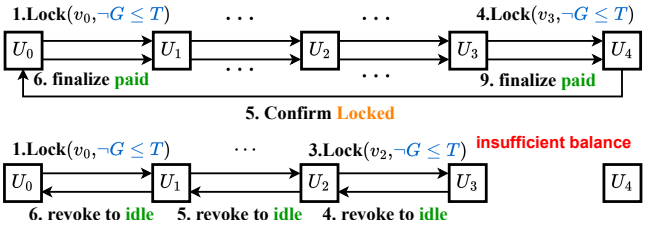


Figure 5: Blitz protocol workflow showing ideal execution (top) and fast revocation path (bottom).

Blitz. To achieve constant collateral, Blitz introduces a "pay-or-revoke" mechanism where locked payments automatically default to the *paid* state after a deadline T, unless the payee publishes a revocation transaction G on-chain before T. Since this requires costly on-chain revocations for failed payments — which are frequent due to PCN anonymity constraints and volatile channel balances — Blitz introduces *fast revoke* for

efficient off-chain cancellation. Blitz enables two execution paths when participants are honest:

1) Fast Confirmation: After the payment locked, once U_{n+1} confirms the payment is locked (step 5, top part of Fig. 5), from the payer to the payee, honest participants settle all channels to the `paid` state (step 6 to step 9).

2) Fast Revoke: During the lock phase, if a relay U_i finds it does not have sufficient balance to continue, U_i quickly revokes the payment with its previous U_{i-1} (step 4, bottom part of Fig. 5). If U_{i-1} receives this revocation within a predefined revoke-period, U_{i-1} accepts and continues the Fast Revoke process, canceling the MHP off-chain.

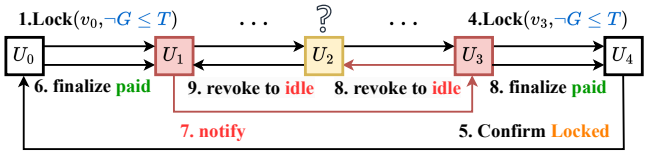


Figure 6: Fee stealing attack over Blitz.

Exploiting Blitz. However, the fast-revoke introduces protocol ambiguity where honest relays cannot distinguish between legitimate liquidity failures and adversarial revocations. This creates an opportunity for colluding parties to simulate revocation and effectively steal fees from honest participants. Consider the example depicted in Fig. 6, which demonstrates how colluding relays (U_1, U_3) exploit the fast-revoke ambiguity in Blitz. The attack proceeds as follows:

1) After all channels are locked, honest U_0 initiates the fast confirmation (step 6), settling the payment directly off-chain.

2) Instead of forwarding the confirmation to U_2, the adversarial U_1 covertly notifies its partner U_3. Once notified, adversarial U_3 initiates a false revocation with U_2 while simultaneously finalizing the payment with U_4 (step 8).

3) If this attack is executed within the revoke period, honest U_2 cannot distinguish this from a legitimate revocation and will accept it, resulting in U_2 being free-ridden.

Consequently, the colluding relays (U_1 and U_3) capture the fee f_2 intended for the honest relay U_2. A naive solution would be to remove the fast-revoke mechanism. However, this would significantly burden the payer, who would then be required to publish an on-chain revocation transaction G whenever a payment fails due to a relay's insufficient balance.

4 Impossibilities of Griefing Resistant MHPs

Before diving into the theoretical results, we first provide theoretical background on fair exchange.

Fair Secret Exchange. A fair secret exchange (FSE) protocol enables two mutually distrusting parties $\mathcal{P}$ and Q to exchange their secret items $(i_{\mathcal{P}}, i_Q)$ like signatures with descriptions $(\text{desc}_{\mathcal{P}}, \text{desc}_Q)$ [39, 53].

Strong/p-partial Fairness: We say a FSE achieves *strong fairness* if honest $\mathcal{P}(Q)$ sends $i_{\mathcal{P}}(i_Q)$, only if $\mathcal{P}(Q)$ obtains i_Q $(i_{\mathcal{P}})$ such that $\text{desc}_Q(i_Q) = 1$ $(\text{desc}_P(i_{\mathcal{P}}) = 1)$. We say that a probabilistic FSE π_{FSE} achieves *p-partial fairness* if the probability of an unfair run is at most p.

T-Timeliness: Honest party terminates by round T.

q-Effectiveness: If both parties are honest, they successfully exchange items $i_{\mathcal{P}}$ and i_Q matching their descriptions upon protocol completion. This successful exchange occurs with a probability of at least q. The notion of q-effectiveness accommodates the probabilistic nature of some FSE protocols, such as the one proposed by Rabin [44].

4.1 Reducing MHPs to FSE.

How to link griefing resistant MHPs to the fair secret exchange problems? We take a two-relay MHP as an example, and demonstrate our reduction through two steps.

Lemma 1 (Griefing resistance and Balance Security imply simultaneously locking)**.** *Consider a two-relay lock-resolve MHP $(U_0 \to U_1 \to U_2 \to U_3)$ that satisfies balance security and strong griefing resistance. Given (U_0, U_3) controlled by the same entity, let $\mathcal{E}_1$ be the event that U_1 sends m^1_{lock} to U_2, and $\mathcal{E}_2$ be the event that U_2 sends m^2_{lock} to U_3. Then 1) if U_1 is honest, $\mathcal{E}_1 \implies \mathcal{E}_2$; and 2) if U_2 is honest, $\mathcal{E}_2 \implies \mathcal{E}_1$.*

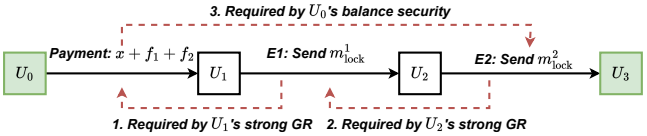


Figure 7: The security guarantee of a two-relay MHP.

Proof. We prove each direction separately, with logical implication highlighted in Fig. 7.

Part 1: $(\mathcal{E}_1 \implies \mathcal{E}_2)$ given honest U_1: (i) Since U_1 is honest and sends m^1_{lock}, this action causes channel γ_1 (between U_1 and U_2) to enter the `locked` state. By the definition of *Strong Griefing Resistance* for the honest relay U_1, the event $\gamma_1 \to$ `locked` guarantees that its preceding channel γ_0 must transition to the `paid` state, i.e., $\gamma_0 \to$ `paid`, as the relay fee is paid solely over γ_0. (ii) Given collaborative (U_0, U_3), by *Balance Security*, if $\gamma_0 \to$ `paid`, then it must be that the final channel γ_2 (between U_2 and U_3) also transitions to `paid`, i.e., $\gamma_2 \to$ `paid`. (iii) By the definition of lock-resolve MHP, for the final channel $\gamma_2 \to$ `paid` is only possible if U_2 has previously locked the funds for U_3 by sending m^2_{lock}. Therefore, $\gamma_2 \to$ `paid` $\implies \mathcal{E}_2$. Combining these steps, we have: $\mathcal{E}_1 \Rightarrow (\gamma_1 \to \texttt{locked}) \overset{(i)}{\Longrightarrow} (\gamma_0 \to \texttt{paid}) \overset{(ii)}{\Longrightarrow} (\gamma_2 \to \texttt{paid}) \overset{(iii)}{\Longrightarrow} \mathcal{E}_2$.

Part 2: $(\mathcal{E}_2 \implies \mathcal{E}_1)$ given honest U_2: (i) By *Strong Griefing Resistance* for the honest relay U_2, if U_2 sends m^2_{lock} (causing $\gamma_2 \to$ `locked`), it is guaranteed that its incoming channel γ_1 will transition to `paid` ($\gamma_1 \to$ `paid`). (ii) Definition of lock-resolve MHP implies that for $\gamma_1 \to$ `paid` $\implies \gamma_1 \to$ `locked`

and given honest U_0, $\gamma_1 \to \texttt{locked} \implies \mathcal{E}_1$. Therefore: $\mathcal{E}_2 \overset{(i)}{\implies} (\gamma_1 \to \texttt{paid}) \overset{(ii)}{\implies} (\gamma_1 \to \texttt{locked}) \overset{(ii)}{\implies} \mathcal{E}_1$. □

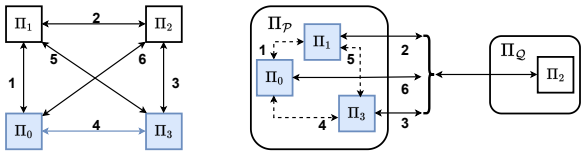


Figure 8: Construct FSE from a griefing resistant MHP. Numbers indicate how communication channels are mapped.

Lemma 2. *Consider a two-relay lock-resolve MHP ($U_0 \to U_1 \to U_2 \to U_3$) that satisfies balance security, correctness, coin availability with deadline N and strong griefing resistance, with U_0 and U_3 controlled by the same entity. This MHP protocol implies the existence of a FSE protocol π_{FSE} between relays U_1 and U_2 that achieves strong fairness, 1-effectiveness, and timeliness with a finite deadline T.*

Proof Sketch. Given any such MHP protocol $\pi_{\text{GR-MHP}} = (\Pi_0, \Pi_1, \Pi_2, \Pi_3)$, we can construct a 2-party FSE protocol $\pi_{FSE} = (\Pi_P, \Pi_Q)$, illustrated in Fig. 8. $\mathcal{P}$ simulates the execution of Π_0, Π_1, Π_3, while party Q executes Π_2. The secret exchange is realized by mapping the lock messages: $\mathcal{P}$'s secret corresponds to m^1_{lock} (from U_1), Q's secret corresponds to m^2_{lock} (from U_2). Therefore, for any execution transcript of π_{FSE}, there exists a corresponding execution transcript of $\pi_{\text{GR-MHP}}$. Strong fairness directly follows Lemma. 1. The correctness of MHP implies the 1-effectiveness of FSE, while the coin availability with deadline N ensures the finite timeliness. The formalized proof is deferred to Appendix B.1. □

4.2 Link FSE in the Public Blockchain World to the FSE with Only Random Beacons.

Lemma 2 implies that solving the griefing-resistant MHP requires solving the FSE in a setting with a public blockchain. We first formalize the FSE problem in two settings: the public blockchain world and the random beacon world. We then demonstrate that FSE in the blockchain world can be reduced to FSE in the beacon world. This reduction allows us to construct impossibility results.

FSE in the Public Blockchain World. We model the FSE in the public blockchain world, denoted π_{FSE}, as a set of three interactive Turing machines (ITMs): $\pi_{FSE} = (\Pi_P, \Pi_Q, \Pi_B)$. Π_P and Π_Q are randomized polynomial-time ITMs executed by parties $\mathcal{P}$ and Q, respectively. Π_B is a deterministic polynomial-time ITM executed by the public blockchain, whose initial state, state transition function, and output function are public parameters. This setting is modeled as a synchronous hybrid world (formalized in Appendix B.2) with functionalities: 1) $\mathcal{F}_{clk}$, the global clock functionality; 2) $\mathcal{F}_{\text{RB}}$ (see Fig. 20 in Appendix B), a random beacon that broadcasts a fresh random number every block/epoch; and 3) $\mathcal{F}_{\text{BC}}$ (see Fig. 22 in Appendix B), a public blockchain functionality. Specifically, when $\mathcal{F}_{\text{BC}}$ receives m at τ, it adds m to a local queue (akin to a mempool) and simultaneously leaks m to the adversary $\mathcal{S}$. Every epoch, $\mathcal{F}_{\text{BC}}$ fetches the random beacon value r, processes all messages in its queue along with r, then invokes the execution of Π_B. The queue is then cleared.

FSE in the Random Beacon World. In contrast to the public blockchain world, an FSE protocol in the random beacon world (formalized in Appendix B.3) consists of only two protocols, $\bar{\Pi}_P$ and $\bar{\Pi}_Q$. The only functionalities available in this world are $\{\mathcal{F}_{clk}, \mathcal{F}_{\text{RB}}\}$.

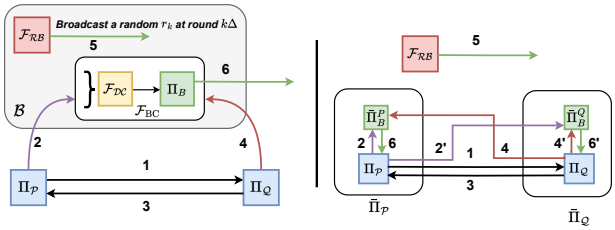


Figure 9: Construction of $(\bar{\Pi}_P, \bar{\Pi}_Q)$ in the $\mathcal{F}_{\text{RB}}$-hybrid world. $\mathcal{F}_{\text{BC}}$ indicates a delayed communication queue.

Theorem 1. *If there exists a FSE protocol $\pi = (\Pi_P, \Pi_Q, \Pi_Q)$ that achieves strong/p-partial fairness, q-effectiveness, and T-timeliness in the $\{\mathcal{F}_{clk}, \mathcal{F}_{RB}, \mathcal{F}_{BC}\}$-hybrid world, then there exists a FSE protocol $\bar{\pi} = (\bar{\Pi}_P, \bar{\Pi}_Q)$ that achieves the same properties in the $\{\mathcal{F}_{clk}, \mathcal{F}_{RB}\}$-hybrid world.*

Proof Sketch. As illustrated in Fig. 9, we first construct a FSE protocol $(\bar{\pi} = (\bar{\Pi}_P, \bar{\Pi}_Q))$ in the random beacon world, given a secure FSE protocol $\pi = (\Pi_P, \Pi_Q, \Pi_B)$ in the blockchain hybrid world. The full proof is in Appendix B.4. The core idea is that the party $\bar{\Pi}_P$/$\bar{\Pi}_Q$ in the new protocol will locally execute $\bar{\Pi}_B^P$/$\bar{\Pi}_B^Q$ as internal subroutines, which simulates $\mathcal{F}_{\text{BC}}$, batching the messages every epoch, and then executing Π_B with the beacon value $\mathcal{F}_{\text{RB}}$. Whenever an internal party (e.g., Π_P) intends to submit a message m to the blockchain, the wrapper protocol $\bar{\Pi}_P$ instead sends m directly to its local blockchain subroutine $\bar{\Pi}_B^P$ and to the other party $\bar{\Pi}_Q$. Since both parties now have the complete set of messages that would have been submitted to the blockchain in any given round, and both have access to the same beacon $\mathcal{F}_{\text{RB}}$, they can each run an local copy of Π_B within $\bar{\Pi}_B^{P/Q}$. This allows them to compute the exact same public transcript that the $\mathcal{F}_{\text{BC}}$ functionality would have produced. Each party will only trust the output of its own local copy. This simulation is perfect. An honest execution of $\bar{\pi}$ is indistinguishable from that of π, thus preserving ***q*-effectiveness** and ***T*-timeliness**. To prove **fairness**, we formalize it with game-based definitions(see Fig. 23, Fig. 24 in Appendix B) and then prove with a reduction. We show that any adversary $\bar{\mathcal{A}}$ capable of breaking the fairness of our

constructed protocol $\bar{\pi}$ in the random beacon world could be used as a subroutine to build an adversary $\mathcal{A}$ that breaks the fairness of the original protocol π in the blockchain world. This would contradict the assumed fairness of π. Therefore, no such $\bar{\mathcal{A}}$ can exist, and $\bar{\pi}$ must be as fair as π. □

Implications. Our reduction has profound implications, as it is a classic result that FSE with strong fairness is impossible to achieve without a trusted third party (TTP) [17, 38]. Given the randomness of random beacons and the deterministic nature of strong fairness, it is impossible to achieve strong fairness in the random beacon world. For partial fairness, existing constructions in the TTP-free, random-beacon world still face significant limitations. Rabin [44] introduced key *disclosure beacons*, enabling a p-partially fair FSE protocol, but with an expected time complexity of $\Omega(1/p)$. While Even et al. [18] eliminated the need for disclosure beacons by using oblivious transfer, their approach requires participants to have equal computational power and only bound that adversary can not get the secret k times faster than the honest one. This is unacceptable in MHPs, considering the delicated timelocks of locked funds. More recent works [30, 31, 47] leverage Verifiable Delay Functions (VDFs) [9] to construct FSEs. However, these are also bound by fundamental limits; Maffei and Roscoe [31] prove that for any p-partially fair FSE without a TTP that consists of $M+1$ messages, the fairness is bounded by $p \geq 1/M$. This lower bound is cross-validiated by the result of Ben-Or et al. [8]. To our best knowledge, no existing FSE protocl breaks this bound by solely introducing random beacons. Combining Lemma 2 and Theorem 1:

Corollary 1. *No lock-resolve MHP with more than one relay that satisfies balance security, correctness, and coin availability can achieve strong griefing resistance. Furthermore, to achieve strong griefing resistance with probability* $\geq 1-p$*, any such MHP requires* $\geq \lceil 1/p \rceil + 1$ *messages.*

5 Protocol Overview

Corollary 1 demonstrates the infeasibility of achieving strong griefing resistance in MHP without incurring significant costs. Therefore, in Zeus, we shift our focus to rational participants, aiming to make the griefing attack "expensive".

Preventing Fee Stealing. As analyzed in Section 3, Blitz's vulnerability to fee stealing stems from its fast-revoke mechanism, which allows relay U_i to accept off-chain revocations from U_{i+1} within a revocation period, even post-payment completion. While enabling efficient off-chain refunds, this exposes relays to fee theft via malicious downstream revocations. A naive solution—removing the fast-revoke mechanism—would reintroduce inefficiency that Blitz originally resolved: locked funds cannot be swiftly reclaimed when a relay lacks liquidity, necessitating costly on-chain transactions. This leads to the first challenge:

Challenge 1. How can we achieve fee stealing resistance without forfeiting the efficiency gains of fast revocation?

A straightforward solution replaces fast revocation with *probing*: each node, from the payer to the payee, sequentially sends a probing message requesting that relays reserve funds. While ensuring routing capability, it introduces critical issues: 1) relays' reserved funds go uncompensated if the payer does not use them, and 2) free probing exposes channel balances, compromising node privacy. Therefore, instead of free probing, we introduce *probing with deposit*, where the payer locks (forwarded by relays) a small deposit to all downstream peers to demonstrate their sincerity: "I, the payer, will use the reserved funds, or I will forfeit my deposit," similar to the security deposit used in rental agreements.

However, *deposit fairness* must be guaranteed: (i) refunds when either the payment completes or a downstream node refuses a liquidity reservation, and (ii) forfeiture to relays whose reserved liquidity remains unused. Achieving this demands reconciling each relay's local channel view. Malicious intermediaries may strategically exploit the discrepancy between local verification and the system-wide truth, leading to allocation conflicts in off-chain enforcement.

Mitigating Griefing Attack. Although prior works [20, 37, 52] propose premium mechanisms to compensate liquidity providers who locked funds in atomic swaps, adapting these solutions to MHPs presents two key challenges:

Challenge 2: How can fair premium distribution be enforced when a revocation is triggered?

Premium-based solutions face a fundamental conflict when integrated with global revocation like those in Blitz. As exemplified in Fig. 2, liquidity consumers must compensate providers immediately upon fund reservation to prevent cost-free griefing. Similarly, to prevent payee-side griefing in MHP, this requires the payee to lock premiums *before* the main payment x (denoted as principal to avoid confusion) locking, allowing relays to claim compensation once they commit liquidity. However, because global revocation can revoke principal regardless of premium settlement, combining premiums with global revocation leads to an unfair outcome: if on-chain revocation is triggered after premium settlement, the honest payee ends up paying premiums to relays *without* ever receiving the promised funds.

Challenge 3: How can we protect premiums from griefing?

Recent work [52] proposes recursive premiums (locking "premiums-of-premiums" until the premium is infinitely small) to mitigate griefing of premiums via infinite recursion. However, this approach suffers from two critical limitations: (i) Bitcoin's dust limits (currently 546 satoshis, $\approx$ \$0.45, denoted as F_{dust}) bound the minimum premium size, making it impossible to achieve infinite recursion; and (ii) these premiums require linearly increasing timelocks. In LN, adversaries can grief premiums up to 483 simultaneous 20-hop payments, yielding approximately 140,000 USD-hours of damage.

Our Solution. *Zeus* addresses these challenges by integrating

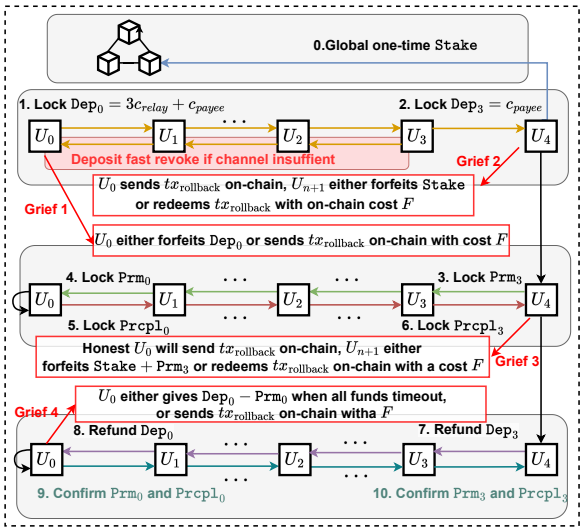


Figure 10: Simplified workflow of *Zeus* in the optimistic case.

deposits and premiums into a Blitz-style MHP, centered on a core principle: requiring both endpoints to have a financial stake before the payment proceeds. Before the principal is locked, the payer and payee commit small, off-chain pledges—deposits and premiums. If any party deviates from the protocol, their pledge is forfeited to the honest participants they harm, making griefing economically irrational.

To enforce this principle and prevent theft of these pledges (Challenge 2), *Zeus* introduces a robust on-chain dispute resolution mechanism as a backstop. While off-chain funds are designed to settle automatically after a timeout (similar to Blitz [5]), disputes trigger an on-chain settlement. The payer (U_0) can initiate a *rollback* by broadcasting a transaction $tx_{rollback}$ that pays the principal directly to the payee (U_{n+1}) on-chain. This transaction serves as a global trigger, revoking all in-flight, payer-side funds. Symmetrically, the payee must redeem this on-chain payment with transaction tx_{redeem}, which in turn revokes all payee-side funds. Should the payee fail to cooperate, the payer can submit transaction tx_{slash}, which simultaneously reclaims their funds and penalizes the payee by slashing a pre-committed on-chain stake. This rollback-redemption process forces endpoints to incur high transaction fees for settlement while atomically revoking all in-flight funds, ensuring relays receive no payment in a dispute. This structure creates a strong, shared incentive to favor the cooperative off-chain path, where endpoints settle with minimal cost and relays earn their fees. As depicted in Fig. 10, we introduce a dedicated deposit-premium locking process.

1. Lock Refundable Deposits. U_0 locks a deposit ($\mathtt{Dep}_0 = c_{payee} + n \cdot c_{relay}$) with a timeout T and forwards it along the path. Relays reserve liquidity upon receiving the deposit and can fast-revoke it upstream if they have insufficient funds. Deposits are held until U_0 fulfills their duty by locking principal. These deposits not only prevent payees from griefing reserved liquidity provided by relays (Challenge 1), but also hedge the griefing risk of premiums (Challenge 3). If the payer aborts after this point (Grief 1), the deposit is forfeited as compensation. To counter payee-side griefing (Grief 2) over deposits, the payee must have a pre-committed on-chain stake; an aborting payee faces a payer-initiated rollback that forces on-chain cooperation (with on-chain transaction cost) or slashing of their stake.

2. Lock Premiums. To hold the payee accountable, U_{n+1} locks a premium with a timeout $T + \Delta$ and sends it backward. Each relay U_i deducts its fee before forwarding the remainder ($\mathtt{Prm}_{i-1} = \mathtt{Prm}_i - f_i$). This premium acts as the payee's pledge. If U_{n+1} aborts now (Grief 3), they forfeit both their premium and their on-chain stake via the rollback-and-slash mechanism, ensuring relays are compensated. This design fully hedges the griefing risk of the principal with only a small on-chain stake, regardless of the principal's amount.

3. Lock Principal. With mutual financial commitments established, U_0 locks the principal (x) plus the total premium ($\mathtt{Prm}_n$). Relays can now safely forward the principal without risk of being grieved.

4. Deposit Refund and Fast Confirmation. Once the principal is locked, the deposits are refunded from U_{n+1} back to U_0. To avoid the inefficiency of waiting for the premium timelock ($T + \Delta$) to expire, we adopt a *fast confirmation* process (like Blitz) initiated by the payer, which expedites the finalization of both premiums and principals.

Security Rationale. *Zeus* ensures that any attempt to abort results in the forfeiture of the malicious party's locked pledge. The on-chain transaction fees required for a rollback-redemption disincentivize endpoints from maliciously triggering disputes. Furthermore, the protocol guarantees fee security by ensuring that relay fees (embedded in the premiums) are revoked atomically in a dispute. This guarantees that either all honest relays are paid or none are, preventing fee theft.

6 *Zeus* Construction

6.1 General Construction for Channel Updates with Two-Stage Revocation

As described in Section 5, for each channel γ_i, Zeus requires a mechanism to achieve: 1) if a global *rollback* event occurs before a deadline prior to T (the timelock of payee-side funds), U_i can reclaim its locked funds α (e.g., principal and/or deposit); 2) if a global *redemption* event occurs, U_{i+1} can reclaim its locked funds β (e.g., premium); and 3) if these events do not occur in time, the counterparty can claim the respective funds. This section details the two-stage revocable channel update protocol, Π_{2SR}, which realizes this functionality. We leverage the pay-or-revoke paradigm, where on-chain *trigger outputs* signal rollback and redemption events. For each γ_i, we assume the existence of a *rollback trigger* $tr^i_{rollback}$ and a *redemption trigger* tr^i_{redeem}, which are globally accessible on $\mathcal{G}_{\mathrm{L}}$ if a dispute arises. Spending these triggers is subject to a relative timelock $t_{close} + \Delta$.

U_{i+1} upon $(\texttt{setup}, \texttt{Meta}_R)$ from $\tilde{U}_{i+1}$: 0) Save $\texttt{Meta}_R$.

U_i upon $(\texttt{init}, \texttt{Meta}_L)$ from $\tilde{U}_i$:

1) Create$(tx_s, tx^l_{Rvk}, tx^r_{Rvk})$ based on $\texttt{Meta}_L$.
2) $\sigma^i_{rvk-l} := \text{Sign}(sk_i, tx_s.\text{Outs}[0] \to tx^l_{Rvk})$
3) $\sigma^i_{rvk-r} := \text{Sign}(sk_i, tx_s.\text{Outs}[1] \to tx^r_{Rvk})$
4) Fill tx^l_{Rvk}, tx^r_{Rvk} with $\sigma^i_s, \sigma^i_{rvk-l}$, and $(tx_s, tx^l_{Rvk}, tx^r_{Rvk}) \hookrightarrow U_{i+1}$

$(tx_s, tx^l_{Rvk}, tx^r_{Rvk}) \overset{1}{\hookleftarrow} U_i, U_{i+1}$:

5) Check (i) if transactions are aligned with $\texttt{Meta}_R$; and (ii) if signatures are correct. Abort if any check fails.
6) $\sigma^{i+1}_s := \text{Sign}(sk_{i+1}, tx_{fund}.\text{Outs}[0] \to tx_s)$
7) $\sigma^{i+1}_{rvk-l} := \text{Sign}(sk_{i+1}, tx_s.\text{Outs}[0] \to tx^l_{Rvk})$
8) Send $(\sigma^{i+1}_s, \sigma^{i+1}_{rvk-l}) \hookrightarrow U_i$
9) Setup rollback and redemption handlers in case of disputes.

$(\sigma^{i+1}_s, \sigma^{i+1}_{rvk-l}) \overset{2}{\hookleftarrow} U_{i+1}, U_i$:

10) Check signatures, sign tx_s and update channel $(\texttt{UPDATE}, \gamma_i.id, tx_s) \hookrightarrow \mathcal{G}_\text{L}$
11) Upon $\mathcal{G}_\text{L}$ returns $\texttt{UPDATED}$, setup rollback and redemption handlers.

Figure 11: Π_{2SR}: (Informal) Channel update with two stage revocation. $\tilde{U}_i$ denotes the external user invoking interface.

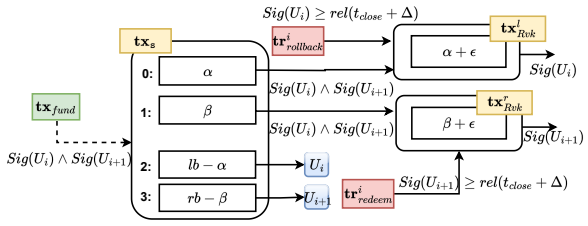


Figure 12: Two-stage revocable channel update transactions.

Let $\texttt{Meta} := \{\gamma_i, \alpha, \beta, T, tr^i_{rollback}, tr^i_{redeem}\}$ denote the parameters governing a state transition. As depicted in Fig. 12, Π_{2SR} employs three key transactions:

1) tx_s: State transaction representing a state with two main outputs: (i) An output of amount α, funded by U_i's channel balance. It is spendable either by U_i via tx^l_{Rvk} (if $tr^i_{rollback}$ is spent) or by U_{i+1} after an absolute timelock T. (ii) An output of amount β, funded by U_{i+1}. It is spendable either by U_{i+1} via tx^r_{Rvk} (if tr^i_{redeem} is spent) or by U_i after timelock $T + \Delta$.

2) tx^l_{Rvk}: The left revoke transaction. It spends the α-output from tx_s (when $tr^i_{rollback}$ is on $\mathcal{G}_\text{L}$) and creates a single output locked to U_i's public key pk_i.

3) tx^r_{Rvk}: The right revoke transaction. It spends the β-output from tx_s (when tr^i_{redeem} is on $\mathcal{G}_\text{L}$) and creates a single output locked to U_{i+1}'s public key pk_{i+1}.

The protocol for establishing this state (Fig. 11) proceeds as follows. Initially, both parties agree on their respective metadata, $\texttt{Meta}_L$ for U_i and $\texttt{Meta}_R$ for U_{i+1}. U_i creates tx_s, tx^l_{Rvk}, and tx^r_{Rvk}. U_i signs the inputs to both revocation transactions, pre-authorizing them. It then sends these transactions to U_{i+1}. U_{i+1} verifies that the transactions conform to $\texttt{Meta}_R$ and that U_i's signatures are valid. It then signs the inputs for tx_s and tx^l_{Rvk} and returns these signatures to U_i. At this point, U_{i+1} holds a fully-signed tx^r_{Rvk}. U_i receives the signatures, creating a fully-signed tx_s and tx^l_{Rvk}. U_i can now update the channel state by broadcasting tx_s.

Once the channel is updated, both parties arm their respective *handlers* ($\Pi^i_{\texttt{rollback-handler}}$ and $\Pi^i_{\texttt{redeem-handler}}$) with the new state information. These handlers monitor $\mathcal{G}_\text{L}$ for the triggers and execute fund recovery. For the left side fund α, the rollback handler works as follows:

1) *Timely Revoke:* If $tr^i_{rollback}$ is finalized on $\mathcal{G}_\text{L}$ no later than block $T - t_\text{close} - \Delta$ and the channel's latest state is still tx_s, U_i's handler has sufficient time to act. It first attempts a cooperative off-chain refund from U_{i+1}. If that fails, it can force an on-chain settlement by closing γ_i (taking up to t_{close} rounds) and broadcasting tx^l_{Rvk}, which will be confirmed before the absolute timelock T expires.

2) *Timeout:* Conversely, if $tr^i_{rollback}$ has not been confirmed in $\mathcal{G}_\text{L}$ by $T - t_\text{close} - \Delta$, U_{i+1} will request U_i to settle the payment α at block $T - t_\text{close} - \Delta$. U_i can safely settle this off-chain, as long as $tr^i_{rollback}$ will never appear on $\mathcal{G}_\text{L}$. If U_i does not respond, U_{i+1} can close γ_i at block $T - t_\text{close}$ (confirmed in block T) and spend the α amount right after block T. A critical race condition arises if $tr^i_{rollback}$ is finalized after $T - t_\text{close} - \Delta$. In this scenario, both U_i (via tx^l_{Rvk}) and U_{i+1} (via timeout at T) could attempt to claim α simultaneously, breaking the balance security. In Zeus, we use the same method as Blitz to fully prevent this race condition, as detailed in Appendix A.1. A symmetric process applies to the β amount and the redemption trigger. Fig 17 in Appendix A illustrates the timeline of the timely-revocation and timeout cases.

Functionality. The protocol Π_{2SR} and its handlers realize the ideal functionality $\mathcal{F}_{2SR}$ (highlighted in Fig. 13). $\mathcal{F}_{2SR}$ guarantees that if a trigger is finalized on $\mathcal{G}_\text{L}$ before its deadline, the original funder can reclaim their funds. Otherwise, the counterparty is entitled to claim them. This mechanism provides a robust foundation for fairly managing deposits, premiums, and principals in Zeus.

6.2 Multi-Hop Payment Description

In this section, we elaborate on the MHP protocol. The formal specifications are specified in Appendix D.

Setup. U_0 cooperates with U_{n+1} to prepare transactions for potential rollback, redemption, and slashing (Fig. 14):

1) $tx_{rollback}$: $tx_{rollback}$ takes one UTXO (i.e., output of tx_{src} in Fig. 14) owned by U_0 as input, with the following primary outputs: (i) a payment with amount $x + (n+1)\varepsilon$ spendable by (a) $Sig(U_{n+1})$ after relative timelock $rel(\Delta)$; or (b) reclaimed by U_0 after $rel(2\Delta)$; or (c) anyone after block height $T - t_\text{close} - 2\Delta$. (c) means if $tx_{rollback}$ is submitted after $T - t_\text{close} - 2\Delta$, the payment will be instantly claimed by any party observing this. Therefore, tx_{redeem} will be finalized no later than $T - t_\text{close}$. (ii) $n+1$ dust outputs with value ε, denoted as *rollback triggers*. $tr^i_{\texttt{rollback}}$ can be redeemed by

$\mathcal{F}_{2SR}$

Parameters: γ_i is a channel between U_i and U_{i+1}, lb and rb are the left and right balances of U_i and U_{i+1} in γ_i, respectively.
API:

Upon $(\texttt{setup}, \texttt{Meta}_R) \xrightarrow{\leq \tau} \tilde{U}_{i+1}$: 0) Save $\texttt{Meta}_R$.

Upon $(\texttt{init}, \texttt{Meta}_L) \xleftarrow{\tau} \tilde{U}_i$:
1) If $\texttt{Meta}_L = \texttt{Meta}_R$, create tx_s based on $\texttt{Meta}_L$. Otherwise, abort.
2) If both U_i and U_{i+1} are honest, $\mathcal{F}_{Channels}$ will return $(\texttt{UPDATED}, \gamma_i.id, tx_s)$ within $2 + t_{update}$ rounds.
3) If the payment fails to settle off-chain:

1. Given honest U_i, if $tr^i_{rollback} \in \mathcal{G}_{\text{L}}$ before $T - t_{\text{close}-\Delta}$, then by round $T - \Delta$, γ_i will be closed; if γ_i is closed at state tx_s, the left-side locked funds α are refunded to U_i by T.
2. Given honest U_{i+1}, if $tr^i_{rollback} \notin \mathcal{G}_{\text{L}}$ before $T - t_{\text{close}-\Delta}$, then by round T, γ_i will be closed; if γ_i is closed at state tx_s, the left-side locked funds α are given to U_{i+1} by $T + \Delta$.
3. Given honest U_{i+1}, if $tr^i_{redeem} \in \mathcal{G}_{\text{L}}$ before $T - t_{\text{close}}$, then by round T, γ_i will be closed; if γ_i is closed at state tx_s, the right-side locked funds β are refunded to U_{i+1} by $T + \Delta$.
4. Given honest U_i, if $tr^i_{redeem} \notin \mathcal{G}_{\text{L}}$ before $T - t_{\text{close}}$, then by round $T + \Delta$, γ_i will be closed; if γ_i is closed at state tx_s, the right-side locked funds β are given to U_i by $T + 2\Delta$.

Figure 13: (Informal) 2SR channel update functionality.

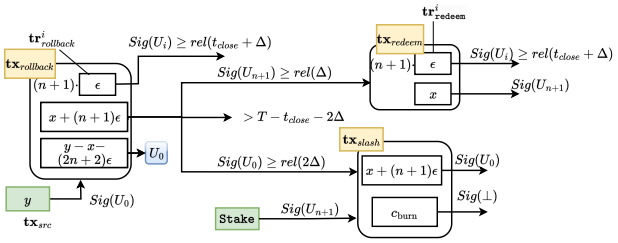


Figure 14: Setup transactions.

U_i's signature after a relative timelock $rel(t_{\text{close}} + \Delta)$. (iii) a change output returning the remaining balance to U_0.

2) tx_{redeem}: tx_{redeem} takes $tx_{rollback}$'s 0-th output as input and contains (i) one output with value x, which can be spent by the payee. (ii) $n+1$ *redeem triggers*. tr^i_{redeem} can be spent by U_{i+1} after a relative timelock $rel(t_{\text{close}} + \Delta)$.

3) tx_{slash}: To discourage the payee's misbehavior, tx_{slash} is prepared. It allows U_0 to reclaim the on-chain payment and burn a pre-committed stake from U_{n+1} if U_{n+1} fails to broadcast tx_{redeem} in a timely manner after $tx_{rollback}$ confirms.

This phase begins with U_0 constructing and broadcasting the unsigned $tx_{rollback}$, tx_{redeem}, and tx_{slash} to all participants. Upon receipt, each relay $U_{i\in[1,n]}$ validates the transactions. Specifically, U_i verifies that: (i) the rollback and redeem triggers for its adjacent channels, γ_{i-1} and γ_i, are correctly embedded within $tx_{rollback}$ and tx_{redeem} respectively, ensuring atomic channel operations; and (ii) tx_{redeem} correctly specifies an output of $tx_{rollback}$ as its input, which prevents fee-stealing attacks by enforcing a sequential on-chain settlement logic. If validation passes, each relay prepares for the subsequent deposit-locking phase. Concurrently, the payee U_{n+1} validates the transactions, signs tx_{slash}, and returns these signatures to U_0. U_{n+1} also arms a *redeem launcher*, a process that monitors the blockchain and automatically broadcasts the fully-signed tx_{redeem} as soon as $tx_{rollback}$ is confirmed.

Upon receiving the partial signature from U_{n+1}, U_0 validates it and then adds its own signatures to construct fully-signed $tx_{rollback}$ and tx_{slash}. Finally, U_0 arms its own handlers: a *rollback launcher* to broadcast $tx_{rollback}$ if the payment stalls (e.g., at timeout $T - t_{close} - 2\Delta$), and a *slash launcher* to broadcast tx_{slash} if U_{n+1} fails to publish tx_{redeem} after $tx_{rollback}$ has been confirmed.

Sequential Fund Lock/Refund. *Zeus* uses the two-stage revocable channel update protocol(Π_{2SR}) to modify channel states, managing off-chain deposits, premiums, and principals to achieve griefing resistance. First, U_0 initiates a forward pass to lock deposits. Sequentially for each channel γ_i from $i = 0$ to n, U_i locks a deposit $\texttt{Dep}_i := c_{\text{payee}} + (n-i) \cdot c_{\text{relay}}$. This is achieved by invoking Π_{2SR} to establish a state where an amount $\alpha = \texttt{Dep}_i$ is locked by U_i. Once deposits are locked across the path, the payee U_{n+1} initiates a backward pass to lock premiums. For each γ_i from $i = n$ down to 0, U_{i+1} locks a premium $\texttt{Prm}_i$. This update, initiated by U_{i+1}, transitions the channel to a new state where $\alpha = \texttt{Dep}_i$ remains locked by U_i and an additional amount $\beta = \texttt{Prm}_i$ is now locked by U_{i+1}. This process can be viewed as a symmetric version of Π_{2SR}, where the roles of U_i and U_{i+1} are reversed. We denote this reversed update sub-protocol as Π_{R-2SR} (see Appendix D.2).

With mutual commitments in place, U_0 initiates another forward pass to lock the principal. For each channel γ_i, U_i locks an identical principal containing the payment amount plus the total premium $\texttt{Prcpl}_i := x + \texttt{Prm}_n$. The channel state is updated via Π_{2SR} to one where U_i's total locked funds are $\alpha = \texttt{Dep}_i + \texttt{Prcpl}_i$, while U_{i+1}'s locked premium $\beta = \texttt{Prm}_i$ is maintained. Finally, after the principal is fully locked, the initial deposits are released in a backward pass initiated by U_{n+1}. For each channel γ_i, the parties transition to a final pre-payment state via Π_{R-2SR} where only the principal and premium are locked ($\alpha = \texttt{Prcpl}_i$, $\beta = \texttt{Prm}_i$). This update supersedes the previous state, effectively refunding $\texttt{Dep}_i$ to U_i. U_{n+1} will send a confirmation message to U_0, indicating that the payment is ready to be made.

Fast Confirmation (Happy path). Once U_0 ensures deposits in γ_0 and γ_{n+1} are refunded and U_{n+1} confirms the payment, to speed up the finalization of the off-chain payment, starting from U_0, each U_i invokes the Π_{2pSettle} sub-protocol to update γ_i to the $\texttt{paid}$ state, where U_i pays $x + \sum_{j=i}^{n} f_j$ to U_{i+1} compared with the IDLE state.

Handling Stalled Payments. Once the off-chain payment in γ_i fails to finish the happy path execution by $T - t_{\text{close}} - 2\Delta$, U_i will submit $tx_{rollback}$ to initiate dispute resolution (illustrated in Fig. 17, Appendix A). Once $tx_{rollback}$ finalizes on $\mathcal{G}_{\text{L}}$, U_i launches tx_{redeem} to claim the payment on behalf of U_{n+1}. These rollback-redeem transactions are captured by the handlers of Π_{2SR} over channel γ_i. As highlighted by functionality

$\mathcal{F}_{2SR}$, all off-chain funds will be revoked atomically.

Handling Timeout Payments. If the payment stalls off-chain while the rollback-redeem fails to finalize in time due to conflicting transactions, the rollback and redeem handlers of Π_{2SR} will take over, claiming funds either through a cooperative off-chain settlement or through on-chain enforcement.

7 Security Analysis

We model *Zeus* in a global universal composability framework, following similar approaches to recent PCN protocols [4–6]. The complete security analysis appears in Appendix F. Here, we denote the global ideal functionalities (e.g., $\mathcal{G}_{\text{L}}$, $\mathcal{F}_{Channels}$) as $\mathcal{F}_{\texttt{prelim}}$. Then, we provide the main functionality $\mathcal{F}_{Zeus}$ in the $\{\mathcal{F}_{\texttt{prelim}}\}$ hybrid world, and prove:

Theorem 2. *Let Σ be an EUF-CMA secure signature scheme, Π_{Zeus} described in Section 6.2 GUC-realizes $\mathcal{F}_{Zeus}$.*

To prove the indistinguishability, for each channel update sub-protocol covered in Section 6, we provide corresponding ideal functionality and prove it with standard simulation paradigm. Then we substitute the real sub-procedures with sub functionality in a hybrid world. Due to space constraints, we informally argue here how *Zeus* achieves *balance security* (Def. 2), *fee security* (Def. 5), and *griefing resistance* (Def. 7).

Balance Security: We first show the payer pays only if the payee gets the payment. With rollback transaction, if payee does not receive the money off-chain, payer can always push tx_{rollup} on-chain, revoking all their locked funds and paying payee directly on-chain. For any honest relay U_i, *Zeus* ensures atomicity for fund revocation across adjacent channels γ_{i-1} and γ_i through linked triggers. Specifically, U_i will check the rollback triggers in both channels are bind in one $tx_{rollback}$(similar for redeem triggers). Furthermore, noting that the functionality provided in $\mathcal{F}_{2SR}$ is identical in every channel. Therefore, the distribution of locked funds is always atomic. For instance, if U_{i-1} can revoke the left-side funds in γ_{i-1}, so U_i can also revoke the left-side funds in γ_i.

Fee Security: For an honest relay $U_{i\in[1,n]}$, once U_i locks any funds, there are two possible cases: 1) *on-chain dispute does not happen:* If U_i has locked the payer's deposit, it is compensated with amount c_{relay}. If U_i has locked the principal, it can claim its relay fee, from the premium locked by U_{i+1}. 2) *on-chain dispute happens:* The payer directly settles the deal with the payee on-chain. Off-chain principals will be revoked, so the payment will not settle off-chain.

Griefing Resistance against k-Cost-Sensitive Adversaries: Table. 4 in Appendix A demonstrates the griefing damage and cost across MHP protocols, under different adversary setting. We put the full analysis in Appendix F.3.

In short, when the payee griefs multiple deposits by forfeiting payee's on-chain stakes, if `Stake` significantly exceeds the cumulative (and typically small) liquidity inconvenience across all these minimal deposit locks, the attack is economically irrational. Fig. 15d shows a 2.5 USD on-chain stakes can make the griefing factor lower than 0.1.

If the payee griefs a single payment after principals fully locked, the griefing damage reaches the peak as all funds are locked and non are refunded. The payee could begin griefing this MHP by triggering the on-chain dispute, and reclaim its premium back at last minute by submitting tx_{redeem}. Doing so costs the adversary an on-chain fee F while incurring a damage of cumulative liquidity inconvenience across channels. Fig. 15b shows even in the worst LN configuration, the factor $\lambda < 0.1$, while $\lambda \leq 0.01$ for payments $\leq$ \$500.

8 Evaluation

Implementation. We implemented *Zeus*[3] using Bitcoin scripts, leveraging the Taproot [41] upgrade. However, Bitcoin does not support the functionality (e.g., covenant [21]) required for slashing. Compared with other expensive contemporary approaches [12, 21], we bypass this with a simple modification of the stake lock script (see Appendix A.2).

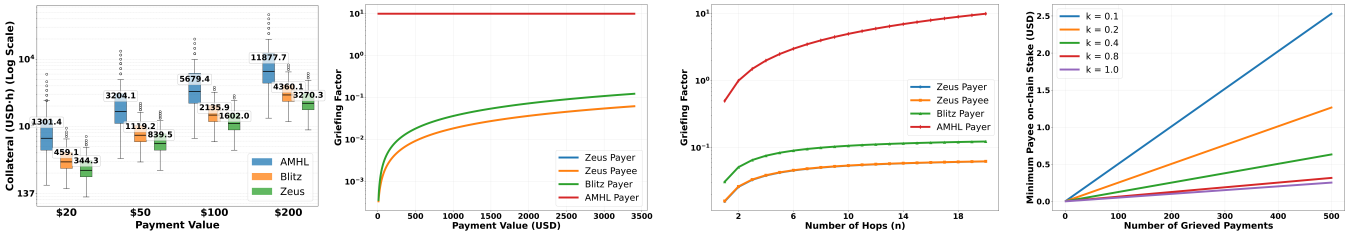
Computation and Communication Overheads. *Zeus* uses only lightweight primitives (i.e., digital signatures), resulting in minimal computational overhead—comparable to standard Bitcoin transactions. To mitigate griefing and fee-stealing, Zeus introduces three additional phases. Consequently, the settlement time of Zeus ($\sim$ 11.3s) is 6.4s longer than that of standard HTLC MHP (4.9s) under a 200ms network delay.

Case	**HTLC/AMHL**		**Blitz**		***Zeus***	
	# txs	vBytes	# txs	vBytes	# txs	vBytes
Payer rollback/revokes	-	-	1	154 + 43n	1	154 + 43n
Payee redeems	-	-	-	-	1	154 + 43n
Relay revokes	1	138	1	194	1	194
Relay settles	1	154/138	1	138	1	138

Table 2: Transaction Costs. "Relay revokes/settles" indicates the cost of force revocation/settlement after channel closure.

Transaction Overheads. In Bitcoin, transaction fees are calculated as vBytes $\times$ fee rate. For context, a standard P2TR transaction is 111 vBytes, costing $\sim$0.33 USD at a moderate fee rate of 3 satoshis/vByte. Ideally, payments in *Zeus* settle entirely off-chain with no on-chain costs. As compared in Table 2, when an n-relay payment stalls off-chain, *Zeus*, similar to Blitz, allows the payer to revoke the jammed payment with a single transaction, sizing $154+43n$ vBytes. Even in a worst-case scenario where $n = 20$, the resulting transaction size is 914 vBytes (2.25 USD). This is substantially below the Bitcoin transaction limit of 10^6 vBytes. Symmetrically, *Zeus* mandates that the payee place tx_{redeem} sized at $154+43n$ vBytes in the dispute, thereby increasing the cost for adversarial payers attempting griefing attacks. Table 3 in Appendix A details the channel closure costs in Zeus.

[3] https://anonymous.4open.science/r/Zeus-074B



(a) Uniformly sampled collateral. (b) Griefing under the longest path. (c) Griefing with max value. (d) Griefing multiple deposits.

Figure 15: (a) shows the collateral distribution given various payment values under uniform sampling. (b) and (c) evaluate how the griefing factor varies when different adversaries grief a single payment across protocols. (b) fixes the payment path length at 20. (c) fixes the payment value at $3400. (d) shows the minimum on-chain stake required by the payee for a target griefing factor.

Fee Stealing and Griefing Simulation. We captured the LN topology in March 2025 from local LN clients, when the BTC price was ∼82k USD. The topology contains 15,447 nodes and 71,007 edges (channels) with an average capacity of approximately 0.057 BTC. For HTLC, we used the minimum lock delta δ of 18 blocks, and the channel closure time is 6 blocks. In LN, the single payment limit is 0.042 BTC (est. $3400), and the maximum number of hops is 20. To simulate MHPs, we (uniformly or liquidity-weighted) sampled endpoints and then found the cheapest relay path. We assume a 1% annual interest rate (matching ACINQ's LN rate).

Griefing Attack Analysis. We assess griefing vulnerability by measuring griefing damage and griefing factor for MHPs.

Griefing Damage. Fig. 15a illustrates how collateral scales with the payment amount. For a 200 USD payment, the inefficiency of linear collateral MHPs like AMHL results in avg. 11,877 USD·h collateral. This vulnerability escalates dramatically if a payee griefs multiple payments concurrently; with an upper bound of 483 such payments, potential damage could reach 5.5M USD·h. Both Blitz and *Zeus* significantly reduce griefing damage compared to HTLC/AMHL—by approximately 63% and 72%, respectively—by implementing constant timelocks across channels. *Zeus* exhibits 25% lower collateral than Blitz due to its shorter principal timelock ($t_{close} + 2\Delta$) versus Blitz's ($t_{close} + 3\Delta$, see [5]'s Fig.10).

Griefing Factor. *Zeus* enforces a minimal griefing factor (GF), rendering attacks economically irrational by combining liquidity costs (opportunity loss from locked funds) and transaction fees. Unlike all other MHPs—where payees grief payments with an *unbounded GF*—*Zeus* caps GF at ≤ 0.1 even at maximum payment values (Fig. 15b), regardless of payer/payee compromise. Fig. 15b further demonstrates that for typical payments less than $500 USD, GF is less than $0.01 \ll 1$. Fig. 15c further shows GF's path-length dependence. Note that *Zeus* requires payees to pledge a one-time on-chain stake. Fig. 15d quantifies the minimum required stake to ensure the griefing factor remains below a threshold k. Even when a payee simultaneously griefs 483 MHPs and bounds $GF \leq 0.1$, the stake remains a few USD.

Fee Stealing Attack Analysis. We evaluate fee-stealing attacks by measuring the maximum portion of relay fees that colluding nodes can capture. We designate the top μ liquid nodes as potential colluders. The results demonstrate significant vulnerability. Under weighted sampling (Fig. 16), merely 0.6% of network nodes can capture over 40% of total relay fees. Under uniform sampling (Fig. 19 in Appendix A), the same fraction captures approximately 25% of fees. This disparity stems from the LN's hub-and-spoke structure, where high-liquidity nodes naturally attract more routing traffic. Under weighted sampling, payments preferentially traverse these central nodes, amplifying adversarial advantage.

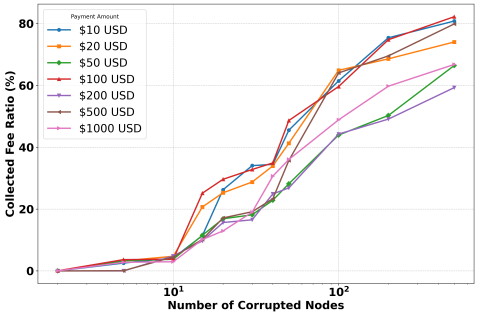


Figure 16: Fee ratio under weighted sampling.

9 Discussion

Improve capital efficiency. In *Zeus* and other Bitcoin-compatible global-state based MHPs (e.g., Blitz and Donner), payers need to reserve on-chain funds (i.e. tx_{src} in *Zeus*) to resolve disputes, synchronizing off-chain channels. To get rid of this, we can utilize channel splicing [49], a LN feature allowing users to *partially* withdraw their channel balance *without* closing channel. In these MHPs, the payer U_0 can use their balance in channel γ_0 as funding source to resolve disputes, and delegate the payer role to the first relay U_1.

UTXO Interoperability. UTXO model lacks native support for synchronization among UTXOs. Protocols like *Zeus* rely on dust outputs to signal state changes, costly and contributing to UTXO set bloat. Exploring alternatives for UTXO interoperability would benefit the broader Bitcoin scaling ecosystem.

Ethical Considerations

Our research, which identifies new vulnerabilities in MHP protocols and proposes the *Zeus* protocol as a remedy, was guided by a careful ethical analysis considering all stakeholders. These include the research community, open-source developers, PCN relay operators, and end-users. Our analysis follows established principles to ensure our work benefits the community while minimizing potential risks.

Beneficence We weighed the positive contributions of our research against its potential negative impacts.

1. Benefits: The primary benefit of our work is the improved security and fairness of PCNs. By identifying subtle but critical vulnerabilities, we enable the community to secure systems against sophisticated fee-stealing and griefing attacks. Our protocol, *Zeus*, offers a concrete, provably secure path forward, contributing to the long-term health, stability, and trustworthiness of the PCN ecosystem. Furthermore, our impossibility result for griefing resistance provides a crucial theoretical boundary, guiding future research toward more practical and achievable security goals.

2. Potential Harms: The primary potential harm is the premature public disclosure of vulnerabilities before mitigations are deployed. The fee-stealing attack and griefing attack have been demonstrated in prior works [33, 50]. Our work primarily reveals vulnerabilities in recent academic MHP proposals that have not been deployed in production networks. While the industry-led PTLC proposal (an AMHL variant) is vulnerable to our fee-stealing attack, its development is still under community discussion and it has not been deployed on the Lightning Network. Therefore, our disclosure does not create any new risk for existing users or funds.

Mitigation via Responsible Disclosure To mitigate the risk associated with disclosing new attacks, we are committed to a coordinated vulnerability disclosure process.

Implementation and Evaluation Ethics. We conducted all experimental evaluations ethically by: i)Our evaluation was conducted using a publicly available snapshot of the Lightning Network's topology. We did not perform any attack experiments on the live Bitcoin or Lightning Network. All protocol implementations and simulations were run in a controlled, local environment. ii) Ensuring that the results are reproducible and verifiable by making the full implementation and evaluation scripts publicly available.

Open Science

Artifact Availability Our artifacts, including the source code for *Zeus*, all evaluation scripts, and the Lightning Network topology data used in our experiments, are available in an anonymized repository: https://anonymous.4open.science/r/Zeus-074B. The repository includes a README file with detailed instructions for reproducing our results.

Methodological Transparency In appendix, we provide a comprehensive, formal description of our methodology, including the system and threat models, protocol designs, and security proofs. The evaluation section details the experimental setup, parameters, and metrics used to ensure our results can be independently verified and extended by other researchers.

References

[1] LN BOLT 2: Peer protocol for channel management, 2024.

[2] bitcoinjs-lib - v7.0.0. https://bitcoinjs.github.io/bitcoinjs-lib/, 2025.

[3] Lukas Aumayr, Kasra Abbaszadeh, and Matteo Maffei. Thora: Atomic and privacy-preserving multi-channel updates. In *Proceedings of the 29th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[4] 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 *Proceedings of the 27th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 635–664. Springer, 2021.

[5] Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Blitz: Secure Multi-Hop payments without Two-Phase commits. In *Proceedings of 30th USENIX Security Symposium (USENIX Security)*, 2021.

[6] Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Breaking and Fixing Virtual Channels: Domino Attack and Donner. In *Proceedings of the 30th Network and Distributed System Security Symposium (NDSS)*, 2023.

[7] 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, 2022.

[8] Michael Ben-Or, Oded Goldreich, Silvio Micali, and Ronald L Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory (TIT)*, 36(1):40–46, 2002.

[9] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *37th Annual international cryptology conference (CRYPTO)*, pages 757–788. Springer, 2018.

[10] Ran Canetti. Universally composable signature, certification, and authentication. In *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004. (CSFW)*, 2004.

[11] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *Proceedings of the 4th Theory of Cryptography (TCC)*, 2007.

[12] Xinshu Dong, Orfeas Stefanos Thyfronitis Litos, Ertem Nusret Tas, David Tse, Robin Linus Woll, Lei Yang, and Mingchao Yu. Remote staking with optimal economic safety, 2025.

[13] Yuhao Dong, Ian Goldberg, Sergey Gorbunov, and Raouf Boutaba. Astrape: Anonymous payment channels with boring cryptography. In *Proceeding of the 20th International Conference on Applied Cryptography and Network Security (ACNS)*, pages 748–768. Springer, 2022.

[14] Christoph Egger, Pedro Moreno-Sanchez, and Matteo Maffei. Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In *Proceedings of the 26th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[15] Oğuzhan Ersoy, Pedro Moreno-Sanchez, and Stefanie Roos. Get me out of this payment! bailout: An htlc re-routing protocol. In *Proceedings of 27th Financial Cryptography and Data Security (FC)*, page 92–109. Springer, 2023.

[16] Oğuzhan Ersoy, Jérémie Decouchant, Satwik Prabhu Kimble, and Stefanie Roos. Syncpcn/psyncpcn: Payment channel networks without blockchain synchrony. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies (AFT)*, pages 16–29, 2022.

[17] S. Even and Yacov Yacobi. Relations amoung public key signature systems. *Technion - Isrel Institute of Technology, Technical Report 175*, 1980.

[18] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[19] William George. An analysis of griefs and griefing factors. *Frontiers in Blockchain*, 6:1137155, 2023.

[20] Runchao Han, Haoyu Lin, and Jiangshan Yu. On the optionality and fairness of atomic swaps. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies (AFT)*, pages 62–75, 2019.

[21] Ethan Heilman, Victor I Kolobov, Avihu M Levy, and Andrew Poelstra. ColliderScript: Covenants in Bitcoin via 160-bit hash collisions. *Cryptology ePrint Archive*, 2024.

[22] Ethan Heilman, Victor I. Kolobov, Avihu M. Levy, and Andrew Poelstra. ColliderScript: Covenants in bitcoin via 160-bit hash collisions. Cryptology ePrint Archive, Paper 2024/1802, 2024.

[23] Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 245–254, 2018.

[24] Maxim Jourenko, Mario Larangeira, and Keisuke Tanaka. Payment trees: Low collateral payments for payment channel networks. In *Proceedings of the 25th Financial Cryptography and Data Security (FC)*, pages 189–208. Springer, 2021.

[25] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *Proceedings of Theory of Cryptography Conference (TCC)*, 2013.

[26] Satwik Prabhu Kumble, Dick Epema, and Stefanie Roos. Game-theoretic analysis of (non-) refundable fees in the lightning network. In *Proceedings of the 29th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 645–652. IEEE, 2023.

[27] Yehuda Lindell. Fast secure two-party ecdsa signing. *Journal of Cryptology*, 34(4):44, 2021.

[28] Robin Linus, Lukas Aumayr, Alexei Zamyati, Andrea Pelosi, Zeta Avarikioti, and Matteo Maffei. BitVM2: Bridging bitcoin to second layers. https://bitvm.org/bitvm_bridge.pdf, 2024.

[29] Aumayr Lukas, Avarikioti Zeta, Salem Iosif, Schmid Stefan, and Yeo Michelle. X-transfer: Enabling and optimizing cross-pcn transactions. In *Proceedings of 20th Financial Cryptography and Data Security (FC)*, 2025.

[30] Ivo Maffei and A. W. Roscoe. Optimally-fair multi-party exchange without trusted parties. In *28th European Symposium on Research in Computer Security (ESORICS)*, page 313–333. Springer, 2023.

[31] Ivo Maffei and Andrew W Roscoe. Optimally-fair exchange of secrets via delay encryption and commutative blinding. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 94–111. Springer, 2023.

[32] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[33] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *Proceedings of the 26th Network and Distributed System Security Symposium (NDSS)*, 2019.

[34] Subhra Mazumdar, Prabal Banerjee, and Sushmita Ruj. Time is money: Countering griefing attack in lightning network. In *Proceeding of the 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1036–1043. IEEE, 2020.

[35] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. In *Proceedings of Financial Cryptography and Data Security (FC)*, pages 508–526. Springer, 2019.

[36] Ayelet Mizrahi and Aviv Zohar. Congestion attacks in payment channel networks. In *Proceedings of the 25th International Conference on Financial Cryptography and Data security (FC)*, pages 170–188. Springer, 2021.

[37] Tejaswi Nadahalli, Majid Khabbazian, and Roger Wattenhofer. Grief-free atomic swaps. In *Proceedings of the 4th IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2022.

[38] Henning Pagnia and Felix C. Gartner Darmstadt. On the impossibility of fair exchange without a trusted third party. 1999.

[39] Henning Pagnia, Holger Vogt, and Felix C Gärtner. Fair exchange. *The Computer Journal*, 46(1):55–75, 2003.

[40] Cristina Pérez-Sola, Alejandro Ranchal-Pedrosa, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joaquin Garcia-Alfaro. Lockdown: Balance availability attack against lightning network channels. In *Proceedings of the 24th International conference on financial cryptography and data security (FC)*, pages 245–263. Springer, 2020.

[41] Anthony Towns Pieter Wuille, Jonas Nick. Bip-341: Taproot: Segwit version 1 spending rules. https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki, 2020.

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

[43] Minfeng Qi, Qin Wang, Zhipeng Wang, Manvir Schneider, Tianqing Zhu, Shiping Chen, William Knottenbelt, and Thomas Hardjono. Sok: Bitcoin layer two (l2), 2024.

[44] Michael O Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences (JCSS)*, 27(2):256–267, 1983.

[45] Sophie Rain, Georgia Avarikioti, Laura Kovács, and Matteo Maffei. Towards a game-theoretic security analysis of off-chain protocols. In *Proceedings of the 36th IEEE Computer Security Foundations Symposium (CSF)*, pages 107–122. IEEE, 2023.

[46] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks. In *Proceeding of the 4th IEEE european symposium on security and privacy workshops (EuroS&PW)*, pages 347–356. IEEE, 2019.

[47] A. Roscoe and Peter Ryan. Auditable pakes: Approaching fair exchange without a ttp. In *28th Cambridge International Workshop on Security Protocols (SPW)*, pages 278–297, 2017.

[48] Rusty Russell. Loop attack with onion routing. https://diyhpl.us/~bryan/irc/bitcoin/bitcoin-dev/linuxfoundation-pipermail/lightning-dev/2015-August/000135.txt, 2015.

[49] Rusty russell. Splicing proposal. https://github.com/lightning/bolts/pull/863, 2021.

[50] Clara Shikhelman and Sergei Tikhomirov. Unjamming lightning: A systematic approach. 2022.

[51] Kirti Singh, Vinay J Ribeiro, and Susmita Mandal. 4-swap: Achieving grief-free and bribery-safe atomic swaps using four transactions. In *6th Conference on Advances in Financial Technologies (AFT)*, 2025.

[52] Yingjie Xue and Maurice Herlihy. Hedging against sore loser attacks in cross-chain transactions. In *Proceedings of the 40th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 155–164. Association for Computing Machinery, 2021.

[53] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167. IEEE, 1986.

[54] Yi Zhang, Xiaofeng Jia, Bianjing Pan, Jun Shao, Liming Fang, Rongxing Lu, and Guiyi Wei. Anonymous multi-hop payment for payment channel networks. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 21(1):476–485, 2023.

A Supplemental Discussions of Zeus.

State	$tx_{\texttt{IDLE}}$	$tx_{\texttt{Lock-Dep}}$	$tx_{\texttt{Lock-Prm}}$	$tx_{\texttt{Lock-Prcpl}}$	$tx_{\texttt{Rfd-Dep}}$	$tx_{\texttt{Paid}}$
vsize	111	154	197	197	197	111

Table 3: Channel closure size of *Zeus* in different states.

A.1 Mitigating Race Conditions via Enforced Timeliness

A critical race condition in dispute resolution arises if the finality of $tx_{rollback}$ cannot be determined by a fixed block height $T - t_{close} - \Delta$. To guarantee that $tx_{rollback}$ is either confirmed by this strict deadline or never confirmed at all, we enforce its timeliness by introducing an intermediate transaction, tx_m. Instead of being funded directly from a source UTXO (tx_{src}), $tx_{rollback}$ is funded from an output of tx_m, creating the chain $tx_{src} \to tx_m \to tx_{rollback}$.

The output of tx_m is encumbered with a script that enforces two mutually exclusive spending paths. The first path allows U_0 to spend the output with $tx_{rollback}$ only after a relative timelock of Δ has passed. The second path allows anyone to spend the output after an absolute timelock of $T - t_{close} - 2\Delta$.

To trigger a rollback, U_0 must first broadcast tx_m. To meet the deadline, U_0 must ensure tx_m is confirmed by block $T - t_{close} - 2\Delta$, which requires broadcasting it around $T - t_{close} - 3\Delta$ to account for confirmation delays. If U_0 fails to broadcast $tx_{rollback}$ by $T - t_{close} - 2\Delta$, the public spending path on tx_m's output becomes active. Any observer can then spend this output, permanently invalidating it as an input for $tx_{rollback}$. Consequently, by observing the blockchain after block $T - t_{close} - \Delta$, any party can definitively conclude whether $tx_{rollback}$ can ever be confirmed. This mechanism deterministically resolves the race condition.

A.2 Slash Functionality over Bitcoin Script

Our protocol requires the payee's on-chain stake to implement a slash mechanism with conditional spending rules: 1) early withdrawal before expiry time T_{expire} with a penalty c_{burn}, or 2) full withdrawal after T_{expire} without penalty. Implementing such conditional forfeiture in Bitcoin's stateless script environment traditionally requires covenants—proposed opcodes that constrain how UTXOs can be spent [4]. While various covenant proposals exist, most require consensus changes. Alternative approaches like multi-signature committees [28] or hash-based constructions [22] avoid soft forks but introduce additional trust assumptions or computational overhead.

We propose a practical alternative that leverages Bitcoin's existing fee market mechanism. Instead of burning c_{burn}, we structure the early withdrawal path to require excessive transaction fees, effectively transferring the penalty to miners. This is implementable using Taproot's script-path spending by constructing a computationally expensive witness script for early withdrawal (e.g., requiring multiple redundant signature verifications or hash computations). The honest payee uses the key-path spending after T_{expire}, incurring only standard transaction costs.

The security of this approach relies on the honesty of Bitcoin miners. Given that c_{burn} represents a small fraction of typical block rewards ($0.25–$5 versus $250,000), and Bitcoin's mining ecosystem exhibits strong decentralization, coordination between miners and adversarial payees is both economically irrational and operationally infeasible. The penalty mechanism thus provides equivalent security guarantees to direct burning while maintaining compatibility with current Bitcoin consensus rules.

A.3 Value of Deposits and Stakes

At the beginning of a payment, the payee needs to lock a stake `Stake` on-chain to hedge against potential griefing of the payer's off-chain deposit. Recall that the off-chain deposit of the payer in channel γ_i ($i \in [1,n]$), denoted as $\texttt{Dep}_i$, has a value of $c_{\text{payee}} + (n-i) \cdot c_{\text{relay}}$. This off-chain deposit hedges the griefing risk associated with the future premium, which amounts to $f_0 + \sum_{i=1}^{n} f_i$, where f_i is the fee charged by relay node U_i, and f_0 is the padding fee. The padding fee ensures that the relay fee in each hop is sufficient (greater than the dust limit F_{dust}) to be safely routed through the PCNs:

$$\text{For } \forall k \in [1,n], \quad \sum_{i=0}^{k} f_i \geq F_{\text{dust}} \tag{1}$$

Next, we show that to ensure balance security, the payer's deposit must satisfy:

$$c_{\text{payee}} \geq \sum_{i=1}^{n} f_i + F_{\text{dust}} \tag{2}$$

This requirement can be understood by examining the last step of the payer's deposit lock phase (`Lock-Dep`), where the last relay U_n locks a payment amount $\texttt{Dep}_n = c_{\text{payee}}$ in channel γ_n to payee U_{n+1}. The payee then locks the premium back

[4] https://bitcoinops.org/en/topics/covenants/

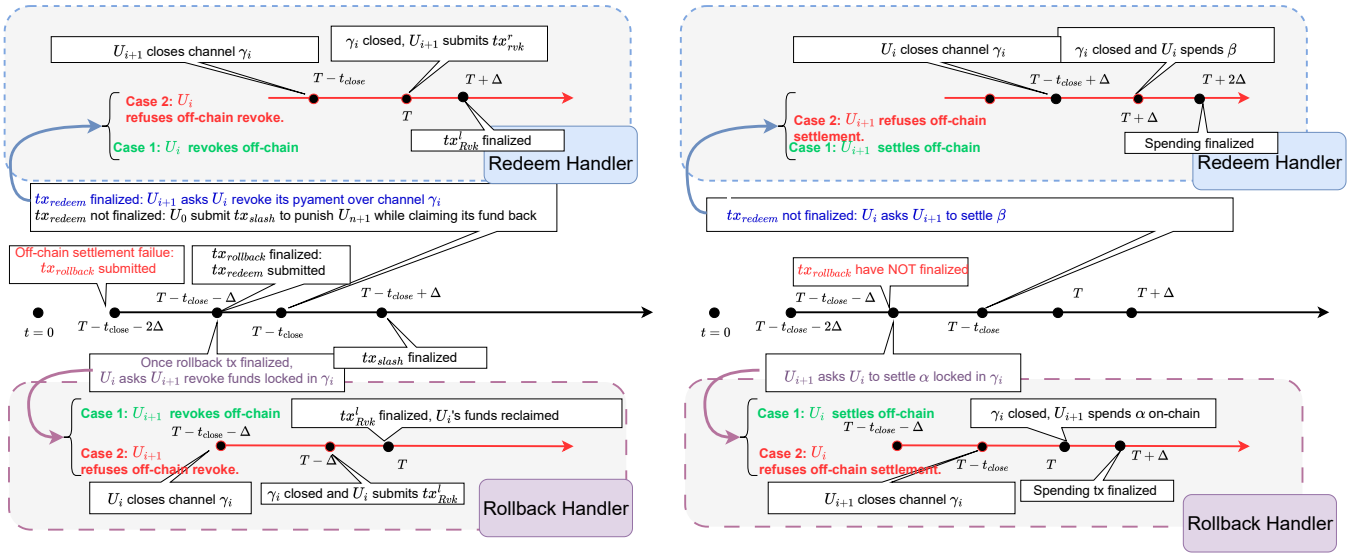


Figure 17: The timeline of the *Zeus* execution. The left part shows how handlers gurantee atomical revocation when on-chain rollback-redemption happens. The right part shows how payments are settled, if the fast confirmation is not completed and on-chain dispute is not triggered.

Protocol	Griefing Damage	Payer corrupted		Payee corrupted		Both corrupted
		payoff_{adv}	payoff_{payee}	payoff_{adv}	payoff_{payer}	payoff_{adv}
HTLC/AMHL	$ir \cdot p \cdot \Theta(n^2x + n^3f)$	-	-	0	0	0
Blitz	$ir \cdot p \cdot \Theta(nx + n^2f)$	$-rp(154+43n)$	0	0	$-43nr$	$-rp(154+43n)$
Zeus(case 1)	$ir \cdot \cdot \Theta(nc_{payee} + n^2c_{relay})$	-	-	$-\min\{rp(154+43n), c_{\text{burn}}\}$	$-43nr$	$-\min\{rp(154+43n), c_{\text{burn}}\}$
Zeus(case 2)	$ir \cdot p \cdot \Theta(n(x+c_{payee}))$ $+n^2(f+c_{relay}))$	$-rp(154+43n)$	$-43rpn$	$-rp(154+43n)$	$-43nr$	$-2rp(154+43n)$

Table 4: Comparison of griefing damage and costs. An adversary controls p in-flight payments with amount x and n relays per payment, each charging a relay fee f, with fee rate r USD/vBytes. ir is the interest rate for griefed funds locked in channels. Case 1: Griefing before the Principal Locking phase; Case 2: Griefing during/after the Principal Locking phase. For on-chain settlement, the payoff for honest parties excludes the idle on-chain transaction costs.

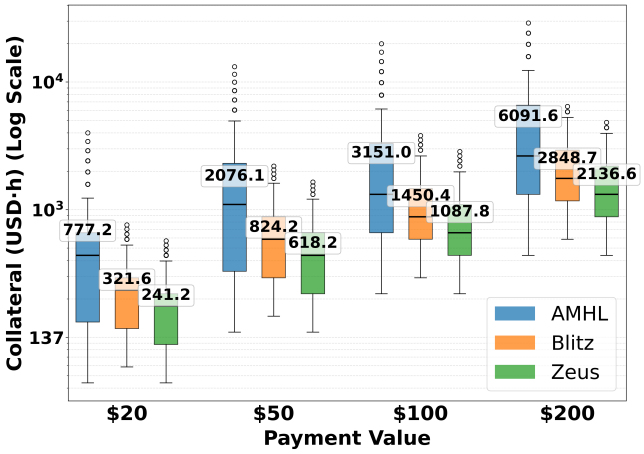


Figure 18: Weighted sampled collateral.

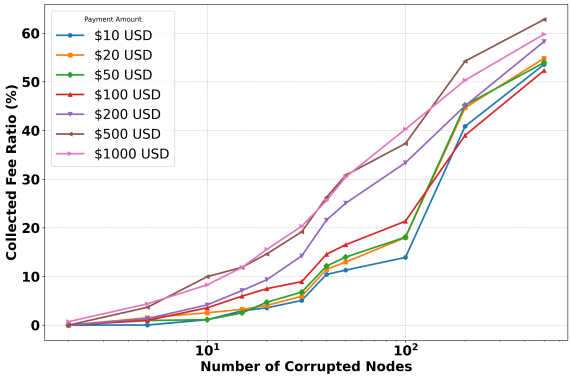


Figure 19: Relay fee collected by corrupted nodes with uniform sampling.

in channel γ_n with an amount $\texttt{Prm}_n = f_0 + \sum_{i=1}^{n} f_i$. To ensure that there is no loss for the payee even if all other parties go offline, we require $c_{\text{payee}} \geq f_0 + \sum_{i=1}^{n} f_i$. Since the padding fee f_0 must satisfy $f_0 \geq F_{\text{dust}} - \sum_{i=1}^{n} f_i$ when $\sum_{i=1}^{n} f_i < F_{\text{dust}}$, we can derive Equation 2.

Now we discuss the maximum values of c_{payee} and the value of c_{relay}. According to Equation 2, the maximum value of c_{payee} is determined by the maximum total relay fees, which depend on the maximum payment amount and the maximum number of hops. In the LN, the default single payment limit is 0.042 BTC (approximately 3,400 USD), and the maximum number of hops is 20. With a typical fee rate of 0.2% (0.01% per hop $\times$ 20 hops), the total fees would be around 6.8 USD. Adding the dust limit (typically 546 satoshis, around \$0.45), the value of c_{payee} is approximately 7.25 USD (around 9k satoshis). The value of c_{relay} can be considered as the relay fee to route c_{payer}. In *Zeus*, we set c_{relay} to the minimum relay fee of 1 satoshi, as this minimum fee is sufficient to cover the liquidity cost of routing the payer's deposit (considering the common fee rate is about 0.01%).

Finally, we calculate the required value of the payee's on-chain stake `Stake`. As shown in Table 4 (case 1), this stake must be sufficient to hedge the griefing risk of p in-flight payer deposits. The worst-case griefing damage (locked amount times locked time) can be calculated as:

$$r \cdot p \cdot (c_{\text{payee}} \cdot n_{\max} + \frac{n_{\max}(n_{\max}+1)}{2} c_{\text{relay}})(t_{\text{close}} + 2\Delta)$$

where:

- $p = 483$ is the maximum number of concurrent MHPs in the Lightning Network.
- $n_{\max} = 20$ is the maximum number of hops.
- $\frac{n_{\max}(n_{\max}+1)}{2} c_{\text{relay}}$ is the total c_{relay} locked in this MHP.
- $(t_{\text{close}} + 2\Delta)$ is the timelock, and r is the interest rate.

Under the worst case, the total griefing collateral is approximately 221k USD·h. Assuming an interest rate of 1% per year (the interest rate charged by ACINQ, one of the largest liquidity providers in the Lightning Network), the griefing damage derived from the payer's collateral is approximately 0.25 USD. If we further constrain the griefing factor to be lower than 0.1, a $\sim$2.5 USD global stake will be sufficient.

B Proofs for Reduction to MHP for FSE

B.1 Formal proof for Lemma 2

Proof. We construct an FSE protocol π_{FSE} and prove its properties by reduction to the security of the MHP protocol $\Pi_{\text{GR-MHP}}$. The core idea is to use the MHP protocol as a mechanism to enforce the fair exchange of secrets between two parties, $\mathcal{P}$ and $\mathcal{Q}$.

- **Parties and Roles:** The FSE protocol involves two parties, $\mathcal{P}$ and $\mathcal{Q}$.
  - Party $\mathcal{P}$ executes Π_P, which simulates the roles of relay U_1 with protocol Π_1 and the endpoints (U_0, U_3) with protocol (Π_0, Π_3).
  - Party $\mathcal{Q}$ executes Π_Q, which simulates the role of relay U_2 by executing protocol Π_2.
- **Secrets:** The secrets to be exchanged are the lock messages from the MHP.
  - $\mathcal{P}$'s secret is $i_{\mathcal{P}} := m^1_{\text{lock}}$.
  - $\mathcal{Q}$'s secret is $i_{\mathcal{Q}} := m^2_{\text{lock}}$.

  The descriptions of the secrets, $\text{desc}_{\mathcal{P}}$ and $\text{desc}_{\mathcal{Q}}$, are the policies that these lock messages must satisfy according to the MHP protocol rules.
- **Protocol Execution:**
  - $\mathcal{P}$ and $\mathcal{Q}$ spawn the MHP protocol $\Pi_{\text{GR-MHP}}$ as described above.
  - Initialize protocols Π_1 and Π_2 with input secrets, where $i_{\mathcal{P}} = m^1_{\text{lock}}$ and $i_{\mathcal{Q}} = m^2_{\text{lock}}$.
  - Simulate the MHP protocol execution.
  - When Π_0, Π_1, and Π_3 terminate at N, $\mathcal{P}$ extracts the lock message m^2_{lock} from the Π_3 transcript (if any) and returns it as $\mathcal{Q}$'s secret. Then $\mathcal{P}$ terminates.
  - When Π_2 terminates at N, $\mathcal{Q}$ extracts the lock message m^1_{lock} from the Π_2 transcript (if any) and returns it as $\mathcal{P}$'s secret. Then $\mathcal{Q}$ terminates.

We now prove that $\pi_{FSE} = (\Pi_P, \Pi_Q)$ satisfies the required properties.

Strong Fairness: We must show that if an honest party provides its secret, it is guaranteed to receive the other's secret.

- Consider an honest $\mathcal{P}$ and a malicious $\mathcal{Q}$. $\mathcal{P}$ honestly simulates U_1 and the endpoints. It sends m^1_{lock} (event $\mathcal{E}_1$) to $\mathcal{Q}$. By Lemma 1 (Part 1), the MHP's properties of Strong Griefing Resistance (for honest U_1) and Balance Security (for the colluding endpoints) guarantee that $\mathcal{E}_1 \implies \mathcal{E}_2$. This means the underlying MHP logic compels the party playing U_2 (i.e., $\mathcal{Q}$) to send m^2_{lock}. Thus, if $\mathcal{Q}$ can extract m^1_{lock} from Π_2's transcript, $\mathcal{P}$ can extract m^2_{lock} from Π_3's transcript, ensuring strong fairness for $\mathcal{P}$ in the FSE.
- Consider an honest $\mathcal{Q}$ and a malicious $\mathcal{P}$. $\mathcal{Q}$ honestly simulates U_2. If $\mathcal{Q}$ sends m^2_{lock} (event $\mathcal{E}_2$), by Lemma 1 (Part 2), $\mathcal{E}_2 \implies \mathcal{E}_1$. This compels the party playing U_1 (i.e., $\mathcal{P}$) to send m^1_{lock}. Thus, if $\mathcal{Q}$ reveals its secret, it receives $\mathcal{P}$'s secret.

Therefore, π_{FSE} achieves strong fairness.

1-Effectiveness: We must show that if both parties are honest, the exchange succeeds.

- If both $\mathcal{P}$ and Q are honest, they correctly simulate their assigned roles in the MHP (U_1, U_0, U_3 for $\mathcal{P}$; U_2 for Q).
- The MHP protocol satisfies *Correctness*, which guarantees that if all parties are honest, the payment settles successfully. In a lock-resolve MHP, successful settlement requires both lock events, $\mathcal{E}_1$ and $\mathcal{E}_2$, to occur.
- This means both parties will send their respective lock messages, m^1_{lock} and m^2_{lock}, successfully completing the secret exchange.

Thus, π_{FSE} achieves effectiveness with probability 1.

Timeliness: We must show that honest parties terminate within a finite time.

- The MHP Simulation Phase is governed by the MHP's *Coin Availability* property. This property ensures that all honest participants in the MHP terminate their involvement within a deadline N.
- Since the execution of π_{FSE} is bound by the execution of the MHP simulation, π_{FSE} must also terminate within deadline N.

Thus, π_{FSE} achieves timeliness.

Since π_{FSE} satisfies strong fairness, effectiveness, and timeliness, the existence of a secure MHP with the stated properties implies the existence of such an FSE protocol. □

B.2 FSE with Public Blockchain

In this section, we provide formal model for the FSE protocol with public blockchain.

$\mathcal{F}_{\text{RB}}$

The functionality is parameterized by a round duration $\bar{\Delta}$ and interacts with a set of parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and an adversary $\mathcal{S}$. It maintains a list of stored random values $L = []$.

- Upon receiving $(\texttt{tick}, k \cdot \bar{\Delta})$ from $\mathcal{F}_{clk}$ for some $k \in \mathbb{N}^+$, generate a fresh random value $r^k \leftarrow\!\!\$\ \{0,1\}^\lambda$. Store (k, r^k) by appending it to L. Send $(\texttt{BeaconOutput}, k, r^k)$ to all parties and to $\mathcal{S}$.
- Upon receiving $(\texttt{Query}, k')$ from a party $\mathcal{P}_i$, search for an entry (k, r^k) in L such that $k = k'$. If the entry exists, send $(\texttt{BeaconValue}, k, r^k)$ to $\mathcal{P}_i$. Otherwise, do nothing.

Figure 20: Ideal functionality of a public random beacon.

FSE Protocol in the Public Blockchain World. An FSE protocol in the public blockchain world is a two-party protocol $\pi = (\Pi_P, \Pi_Q, \Pi_B)$ executed in the $\{\mathcal{F}_{clk}, \mathcal{F}_{\text{RB}}, \mathcal{F}_{\text{BC}}\}$-hybrid model, where Π_P and Π_Q are two randomized PPT ITM machines representing the two parties $\mathcal{P}$ and Q, respectively. And Π_B is a deterministic iTM that represents the public blockchain's state transition logic.

$\mathcal{F}_{\text{DC}}$

The functionality is parameterized by a round duration $\bar{\Delta}$. It maintains an ordered list (FIFO queue) of pending messages, $\mathsf{Q} = []$. It interacts with parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and an adversary $\mathcal{S}$.

- Upon receiving $(\texttt{Send}, m)$ from a party $\mathcal{P}_i$, add m to the end of the queue Q. Send $(\texttt{Sent}, \mathcal{P}_i, m)$ to $\mathcal{S}$ at the same round.
- Upon receiving a message $(\texttt{tick}, k \cdot \bar{\Delta})$ from $\mathcal{F}_{clk}$ for some $k \in \mathbb{N}^+$, let $\mathsf{out}_{\mathsf{batch}} \leftarrow \mathsf{Q}$ and then reset $\mathsf{Q} \leftarrow []$. Send the message $(\texttt{Deliver}, k, \mathsf{out}_{\mathsf{batch}})$ to all parties and to $\mathcal{S}$.

Figure 21: Ideal functionality of delayed communication.

$\mathcal{F}_{\text{BC}}$

The functionality is parameterized by a round duration $\bar{\Delta}$ and a deterministic state-transition machine $\Pi_B = (g, f)$ with an initial state s_0. $\mathcal{F}_{\text{BC}}$ maintains an ordered list (FIFO queue) of pending messages.

Global Setup: Upon $(\texttt{Init}, \mathsf{s}_0, \Pi_B)$ from $\mathcal{E}$, initialize an internal state $\mathsf{s} \leftarrow \mathsf{s}_0$ and empty the queue $\mathsf{Q} \leftarrow []$.

- Upon receiving a message $(\texttt{Submit}, m)$ from a party $\mathcal{P}$, push m to Q. Leak $(\texttt{Sent}, \mathcal{P}, m)$ to $\mathcal{S}$ at the same round.
- Upon receiving $(\texttt{tick}, \tau = k \cdot \bar{\Delta})$ from $\mathcal{F}_{clk}$ for some $k \in \mathbb{N}^+$:
  1. Query $(\texttt{Query}, k) \hookrightarrow \mathcal{F}_{\text{RB}}$, which returns $(\texttt{BeaconValue}, k, r^k)$.
  2. Set $\mathsf{in}^k \leftarrow \mathsf{Q}$ and $\mathsf{Q} \leftarrow []$.
  3. Compute the output $\mathsf{out}^k \leftarrow f(\mathsf{s}, \mathsf{in}^k, r^k)$.
  4. Update its internal state to $\mathsf{s} \leftarrow g(\mathsf{s}, \mathsf{in}^k, r^k)$.
  5. The tuple $(\mathsf{in}^k, r^k, \mathsf{out}^k)$ is recorded as the public transcript for epoch k. Send $(\texttt{Transcript}, k, (\mathsf{in}^k, r^k, \mathsf{out}^k))$ to all parties and to $\mathcal{S}$.

Figure 22: Ideal functionality of a public blockchain in the $\{\mathcal{F}_{\text{RB}}, \mathcal{F}_{clk}\}$-hybrid world

- **Communication:** The execution proceeds in synchronous rounds. In each round, parties can send messages to each other and can send $(\texttt{Submit}, m)$ messages to the ideal functionality $\mathcal{F}_{\text{BC}}$. At intervals of $\bar{\Delta}$, $\mathcal{F}_{\text{BC}}$ delivers a new public transcript to both parties.
- **Global Setup:** At round 0, the environment $\mathcal{E}$ initializes $\mathcal{F}_{\text{BC}}$ by sending $(\texttt{Init}, \mathsf{s}_0, \Pi_B)$, and provides inputs $i_{\mathcal{P}}$ and i_Q to Π_P and Π_Q, respectively.
- **Protocol Execution:** The protocol proceeds from round $\tau = 1$ to T. At each round τ, we assume that Π_P and Π_Q execute the following steps:
  1. At the beginning of round τ, Π_P and Π_Q receive inputs for the current round. For $\tau = 1$, the input for Π_P is $i_{\mathcal{P}}$ (and symmetrically for Π_Q). For $\tau > 1$, the

The Fairness Game of P: $\text{Game}^{\text{fair}}_{\Pi_P,\mathcal{A}}(\lambda)$

Setup Phase: At round 0, $\mathcal{E}$ generates $(i_{\mathcal{P}}, \text{desc}_{\mathcal{P}})$ and (i_Q, desc_Q). $\mathcal{E}$ initializes $\mathcal{F}_{\text{BC}}$ with state machine Π_B and an empty initial transcript $\mathcal{T}^0$. $\mathcal{E}$ sends $(\texttt{init}, i_{\mathcal{P}})$ to Challenger and gives i_Q to $\mathcal{A}$.

Challenger (runs honest Π_P)

- At round 1:
  1. $\mathsf{s}_P^1 \leftarrow g_P(\mathsf{s}_P^0, i_{\mathcal{P}})$, $(m^1_{P\to Q}, m^1_{P\to B}) \leftarrow f_P(\mathsf{s}_P^1)$.
  2. Send $(\texttt{P2Q}, m^1_{P\to Q})$ to $\mathcal{A}$ and $(\texttt{Submit}, m^1_{P\to B})$ to $\mathcal{F}_{\text{BC}}$.
- For each round $\tau = 2, 3, \ldots, T$:
  1. Let $\mathsf{in}_P^\tau = (m^{\tau-1}_{Q\to P}, \mathcal{T}^{\tau-1})$.
  2. $\mathsf{s}_P^\tau \leftarrow g_P(\mathsf{s}_P^{\tau-1}, \mathsf{in}_P^\tau)$, $(m^\tau_{P\to Q}, m^\tau_{P\to B}) \leftarrow f_P(\mathsf{s}_P^\tau)$.
  3. If $\tau < T$, send $(\texttt{P2Q}, m^\tau_{P\to Q})$ to $\mathcal{A}$, $(\texttt{Submit}, m^\tau_{P\to B})$ to $\mathcal{F}_{\text{BC}}$.
- At round T, compute $o_P \leftarrow \mathsf{Ext}_P(\mathsf{s}_P^T)$ and return o_P.

Adversary $\mathcal{A}$ (controls Q)

- For each round $\tau = 1, 2, \ldots, T-1$:
  1. Receive $m^\tau_{P\to Q}$ from Challenger and $\mathcal{T}^{\tau-1}$ from $\mathcal{F}_{\text{BC}}$, learning $m^\tau_{P\to B}$ from $\mathcal{F}_{\text{BC}}$'s leakage.
  2. $(m^\tau_{Q\to P}, m^\tau_{Q\to B}) \leftarrow \mathcal{A}(\{m^j_{P\to Q}\}_{j=1}^\tau, i_Q, \{\mathcal{T}^j\}_{j=0}^{\tau-1}, \{m^j_{P\to B}\}_{j=t}^\tau)$, where $t\bar{\Delta} \le \tau < (t+1)\bar{\Delta} \wedge t \in \mathbb{N}^+$.
  3. Send $(\texttt{Q2P}, m^\tau_{Q\to P})$ to Challenger, and $(\texttt{Submit}, m^\tau_{Q\to B})$ to $\mathcal{F}_{\text{BC}}$.
- At round T, compute and return its output: $o_Q \leftarrow \mathcal{A}(\{m^j_{P\to Q}\}_{j=1}^{T-1}, i_Q, \{\mathcal{T}^j\}_{j=0}^{T-1})$.

Winning Condition: The game's output is defined as follows:

- The game outputs 1 if the unfair outcome does **not** occur: $(\text{desc}_P(o_Q) \neq 1 \wedge \text{desc}_Q(o_P) \neq 1) \vee (\text{desc}_P(o_Q) = 1 \wedge \text{desc}_Q(o_P) = 1)$.
- Otherwise, the game outputs 0 if the unfair outcome for $\mathcal{P}$ occurs.

Figure 23: The game-based definition of fairness for party $\mathcal{P}$ in the public blockchain world.

input for Π_P, denoted in_P^τ, is the tuple $(m^{\tau-1}_{Q\to P}, \mathcal{T}^{\tau-1})$, where $m^{\tau-1}_{Q\to P}$ is the message received from Π_Q in the previous round and $\mathcal{T}^{\tau-1}$ is the transcript from $\mathcal{F}_{\text{BC}}$ for the previous round.

2. Π_P and Π_Q update their internal states. For Π_P: $\mathsf{s}_P^\tau \leftarrow g_P(\mathsf{s}_P^{\tau-1}, \mathsf{in}_P^\tau)$, where g_P is a randomized state transition function. The state transition for Π_Q is symmetric.
3. Π_P and Π_Q generate messages for the current round. For Π_P: $(m^\tau_{P\to Q}, m^\tau_{P\to B}) \leftarrow f_P(\mathsf{s}_P^\tau)$, where f_P is a randomized output function.
4. Π_P sends $(\texttt{P2Q}, m^\tau_{P\to Q})$ to Π_Q and $(\texttt{Submit}, m^\tau_{P\to B})$ to $\mathcal{F}_{\text{BC}}$. Symmetrically, Π_Q sends $(\texttt{Q2P}, m^\tau_{Q\to P})$ to Π_P and $(\texttt{Submit}, m^\tau_{Q\to B})$ to $\mathcal{F}_{\text{BC}}$.
5. At a pre-defined deadline T, both parties compute their final output. Π_P computes its output by running an extraction function Ext_P on its final state s_P^T. It outputs the extracted item $(\texttt{result}, i_Q)$ or $(\texttt{result}, \perp)$ if extraction fails. The process for Π_Q is symmetric.

Fairness in the Public Blockchain World. We say that $\mathcal{P}/\mathcal{Q}$ is *p-partially fair* in the public blockchain world if, for any PPT adversary $\mathcal{A}$ controlling $Q/\mathcal{P}$, the probability that $\mathcal{A}$ obtains $i_{\mathcal{P}}/i_Q$ while Π_P/Π_Q does not obtain $i_Q/i_{\mathcal{P}}$ is at most p. Formally, given the fairness game defined in Fig. 23, we say that a protocol $\pi = (\Pi_P, \Pi_Q, \Pi_B)$ achieves p-partial fairness for party $\mathcal{P}$ if for any PPT adversary $\mathcal{A}$, the following holds:

$$\Pr[\text{Game}^{\text{fair}}_{\Pi_P,\mathcal{A}} = 0] \le p \tag{3}$$

where the probability is taken over the random coins of the environment, the challenger, and the adversary. The formal definition of fairness for party $\mathcal{Q}$ is symmetric.

B.3 FSE in the Random Beacon World

FSE Protocol in the Random Beacon World. An FSE protocol in the random beacon world is a two-party protocol $\bar{\pi} = (\bar{\Pi}_P, \bar{\Pi}_Q)$ executed in the $\{\mathcal{F}_{clk}, \mathcal{F}_{\text{RB}}\}$-hybrid model. $\bar{\Pi}_P$ and $\bar{\Pi}_Q$ are two randomized PPT ITM machines representing the two parties $\mathcal{P}$ and $\mathcal{Q}$.

- **Communication:** The execution proceeds in synchronous rounds. In each round, $\bar{\Pi}_P$ and $\bar{\Pi}_Q$ can send messages to each other directly. They also have access to a shared instance of $\mathcal{F}_{\text{RB}}$. At beginning of round $k \cdot \bar{\Delta}$ for $k \in \mathbb{N}^+$, $\mathcal{F}_{\text{RB}}$ broadcasts a new random value $(\texttt{BeaconOutput}, k, r^k)$ to both parties.
- **Global Setup:** At round 0, the environment $\mathcal{E}$ provides inputs $i_{\mathcal{P}}$ and i_Q to $\bar{\Pi}_P$ and $\bar{\Pi}_Q$, respectively, and instantiates $\mathcal{F}_{\text{RB}}$ with round duration $\bar{\Delta}$.
- **Protocol Execution:** The protocol proceeds from round $\tau = 1$ to T. At each round τ, $\bar{\Pi}_P$ and $\bar{\Pi}_Q$ execute the following steps:
  1. At the beginning of round τ, $\bar{\Pi}_P$ and $\bar{\Pi}_Q$ receive inputs $(\mathsf{in}_P^\tau / \mathsf{in}_Q^\tau)$ for the current round. For $\tau = 1$, the input for $\bar{\Pi}_P$ is $i_{\mathcal{P}}$. For $\tau > 1$, the input for $\bar{\Pi}_P$, denoted in_P^τ, is the

The Fairness Game of P in the Random Beacon World: $\text{Game}^{\text{fair}}_{\bar{\Pi}_P,\bar{\mathcal{A}}}(\lambda)$

Setup Phase: At round 0, $\mathcal{E}$ generates $(i_{\mathcal{P}}, \text{desc}_{\mathcal{P}})$ and (i_Q, desc_Q). $\mathcal{E}$ instantiates $\mathcal{F}_{\text{RB}}$ with round duration $\bar{\Delta}$. $\mathcal{E}$ sends $(\texttt{init}, i_{\mathcal{P}})$ to Challenger and gives i_Q to $\bar{\mathcal{A}}$.

Challenger (runs honest $\bar{\Pi}_P$)

- At round 1:
  1. $\bar{\mathsf{s}}^1_P \leftarrow \bar{g}_P(\bar{\mathsf{s}}^0_P, i_{\mathcal{P}})$, $m^1_{P\to Q} \leftarrow \bar{f}_P(\bar{\mathsf{s}}^1_P)$.
  2. Send $(\texttt{P2Q}, m^1_{P\to Q})$ to $\bar{\mathcal{A}}$.
- For each round $\tau = 2,3,\ldots,T$:
  1. Let $\mathsf{in}^\tau_P = m^{\tau-1}_{Q\to P}$. If round τ was a beacon round, in^τ_P also includes $(\texttt{BeaconOutput}, k, r^k)$ from $\mathcal{F}_{\text{RB}}$.
  2. $\bar{\mathsf{s}}^\tau_P \leftarrow \bar{g}_P(\bar{\mathsf{s}}^{\tau-1}_P, \mathsf{in}^\tau_P)$, $m^\tau_{P\to Q} \leftarrow \bar{f}_P(\bar{\mathsf{s}}^\tau_P)$.
  3. If $\tau < T$, send $(\texttt{P2Q}, m^\tau_{P\to Q})$ to $\bar{\mathcal{A}}$.
- At round T, compute $o_P \leftarrow \bar{\mathsf{Ext}}_P(\bar{\mathsf{s}}^T_P)$ and return o_P.

Adversary $\bar{\mathcal{A}}$ (controls Q)

- For each round $\tau = 1,2,\ldots,T-1$:
  1. Receive $m^\tau_{P\to Q}$ from Challenger. If round τ is a beacon round, receive $(\texttt{BeaconOutput}, k, r^k)$ from $\mathcal{F}_{\text{RB}}$.
  2. $m^\tau_{Q\to P} \leftarrow \bar{\mathcal{A}}(\{m^j_{P\to Q}\}^\tau_{j=1}, i_Q, \{\text{beacon outputs}\}^{\tau-1})$.
  3. Send $(\texttt{Q2P}, m^\tau_{Q\to P})$ to Challenger.
- At round T, compute and return its output: $o_Q \leftarrow \bar{\mathcal{A}}(\{m^j_{P\to Q}\}^{T-1}_{j=1}, i_Q, \{\text{beacon outputs}\}^{T-1})$.

Winning Condition: The game's output is defined as follows:

- The game outputs 1 if the unfair outcome does **not** occur: $(\text{desc}_P(o_Q) \neq 1 \wedge \text{desc}_Q(o_P) \neq 1) \vee (\text{desc}_P(o_Q) = 1 \wedge \text{desc}_Q(o_P) = 1)$.
- Otherwise, the game outputs 0 if the unfair outcome for $\mathcal{P}$ occurs: $(\text{desc}_P(o_Q) = 1 \wedge \text{desc}_Q(o_P) \neq 1)$.

Figure 24: Fairness for party $\mathcal{P}$ in the Random Beacon World.

message $m^{\tau-1}_{Q\to P}$ received from $\bar{\Pi}_Q$ in the previous round. If round $\tau-1$ was a beacon round (i.e., $(\tau-1)\cdot\bar{\Delta} = k$ for some integer k), the input also includes the beacon output $(\texttt{BeaconOutput}, k, r^k)$.

2. $\bar{\Pi}_P$ and $\bar{\Pi}_Q$ update their internal states. For $\bar{\Pi}_P$: $\bar{\mathsf{s}}^\tau_P \leftarrow \bar{g}_P(\bar{\mathsf{s}}^{\tau-1}_P, \mathsf{in}^\tau_P)$, where $\bar{g}_P$ is a randomized state transition function. The state transition for $\bar{\Pi}_Q$ is symmetric.
3. $\bar{\Pi}_P$ and $\bar{\Pi}_Q$ generate messages for the current round. For $\bar{\Pi}_P$: $m^\tau_{P\to Q} \leftarrow \bar{f}_P(\bar{\mathsf{s}}^\tau_P)$, where $\bar{f}_P$ is a randomized output function.
4. $\bar{\Pi}_P$ sends $(\texttt{P2Q}, m^\tau_{P\to Q})$ to $\bar{\Pi}_Q$. Symmetrically, $\bar{\Pi}_Q$ sends $(\texttt{Q2P}, m^\tau_{Q\to P})$ to $\bar{\Pi}_P$.
5. At the pre-defined deadline T, both parties compute their final output. $\bar{\Pi}_P$ computes its output by running an extraction function $\bar{\mathsf{Ext}}_P$ on its final state $\bar{\mathsf{s}}^T_P$. It outputs the extracted item i_Q or $\perp$ if extraction fails. The process for $\bar{\Pi}_Q$ is symmetric.

Fairness of P in the Random Beacon World. We say that $\mathcal{P}$ is *p-partially fair* in the random beacon world if, for any PPT adversary $\bar{\mathcal{A}}$ controlling Q, the probability that $\bar{\mathcal{A}}$ obtains $i_{\mathcal{P}}$ while $\bar{\Pi}_P$ does not obtain i_Q is at most p. Formally, given the fairness game defined in Fig. 24, we say that a protocol $\bar{\pi} = (\bar{\Pi}_P, \bar{\Pi}_Q)$ achieves p-partial fairness for party $\mathcal{P}$ if for any PPT adversary $\bar{\mathcal{A}}$, the following holds:

$$\Pr[\text{Game}^{\text{fair}}_{\bar{\Pi}_P,\bar{\mathcal{A}}} = 0] \leq p \tag{4}$$

where the probability is taken over the random coins of the environment, the challenger, and the adversary.

$q-$Effectiveness. In the random beacon world, $q-$Effectiveness is indential to the public blockchain world, where given both parties are honest, at T, $\bar{\Pi}_P$ returns i_Q such that $\text{desc}_Q(i_Q) = 1$, and $\bar{\Pi}_Q$ returns $i_{\mathcal{P}}$ such that $\text{desc}_P(i_{\mathcal{P}}) = 1$, with a probability higher than q, $q \in [0,1]$.

B.4 Proof of Theorem 1

Proof. We prove this theorem by constructing a protocol $\bar{\pi} = (\bar{\Pi}_P, \bar{\Pi}_Q)$ for the $\mathcal{F}_{\text{RB}}$-hybrid world that simulates the execution of $\pi = (\Pi_P, \Pi_Q)$ in the $\mathcal{F}_{\text{BC}}$-hybrid world, as illustrated in Fig. 9. The construction of $\bar{\pi}$ remains as defined in Figures 25 and 26. We focus on proving that the security properties are preserved.

Fairness. We prove the fairness of the constructed protocol $\bar{\pi}$ in the $\mathcal{F}_{\text{RB}}$-hybrid world by reduction. Assume for contradiction that there exists a PPT adversary $\bar{\mathcal{A}}$ that breaks the p-partial fairness of $\bar{\pi}$ for party $\mathcal{P}$ in the random beacon world. That is, in the game $\text{Game}^{\text{fair}}_{\bar{\Pi}_P,\bar{\mathcal{A}}}$ (defined in Figure 24), the adversary wins with probability $p' > p$.

$$\Pr[\text{Game}^{\text{fair}}_{\bar{\Pi}_P,\bar{\mathcal{A}}} = 0] = p' > p$$

We construct a PPT adversary $\mathcal{A}$ that uses $\bar{\mathcal{A}}$ as a black-box subroutine to break the p-partial fairness of the original protocol π in the $\mathcal{F}_{\text{BC}}$-hybrid world with the same probability p'.

Blockchain Simulator Sub-protocol $\bar{\Pi}_B^i$, $i \in \{\mathcal{P}, \mathcal{Q}\}$

Parameters: a round duration $\bar{\Delta}$, a deterministic state-transition machine $\Pi_B = (g, f)$ with an initial state s_0, and the FIFO queue Q.
Setup: Upon $(\texttt{Init}, \mathsf{s}_0, \Pi_B)$ from $\mathcal{E}$, initialize an internal state $\mathsf{s} \leftarrow \mathsf{s}_0$. Q: the FIFO queue of pending messages, initialized to empty.

- Upon receiving a message $(\texttt{Submit}, m)$ from a party $\mathcal{P}_i$, push m in the queue Q.
- Upon receiving $(\texttt{tick}, \tau = k \cdot \bar{\Delta})$ from $\mathcal{F}_{clk}$ for some $k \in \mathbb{N}^+$:
  1. Query $(\texttt{Query}, k) \hookrightarrow \mathcal{F}_{\text{RB}}$, which returns $(\texttt{BeaconValue}, k, r^k)$.
  2. Set $\mathsf{in}^k \leftarrow \mathsf{Q}$, empty the queue $\mathsf{Q} \leftarrow []$.
  3. Update its internal state to $\mathsf{s} \leftarrow g(\mathsf{s}, \mathsf{in}^k, r^k)$.
  4. Compute the output $\mathsf{out}^k = f(\mathsf{s}, \mathsf{in}^k, r^k)$.
  5. Return $(\texttt{Transcript}, k, (\mathsf{in}^k, r^k, \mathsf{out}^k))$.

Figure 25: Blockchain Simulator sub-protocol executed by $\mathcal{P}$.

This contradicts the assumed fairness of π. The construction of $\mathcal{A}$ is detailed in Figure 27.

Analysis of the Reduction. We argue that the view of $\bar{\mathcal{A}}$ when run as a subroutine inside $\mathcal{A}$ is identical to its view in a real execution of $\text{Game}^{\text{fair}}_{\bar{\Pi}_P, \bar{\mathcal{A}}}$ in the $\mathcal{F}_{\text{RB}}$-hybrid world.

- **Challenger's Messages:** In the real $\mathcal{F}_{\text{RB}}$-world game, the honest challenger $\bar{\Pi}_P$ runs an internal copy of Π_P. When Π_P outputs $(m^\tau_{P\to Q}, m^\tau_{P\to B})$, the construction of $\bar{\Pi}_P$ (Figure 26) dictates that it sends two messages to $\bar{\Pi}_Q$ (controlled by $\bar{\mathcal{A}}$): $(\texttt{P2Q}, (\texttt{P}, m^\tau_{P\to Q}))$ and $(\texttt{P2Q}, (\texttt{B}, m^\tau_{P\to B}))$. In our reduction, $\mathcal{A}$ receives $m^\tau_{P\to Q}$ from its challenger and $m^\tau_{P\to B}$ from $\mathcal{F}_{\text{BC}}$'s leakage. Our revised construction of $\mathcal{A}$ explicitly forwards both of these messages to the internal $\bar{\mathcal{A}}$, tagged appropriately as $(\texttt{P}, \cdot)$ and $(\texttt{B}, \cdot)$. Thus, the messages $\bar{\mathcal{A}}$ receives from the challenger are identically distributed in both the real game and the simulation.
- **Beacon/Transcript Simulation:** In the real $\mathcal{F}_{\text{RB}}$-world game, both parties receive beacon outputs $(\texttt{BeaconOutput}, k, r^k)$ directly from $\mathcal{F}_{\text{RB}}$. In our reduction, $\mathcal{A}$ receives the transcript $\mathcal{T}_k = (\mathsf{in}^k, r^k, \mathsf{out}^k)$ from $\mathcal{F}_{\text{BC}}$. Since $\mathcal{F}_{\text{BC}}$ itself is defined in the $\{\mathcal{F}_{\text{RB}}, \mathcal{F}_{clk}\}$-hybrid world, the randomness r^k it uses is the same randomness from the underlying $\mathcal{F}_{\text{RB}}$. $\mathcal{A}$ extracts this r^k and forwards it to $\bar{\mathcal{A}}$. Therefore, the sequence of beacon values seen by $\bar{\mathcal{A}}$ is identical in both scenarios.

Since the view of $\bar{\mathcal{A}}$ is identical in both the real $\mathcal{F}_{\text{RB}}$-world game and our simulated game, its behavior, and in particular its final output o_Q, will be identical. Consequently, if $\bar{\mathcal{A}}$ wins its game, then $\mathcal{A}$ also wins its game. The winning condition for both games depends on the outputs o_P and o_Q. The output o_Q is identical by construction. The challenger's output o_P is computed by the internal machine Π_P. The execution trace of Π_P, and thus its output o_P, depends on the sequence of inputs

$\bar{\Pi}_P$

Parameters: a round duration $\bar{\Delta}$. Two sub-protocol $\bar{\Pi}_B^P$ and Π_P.
Setup: At round 0, upon receiving input $(\texttt{Init}, i_{\mathcal{P}}, \mathsf{s}_0, \Pi_B)$ from the environment $\mathcal{E}$, $\bar{\Pi}_P$ initializes its state as described above and provides i_P to its internal instance of Π_P. Send $(\texttt{Init}, i_{\mathcal{P}})$ to Π_P. Send $(\texttt{Init}, \mathsf{s}_0, \Pi_B)$ to $\bar{\Pi}_B^P$. At each round $\tau = 1, 2, \ldots, T$, $\bar{\Pi}_P$:

- Handle Internal outputs:
  - Upon outputing $(\texttt{P2Q}, \bar{m}^\tau_{P\to Q})$ from Π_P, send $(\texttt{P2Q}, m)$ to $\bar{\Pi}_Q$.
  - Upon outputing $(\texttt{Submit}, \bar{m}^\tau_{P\to B})$ from Π_P, send $(\texttt{Submit}, \bar{m}^\tau_{P\to B})$ to $\bar{\Pi}_B^P$, and $(\texttt{P2Q}, (\texttt{B}, \bar{m}^\tau_{P\to B}))$ to $\bar{\Pi}_Q$.
  - Upon outputing $(\texttt{Transcript}, k, \mathcal{T}_k)$ from $\bar{\Pi}_B^P$, send $(\texttt{Transcript}, k, \mathcal{T}_k)$ to Π_P at the end of round $k \cdot \bar{\Delta}$.
  - Upon returning $(\texttt{result}, o_Q)$, return the same output to $\mathcal{E}$.
- Handle Messages from $\bar{\Pi}_Q$, Upon $(\texttt{P2Q}, m) \overset{\tau}{\hookleftarrow} \bar{\Pi}_Q$
  - If $m = (\texttt{B}, m')$, append m', send $(\texttt{Submit}, m')$ to $\bar{\Pi}_B^P$.
  - If $m = (\texttt{P}, m')$, forward $(\texttt{Send}, m')$ to Π_P.
  - Throw this message if m is not of the form $(\texttt{P}, m')$ or $(\texttt{B}, m')$.

Figure 26: Construction of $\bar{\Pi}_P$ in the $\mathcal{F}_{\text{RB}}$-hybrid world.

it receives: $(\mathsf{in}^\tau_P)^T_{\tau=1}$, where $\mathsf{in}^\tau_P = (m^{\tau-1}_{Q\to P}, \mathcal{T}_{\tau-1})$. We argue that this sequence of inputs is identical in both the reduction and the real game.

- **Peer-to-Peer Messages ($m^{\tau-1}_{Q\to P}$):** In the reduction, $\bar{\mathcal{A}}$ produces a tagged message $(\texttt{P}, m')$. Our adversary $\mathcal{A}$ extracts m' and relays it to its challenger, Π_P. In the real $\mathcal{F}_{\text{RB}}$-world game, $\bar{\mathcal{A}}$ sends the same tagged message $(\texttt{P}, m')$ to the honest challenger $\bar{\Pi}_P$, which, by its construction (Figure 26), also extracts and forwards m' to its internal Π_P instance. Thus, the sequence of peer-to-peer messages received by Π_P is identical in both settings.
- **Transcripts ($\mathcal{T}_{\tau-1}$):** The transcript $\mathcal{T}_k = (\mathsf{in}^k, r^k, \mathsf{out}^k)$ is generated deterministically from the set of submitted messages in^k and the beacon randomness r^k.
  - The beacon randomness r^k is identical in both worlds, as established previously.
  - We now show that the set of submitted messages in^k is also identical. In any given epoch, messages are submitted to the blockchain by the challenger (Π_P) and the adversary ($\bar{\mathcal{A}}$).
    - *Challenger's messages:* In the real $\mathcal{F}_{\text{RB}}$-game, Π_P outputs $m^\tau_{P\to B}$, which $\bar{\Pi}_P$ submits to its local simulator $\bar{\Pi}_B^P$. In the reduction, Π_P outputs $m^\tau_{P\to B}$ and submits it to $\mathcal{F}_{\text{BC}}$.
    - *Adversary's messages:* In the real $\mathcal{F}_{\text{RB}}$-game, $\bar{\mathcal{A}}$ outputs $(\texttt{B}, m')$, which $\bar{\Pi}_P$ receives and submits to its local simulator $\bar{\Pi}_B^P$. In the reduction, $\bar{\mathcal{A}}$ outputs the same $(\texttt{B}, m')$, which $\mathcal{A}$ relays to $\mathcal{F}_{\text{BC}}$.

Construction of Adversary $\mathcal{A}$

$\mathcal{A}$ internally runs a copy of $\bar{\mathcal{A}}$ and simulates its environment. $\mathcal{A}$ interacts with the challenger (running Π_P) and the functionality $\mathcal{F}_{\text{BC}}$ in the $\mathcal{F}_{\text{BC}}$-hybrid world game.

- **Setup:**
  1. $\mathcal{A}$ receives i_Q from the environment $\mathcal{E}$.
  2. $\mathcal{A}$ initializes an internal instance of $\bar{\mathcal{A}}$, forwarding i_Q to it.
- **Simulation Loop (for each round $\tau = 1, \ldots, T-1$):**
  1. **Simulating Challenger's Messages to $\bar{\mathcal{A}}$:**
     - The challenger (running Π_P) sends the peer-to-peer message $(\texttt{P2Q}, m^{\tau}_{P\to Q})$ to $\mathcal{A}$.
     - The challenger sends $(\texttt{Submit}, m^{\tau}_{P\to B})$ to $\mathcal{F}_{\text{BC}}$. $\mathcal{F}_{\text{BC}}$ leaks this message to $\mathcal{A}$.
     - $\mathcal{A}$ simulates the messages from the honest challenger $\bar{\Pi}_P$ to $\bar{\mathcal{A}}$ as follows:
       - It forwards the peer-to-peer message by sending $(\texttt{P2Q}, (\texttt{P}, m^{\tau}_{P\to Q}))$ to its internal $\bar{\mathcal{A}}$.
       - It forwards the challenger's blockchain message by sending $(\texttt{P2Q}, (\texttt{B}, m^{\tau}_{P\to B}))$ to its internal $\bar{\mathcal{A}}$.
  2. **Simulating Beacon/Transcript:**
     - At the end of each epoch $k \cdot \bar{\Delta}$, $\mathcal{A}$ receives the transcript $(\texttt{Transcript}, k, \mathcal{T}_k)$ from $\mathcal{F}_{\text{BC}}$.
     - $\mathcal{A}$ forwards the beacon output $(\texttt{BeaconOutput}, k, r^k)$ contained within $\mathcal{T}_k$ to its internal $\bar{\mathcal{A}}$.
  3. **Receiving and Relaying $\bar{\mathcal{A}}$'s Messages:**
     - $\bar{\mathcal{A}}$ outputs a message $m^{\tau}_{Q\to P}$ intended for $\bar{\Pi}_P$. According to the construction of $\bar{\Pi}_Q$, this message is a tagged tuple.
     - If the message from $\bar{\mathcal{A}}$ is of the form $(\texttt{P}, m')$, $\mathcal{A}$ sends $(\texttt{Q2P}, m')$ to its challenger (the honest Π_P).
     - If the message is of the form $(\texttt{B}, m')$, $\mathcal{A}$ sends $(\texttt{Submit}, m')$ to the $\mathcal{F}_{\text{BC}}$ functionality.
- **Output:** At round T, $\bar{\mathcal{A}}$ computes its output o_Q. $\mathcal{A}$ adopts this output and returns o_Q as its own.

Figure 27: Revised construction of adversary $\mathcal{A}$ using $\bar{\mathcal{A}}$.

Therefore, the set of messages in^k provided to the transcript generator is identical in both scenarios.

Since both the local simulator $\bar{\Pi}^P_B$ and the ideal functionality $\mathcal{F}_{\text{BC}}$ are deterministic state machines that receive the exact same sequence of inputs (submitted messages and beacon randomness), they will compute and output identical transcripts $\mathcal{T}_k$ in every epoch.

Because the internal Π_P instance receives an identical sequence of inputs (both peer-to-peer messages and transcripts) in both the reduction and the real game, its execution trace and final output o_P will also be identical.

This implies that the event "$\text{Game}^{\text{fair}}_{\Pi_P,\mathcal{A}} = 0$" occurs if and only if the event "$\text{Game}^{\text{fair}}_{\bar{\Pi}_P,\bar{\mathcal{A}}} = 0$" occurs. Therefore, their probabilities must be equal:

$$\Pr[\text{Game}^{\text{fair}}_{\Pi_P,\mathcal{A}} = 0] = \Pr[\text{Game}^{\text{fair}}_{\bar{\Pi}_P,\bar{\mathcal{A}}} = 0] = p'$$

By our initial assumption, $p' > p$. This means we have constructed an adversary $\mathcal{A}$ that breaks the p-partial fairness of protocol π in the $\mathcal{F}_{\text{BC}}$-hybrid world. This is a contradiction to the assumed fairness of π. Therefore, no such adversary $\bar{\mathcal{A}}$ can exist, and the protocol $\bar{\pi}$ must be p-partially fair in the $\mathcal{F}_{\text{RB}}$-hybrid world.

Effectiveness. We prove that if protocol π is q-effective in the $\mathcal{F}_{\text{BC}}$-hybrid world, then the constructed protocol $\bar{\pi}$ is q-effective in the $\mathcal{F}_{\text{RB}}$-hybrid world. The proof relies on showing that an honest execution of $\bar{\pi}$ in the $\mathcal{F}_{\text{RB}}$-hybrid world perfectly simulates an honest execution of π in the $\mathcal{F}_{\text{BC}}$-hybrid world.

Consider an honest execution of $\bar{\pi} = (\bar{\Pi}_P, \bar{\Pi}_Q)$ in the $\mathcal{F}_{\text{RB}}$-hybrid world. Both parties, $\bar{\Pi}_P$ and $\bar{\Pi}_Q$, internally run instances of Π_P and Π_Q, and local blockchain simulators $\bar{\Pi}^P_B$ and $\bar{\Pi}^Q_B$, respectively, as specified in Figures 25 and 26.

Let's analyze the inputs received by the internal machines Π_P and Π_Q during the execution of $\bar{\pi}$. The input for Π_P at round $\tau > 1$ is the tuple $\mathsf{in}^{\tau}_P = (m^{\tau-1}_{Q\to P}, \mathcal{T}^{\tau-1})$.

- **Peer-to-Peer Messages ($m^{\tau-1}_{Q\to P}$):** In an honest execution of $\bar{\pi}$, the internal Π_Q of $\bar{\Pi}_Q$ generates $(m^{\tau-1}_{Q\to P}, m^{\tau-1}_{Q\to B})$. According to the construction in Figure 26, $\bar{\Pi}_Q$ sends $(\texttt{P2Q}, (\texttt{P}, m^{\tau-1}_{Q\to P}))$ to $\bar{\Pi}_P$. Upon receiving this, $\bar{\Pi}_P$ extracts $m^{\tau-1}_{Q\to P}$ and forwards it to its internal Π_P. This perfectly mirrors the direct peer-to-peer communication channel in the $\mathcal{F}_{\text{BC}}$-hybrid world.
- **Transcripts ($\mathcal{T}^{\tau-1}$):** The transcript $\mathcal{T}_k$ for epoch k is generated by the local blockchain simulators, $\bar{\Pi}^P_B$ and $\bar{\Pi}^Q_B$. The transcript is a deterministic function of the submitted messages within the epoch and the beacon randomness, i.e., $\mathcal{T}_k = (\mathsf{in}^k, r^k, \mathsf{out}^k)$.
  - *Beacon Randomness (r^k):* Both simulators $\bar{\Pi}^P_B$ and $\bar{\Pi}^Q_B$ are connected to the same ideal functionality $\mathcal{F}_{\text{RB}}$. At each epoch k, they both receive the identical beacon value r^k from $\mathcal{F}_{\text{RB}}$.
  - *Submitted Messages (in^k):* In an honest execution, for any round τ within epoch k, Π_P generates $m^{\tau}_{P\to B}$ and Π_Q generates $m^{\tau}_{Q\to B}$. According to the construction, $\bar{\Pi}_P$ submits $m^{\tau}_{P\to B}$ to its simulator $\bar{\Pi}^P_B$ and also sends it to $\bar{\Pi}_Q$, which then submits it to its simulator $\bar{\Pi}^Q_B$. Symmetrically, $\bar{\Pi}_Q$ submits $m^{\tau}_{Q\to B}$ to $\bar{\Pi}^Q_B$ and sends it to $\bar{\Pi}_P$, which submits it to $\bar{\Pi}^P_B$. Since communication is reliable in the ideal model, both simulators $\bar{\Pi}^P_B$ and $\bar{\Pi}^Q_B$ receive the exact same set of messages $\{m^{\tau}_{P\to B}, m^{\tau}_{Q\to B}\}$ for all rounds τ in epoch k.

Since both simulators are deterministic state machines that start in the same initial state and receive the exact same sequence of inputs (submitted messages and beacon randomness), they will compute and output identical

transcripts $\mathcal{T}_k$ in every epoch. This transcript is identical to the one that would be produced by the ideal functionality $\mathcal{F}_{\text{BC}}$ in an honest execution of π.

The execution trace of the internal machines (Π_P, Π_Q) within $\bar{\pi}$ is therefore identically distributed to an execution trace of (Π_P, Π_Q) in the $\mathcal{F}_{\text{BC}}$-hybrid world. By the q-effectiveness of π, the probability that Π_P and Π_Q output the correct items is at least q. Since the final outputs of $\bar{\Pi}_P$ and $\bar{\Pi}_Q$ are the outputs of their internal Π_P and Π_Q instances, $\bar{\pi}$ is also q-effective.

T-Timeliness. The T-timeliness property of π states that honest parties terminate by a predefined round T. The construction of $\bar{\pi}$ in Figure 26 explicitly defines the protocol execution for rounds $\tau = 1, 2, \ldots, T$. At round T, the parties compute and return their final outputs. The construction does not add any communication rounds or blocking operations that would extend the protocol's duration beyond T. Therefore, if π terminates in T rounds, the wrapper protocol $\bar{\pi}$ also terminates in T rounds. The T-timeliness property is preserved. □

C Extended Preliminaries of *Zeus*

In the GUC framework, a protocol Π is executed between a set of parties $\mathcal{P}$ with an adversary $\mathcal{A}$, who receives as input a security parameter $\lambda \in \mathbb{N}$ along with an auxiliary input $z \in \{0,1\}^*$. In our work, we follow a static corrupt model, where $\mathcal{A}$ can corrupt any party $P_i \in \mathcal{P}$ at the beginning of the protocol execution. Corrupting a party P_i means that $\mathcal{A}$ takes control over P_i and learns all its internal state(e.g., credential, private key). The parties and the adversary $\mathcal{A}$ take their input from the environment $\mathcal{E}$, while $\mathcal{E}$ observes the messages that are output by the parties of the protocol.

C.1 Preliminary ideal functionalities

In our model, we assume a synchronous communication, formalized by a global clock idea functionality [25], in which all parties have access to the current round number. We assume the existence of synchronous and private communication channels $\mathcal{F}_{smt}$ [25]. In $\mathcal{F}_{smt}$, a party P can privately send a message to another party Q with guaranteed delivery after precisely one round. We denote $m \overset{\tau}{\hookrightarrow} Q$ to represent that some party sends a message m to Q at round τ. $Q \overset{\tau+1}{\longleftarrow} m$ represents that Q receives the message m at round $\tau + 1$.

A global digital signature ideal functionality $\mathcal{F}_{Sig}$ [10] has a tuple of interfaces $(\text{KeyGen}, \text{Sign}, \text{Vrfy})$ satisfying EUF-CMA security. Specifically

- $(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)$: On input a security parameter λ, this PPT algorithm generates a public/secret key pair (pk, sk).
- $\sigma \leftarrow \text{Sign}(sk, m)$: On input a secret key sk and a message m, this algorithm generates a signature σ.
- $\{0,1\} \leftarrow \text{Vrfy}(pk, m, \sigma)$: On input a public key pk, a message m, and a signature σ, this algorithm outputs 1 if the signature is valid, and 0 otherwise.

C.2 Ideal functionalities of ledger and a PCN

Ledger. We use the global ideal functionality $\mathcal{G}_{\text{L}}$ [4] to model a UTXO based blockchain. $\mathcal{G}_{Ledger}$ communicates with a fixed set of parties $\mathcal{P}$. The environment $\mathcal{E}$ first initializes $\mathcal{G}_{Ledger}$ by setting up a key pair $(\text{sk}_P, \text{pk}_P)$ for every party $P \in \mathcal{P}$ and registers it to the ledger by sending $(\texttt{sid}, \texttt{REGISTER}, \text{pk}_P)$ to $\mathcal{G}_{Ledger}$. Any party $P \in \mathcal{P}$ can always post a transaction $\overline{tx}$ on $\mathcal{L}$ via $(\texttt{sid}, \texttt{POST}, \overline{tx})$. If a transaction is valid, it will be confirmed on $\mathcal{G}_{\text{L}}$ after Δ block. $\mathcal{G}_{\text{L}}$ maintains a list of blocks, while each block contains a list of transactions. A transaction is valid at a block with height h, only if all its inputs are UTXOs from block 0 to block h, the witnesses are valid, and the input amounts are greater than the output amounts.

$\mathcal{G}_{\text{L}}$

Parameters:
- Δ: the transaction confirmation blocks. Each block is μ rounds.
- $\mathcal{F}_{Sig}$: a digital signature ideal functionality.

Local vairables:
- PKI: a public key infrastructure mapping public keys to users.
- Blocks: a set of valid block. Each block contains a list of transactions TXs. h is the latest index of the blocks.

API
- $(\texttt{sid}, \texttt{REGISTER}, \text{pk}_P) \overset{\tau}{\hookleftarrow} P$: adds an entry $(\text{PKI}[\text{pk}_P] := P)$ to PKI table.
- $(\texttt{sid}, \texttt{POST}, tx) \overset{\tau}{\hookleftarrow} P$: Leak tx to $\mathcal{S}$. Use h_c to denotes the latest block height before round τ. After Δ block time, if tx is valid at block height $h_c + 1$, add tx to TXs at block height $h_c + 1$.

Figure 28: Ideal functionality of UTXO Ledger.

PCN. In this work, we model a payment channel network (PCN) by global ideal functionality $\mathcal{F}_{Channels}$ (Fig. 29) based on the standard payment channel ideal functionality widely adopted in $\mathcal{F}_{Channels}$ [3, 4, 6, 7]. Compare with the standard payment channel ideal functionality, We omit the create interface, and assume all channels required in a multi-hop payment is pre-established. For each channel $\bar{\gamma}$, $\mathcal{F}_{Channels}$ provides standard UPDATE interfaces, where if both the channel owners $\bar{\gamma}.lu/ru$ agree, can update the channel state to a new state represented by the output of a update transaction tx_{state}. The CLOSE interface allows any channel owner to close the channel peacefully or forcefully.

C.3 UC Security Definition

We adopt the Global Universal Composability (GUC) framework. Our construction operates in a

$\mathcal{F}_{Channels}$

Local vairables:

- t_{close}: the upper bound for the time to close the channel.
- t_{update}: the upper bound for the time to update the channel.

API

- $(\mathtt{sid}, \mathtt{UPDATE}, \mathrm{id}, tx_{state}) \overset{\tau}{\hookleftarrow} P$: Let $\bar{\gamma}$ be the channel where $\bar{\gamma}.\mathrm{id} = \mathrm{id}$. When invoked by $P \in \bar{\gamma}.luor\bar{\gamma}.ru$ and both parties agree, the channel $\bar{\gamma}$ is updated to the new state represented by the outputs of tx_{state}. $(\mathtt{sid}, \mathrm{UPDATED}, \mathrm{id}, tx_{state}) \xhookrightarrow{[\tau,\tau+t_{update}]} \bar{\gamma}.\mathrm{users}$ is output.
- $(\mathtt{sid}, \mathtt{CLOSE}, \mathrm{id}) \overset{\tau}{\hookleftarrow} P$: close the channel $\bar{\gamma}$, where $\bar{\gamma}.\mathrm{id} = \mathrm{id}$, either peacefully or forcefully. In round $[\tau, \tau + t_{close}]$, a transaction $\mathrm{tx}_{\mathrm{close}}$ with the current state $\bar{\gamma}.\mathrm{state}$ as output ($\mathrm{tx}_{\mathrm{close}}.\mathrm{Outs} := \bar{\gamma}.\mathrm{state}$) appears on $\mathcal{G}_{\mathrm{L}}$. Then $\mathcal{F}_{Channels}$ boardcast $(\mathtt{sid}, \mathtt{CLOSED}, \mathrm{id}, \mathrm{tx}_{\mathrm{close}})$.
- $(\mathtt{sid}, \mathtt{UPDATESPLICING}, \mathrm{id}, tx_{splicing}) \overset{\tau}{\hookleftarrow} P$: Let $\bar{\gamma}$ be the channel where $\bar{\gamma}.\mathrm{id} = \mathrm{id}$ from either channel owner. When invoked by $P = \bar{\gamma}.lu/ru$ and both parties agree, $\bar{\gamma}$ is updated to the new state represented by the outputs of $tx_{splicing}$. $tx_{splicing}$ contains *splicing outputs* $(\hat{\theta}_0, \ldots)$ and *funding outputs* $(\bar{\theta}_0, \ldots)$. $\bar{\gamma}.\mathrm{state} := \{\hat{\theta}_0, \ldots, \bar{\theta}_0, \ldots\}$, $\mathrm{UPDATED\text{-}SPLICING}, \mathrm{id}, tx_{splicing}) \xhookrightarrow{[\tau,\tau+t_{update}]} \bar{\gamma}.lu/ru$ is output.
- $(\mathtt{sid}, \mathtt{SPLICINGCHANNEL}, \mathrm{id}) \overset{\tau}{\hookleftarrow} P$: For $\bar{\gamma}.\mathrm{id} = \mathrm{id}$ and $P = \bar{\gamma}.lu/ru$. In round $\tau' \in [\tau, \tau + t_{close}]$, a transaction $tx_{spliced}$ containing all outputs within $\bar{\gamma}.\mathrm{state}$ appears on $\mathcal{G}_{\mathrm{L}}$. Meanwhile, if the latest state contains *funding outputs*, $\mathcal{G}_{\mathrm{L}}$ will update the channel state: $\bar{\gamma}.\mathrm{state} := \{\bar{\theta}_0, \ldots\}$. $(\mathtt{sid}, \mathtt{SPLICED}, \mathrm{id}, tx_{spliced}) \overset{\tau'}{\hookrightarrow} \bar{\gamma}.lu/ru$ is output.

Figure 29: Ideal functionality of payment channel.

hybrid model where all parties have access to a set of preliminary ideal functionalities, denoted by $\mathcal{F}_{\mathtt{prelim}} = \{\mathcal{F}_{clk}, \mathcal{F}_{smt}, \mathcal{F}_{Onion}, \mathcal{F}_{Sig}, \mathcal{F}_{\mathrm{Commit}}, \mathcal{G}_{\mathrm{L}}, \mathcal{F}_{Channels}\}$.

In the *real world*, a protocol Π interacts with a real-world adversary $\mathcal{A}$ and an environment $\mathcal{E}$ in the presence of the functionalities in $\mathcal{F}_{\mathtt{prelim}}$. The environment $\mathcal{E}$, upon receiving a security parameter λ and an auxiliary input z, captures the global output of this execution, denoted as $\mathrm{EXEC}^{\mathcal{F}_{\mathtt{prelim}}}_{\Pi,\mathcal{A},\mathcal{E}}(\lambda, z)$.

In the *ideal world*, an ideal functionality $\mathcal{F}$ interacts with a simulator $\mathcal{S}$ and the environment $\mathcal{E}$, also in the presence of $\mathcal{F}_{\mathtt{prelim}}$. The parties are replaced by dummy parties that forward their inputs to $\mathcal{F}$. The simulator $\mathcal{S}$ aims to replicate the behavior of the real-world adversary. The global output of this execution is denoted as $\mathrm{EXEC}^{\mathcal{F}_{\mathtt{prelim}}}_{\mathcal{F},\mathcal{S},\mathcal{E}}(\lambda, z)$.

Definition 8 (GUC Security)**.** *A protocol Π GUC-realizes an ideal functionality $\mathcal{F}$ with respect to $\mathcal{F}_{prelim}$ if for every PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that for all PPT environments $\mathcal{E}$, the following two probability ensembles are computationally indistinguishable:*

$$\{EXEC^{\mathcal{F}_{prelim}}_{\Pi,\mathcal{A},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \approx_c \{EXEC^{\mathcal{F}_{prelim}}_{\mathcal{F},\mathcal{S},\mathcal{E}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1}$$

where $\approx_c$ denotes computational indistinguishability.

D Formal Protocol Description

D.1 Main protocol of Zeus

In this section, we formally describe the Zeus protocol instance, given a session with ID sid. The protocol, Π_{Zeus} interacts with the global ideal functionalities, channel update subprotocols: deposit lock protocol $\Pi^i_{\mathrm{2pSetupDep}}$; premium lock subprotocol $\Pi^i_{\mathrm{2pSetupPrm}}$; principal lock protocol $\Pi^i_{\mathrm{2pSetupPrcpl}}$, the deposit-refund subprotocols ($\Pi^i_{\mathrm{2pRfdDep}}$), and the fast settlement subprotocol $\Pi^i_{\mathrm{2pSettle}}$. Meanwhile, Π_{Zeus} uses launcher sub-protocols to issue the on-chain disputes. We formally describe them in the following sections. Here, we highlight the sub-protocols in red.

Zeus Protocol: Π_{Zeus}

Parameters owned by all parties:

- The protocol-level deposit value c_{relay} and c_{payee}. And the padding fee f_0 to overcome the dust limit. We denote the on-chain transaction cost as F.
- sidSet: the set of all payment ids. Initially empty. For each $\mathtt{sid} \in \mathtt{sidSet}$, user maintains metadata $\mathcal{M}$, the payment lifetime T, the payee's stake Stake, and the status status of this payment.

Global Setup. For simplicity, for any payment with seesio id sid, we assume the payment metadata $\mathcal{M}$ the payment lifetime T, and the payee's stake Stake, are all known to all parties. $\mathcal{M}$ contains payment amount x, payment path and the corresponding relay fee $\{f_i\}_{i=1}^n$. The deposit, premium, and principal values can be derived from the metadata(see Section 6). After the global setup, sid is added to sidSet and status = setup.

Setup phase

$(\mathtt{sid}, \mathtt{init}, \mathtt{UTXO}_{src}) \overset{0}{\hookleftarrow} \mathcal{E}, U_0$:

1) If the Status of sid is not setup, then abort.
2) **Check payee's stake and source transaction.** $\mathrm{Out}_{\mathtt{Stake}} := \mathrm{checkPayeeStake}(U_{n+1}, T)$ If $\mathrm{Out}_{\mathtt{Stake}} = \perp$, then abort. Check if $\mathtt{UTXO}_{src}$ is a valid UTXO controlled by U_0 and have balance $\geq x + F$.
3) **Construct setup transactions.**

$$(tx_{\mathrm{rollback}}, tx_{\mathrm{redeem}}, tx_{\mathrm{slash}}) := \mathrm{genSetupTx}(\mathtt{UTXO}_{src}, \mathrm{Out}_{\mathtt{Stake}}, \mathcal{M})$$

4) **Request for accountability.** Send:

$$(\mathtt{sid}, \mathtt{request\text{-}acc}, tx_{\mathrm{rollback}}, tx_{\mathrm{redeem}}, tx_{\mathrm{slash}}) \overset{0}{\hookrightarrow} U_{n+1}$$

5) **Request for relay setup.** Send:

$$(\mathtt{sid}, \mathtt{request\text{-}setup}, tx_{\mathrm{rollback}}, tx_{\mathrm{redeem}}, tx_{\mathrm{slash}}) \overset{0}{\hookrightarrow} U_{i\in[1,n]}$$

$(\mathtt{sid}, \mathtt{request\text{-}acc}, tx_{\mathrm{rollback}}, tx_{\mathrm{redeem}}, tx_{\mathrm{slash}}) \overset{1}{\hookleftarrow} U_0, U_{n+1}$:

6) **Check the validity.** Check if $tx_{\mathrm{rollback}}, tx_{\mathrm{redeem}}, tx_{\mathrm{slash}}$ are constructed using the function genSetupTxs. If not, abort.
7) **Sign tx_{redeem} and enable the redeem launcher.**
   - $\sigma_{\mathrm{redeem}} := \mathrm{approveTx}(tx_{\mathrm{redeem}}, 0)$
   - $tx_{\mathrm{redeem}}.\mathrm{ScriptSigs}[0] := (\sigma_{\mathrm{redeem}})$
   - Send $(\mathtt{ssid}^{n+1}_{rl}, \mathtt{init}, T, tx_{\mathrm{rollback}}, tx_{\mathrm{redeem}}) \overset{1}{\hookrightarrow} \Pi_{\mathrm{redeem\text{-}launcher}}$
8) **Approve accountability.**

- $\sigma_{\text{slash}} := \text{approveTx}(tx_{\text{slash}}, 1)$, $tx_{\text{slash}}.\text{ScriptSigs}[1] := (\sigma_{\text{slash}})$.
- Send $(\texttt{sid}, \texttt{approved-acc}, tx_{\text{redeem}}, tx_{\text{slash}}) \overset{1}{\hookrightarrow} U_0$

Upon $(\texttt{sid}, \texttt{request-setup}, tx_{\text{rollback}}, tx_{\text{redeem}}, tx_{\text{slash}})$ from U_0, $U_{i\in[1,n]}$:

9) Check if $tx_{\text{rollback}}, tx_{\text{redeem}}, tx_{\text{slash}}$ are constructed using the function genSetupTxs. If not, abort.
10) Setup the next state of γ_{i-1}: Set $\alpha = \texttt{Dep}_{i-1}$, $\beta = 0$, $(\texttt{ssid}^{i-1}_{\text{Dep}}, \texttt{setup}, \alpha, \beta, T, tr^{i}_{\text{rollback}}, \perp) \overset{3}{\hookrightarrow} \Pi^{i-1}_{\text{2pSetupDep}}$ Then enter the lock-deposit phase.

$(\texttt{sid}, \texttt{approved-acc}, tx_{redeem}, tx_{\text{slash}}) \overset{2}{\hookleftarrow} U_{n+1}, U_0$:

11) **Check signature.** Check if tx_{redeem} and tx_{slash} contains valid signatures of U_{n+1}. If not, abort.
12) Sign the rollback and slash transactions:
    - $\sigma_{\text{rollback}} := \text{approveTx}(tx_{\text{rollback}}, 0)$, $tx_{\text{rollback}}.\text{ScriptSigs}[0] := (\sigma_{\text{rollback}})$.
    - $\sigma'_{\text{slash}} := \text{approveTx}(tx_{\text{slash}}, 0)$
    - $tx_{\text{slash}}.\text{ScriptSigs}[0] := (\sigma'_{\text{slash}})$
13) **Enable the rollback and slash launcher.**
    - $(\texttt{sid}, \texttt{init}, T, tx_{\text{rollback}}) \overset{2}{\hookrightarrow} \Pi^{0}_{\text{rollback-launcher}}$
    - $(\texttt{sid}, \texttt{init}, T, tx_{\text{rollback}}, tx_{slash}) \overset{2}{\hookrightarrow} \Pi_{\text{slash-launcher}}$

    Enter the lock-deposit phase.

`Lock-deposit phase`

U_0:

1) **Lock payer's deposit within γ_0.** Invoke:

$$(\texttt{ssid}^{0}_{\text{Dep}}, \texttt{init}, \gamma_0, \texttt{Dep}_1, tr^{0}_{\text{rollback}}, T) \hookrightarrow \Pi^{0}_{\text{2pSetupDep}}$$

and enter the lock-premium phase.

$(\texttt{ssid}^{i-1}_{\text{Dep}}, \texttt{Update-OK}, tx^{i-1}_{\text{LockDep}}) \hookleftarrow \Pi^{i-1}_{\text{2pSetupDep}}$, where $tx^{i-1}_{\text{LockDep}}$ indicates $\texttt{Dep}_{i-1}$ locked in γ_{i-1}, $U_{i\in[1,n]}$:

2) **Forward the deposit lock.** Invoke:

$$(\texttt{ssid}^{i}_{\text{Dep}}, \texttt{init}, \gamma_i.\text{id}, \texttt{Dep}_i, tr^{i}_{\text{rollback}}, T) \hookrightarrow \Pi^{i}_{\text{2pSetupDep}}$$

3) **Setup the Premium-lock state of γ_i.** Upon sub-protocol $\Pi^{i}_{\text{2pSetupDep}}$ confirms deposit is locked with a `Update-OK` then invoke:

$$(\texttt{ssid}^{i}_{\text{Prm}}, \texttt{setup}, \gamma_i, \texttt{Dep}_i, \texttt{Prm}_i, T, tr^{i}_{\text{rollback}}, tr^{i}_{\text{redeem}}) \hookrightarrow \Pi^{i}_{\text{2pSetupPrm}}$$

and enter the lock-premium phase.

$(\texttt{ssid}^{n}_{\text{Dep}}, texttt{Update}-OK) \hookleftarrow \Pi^{n}_{\text{2pSetupDep}}, U_{n+1}$:

4) **Enter the lock-premium phase.**

`Lock-premium phase`

U_{n+1}:

1) **Lock the premium.** Invoke:

$$(\texttt{ssid}^{n}_{\text{Prm}}, \texttt{init}, \gamma_n, \texttt{Dep}_n, \texttt{Prm}_n, T, tr^{n}_{rollback}, tr^{n}_{redeem}) \hookrightarrow \Pi^{n}_{\text{2pSetupPrm}}$$

2) **Prepare for principal-lock.** If $(\texttt{Update-Ok})$ is confirmed: $(\texttt{ssid}^{n}_{\text{Prcpl}}, \texttt{setup}, \texttt{Prcpl} + \texttt{Dep}_n, \texttt{Prm}_n, T, tr^{n}_{\text{rollback}}, tr^{n}_{\text{redeem}}) \hookrightarrow \Pi^{n}_{\text{2pSetupPrcpl}}$ then enter the next phase.

$(\texttt{ssid}^{i}_{\text{Prm}}, \texttt{Update-OK}, tx^{i}_{\text{Lock-Prm}}) \hookleftarrow \Pi^{i}_{\text{2pSetupPrm}}, U_{i\in[1,n]}$:

3) **Lock premium over γ_{i-1}:** $(\texttt{ssid}^{i-1}_{\text{Prm}}, \texttt{init}, \gamma_{i-1}, \texttt{Dep}_{i-1}, \texttt{Prm}_{i-1}, tr^{i-1}_{redeem}, T, tr^{i-1}_{\text{redeem}}) \hookrightarrow \Pi^{i-1}_{\text{2pSetupPrm}}$
4) **Setup the Principal-Lock state of γ_{i-1}.** Upon $(\texttt{Updated-Ok})$ confirmed: $(\texttt{ssid}^{i-1}_{\text{Prcpl}}, \texttt{setup}, \texttt{Dep}_{i-1} + \texttt{Prcpl}, \texttt{Prm}_{i-1}, T, tr^{i-1}_{\text{rollback}}, tr^{i-1}_{\text{redeem}}) \hookrightarrow \Pi^{i-1}_{\text{2pSetupPrcpl}}$ and enter the lock-principal phase.

$(\texttt{ssid}^{0}_{\text{Prm}}, \texttt{Update-OK}, tx^{0}_{\text{Lock-Prm}}) \hookleftarrow \Pi^{0}_{\text{2pSetupPrm}}, U_0$:

5) **Enter the lock-principal phase.**

`Lock-principal phase`

U_0:

1) **Lock principal within γ_0.** Invoke: $(\texttt{ssid}^{0}_{\text{Prcpl}}, \texttt{init}, \gamma_0, \texttt{Prcpl} + \texttt{Dep}_0, \texttt{Prm}_0, T, tr^{0}_{\text{rollback}}, tr^{0}_{\text{redeem}}) \hookrightarrow \Pi^{0}_{\text{2pSetupPrcpl}}$
2) **Setup the next phase.** Upon the confimration $(\texttt{Update-OK})$ received: $(\texttt{ssid}^{0}_{\text{RfdDep}}, \texttt{setup}, \texttt{Prcpl}, \texttt{Prm}_0, T, tr^{0}_{\text{rollback}}, tr^{0}_{\text{redeem}}) \hookrightarrow \Pi^{0}_{\text{2pRfdDep}}$

$(\texttt{ssid}^{i-1}_{\text{Prcpl}}, \texttt{Update-Ok}, tx^{i-1}_{\text{Lock-Prcpl}}) \hookleftarrow \Pi^{i-1}_{\text{2pSetupPrcpl}}, U_{i\in[1,n]}$:

3) **Forward the Principal lock.** Invoke: $(\texttt{ssid}^{i}_{\text{Prcpl}}, \texttt{init}, \gamma_i, \texttt{Prcpl} + \texttt{Dep}_i, \texttt{Prm}_i, T, tr^{i}_{\text{rollback}}, tr^{i}_{\text{redeem}}) \hookrightarrow \Pi^{i}_{\text{2pSetupPrcpl}}$
4) **Setup the deposit-refund phase.** Upon the principal lock confirmed: $(\texttt{ssid}^{i}_{\text{RfdDep}}, \texttt{setup}, \texttt{Prcpl}, \texttt{Prm}_i, T, tr^{i}_{\text{rollback}}, tr^{i}_{\text{redeem}}) \hookrightarrow \Pi^{i}_{\text{2pRfdDep}}$ and enter the next phase.

$(\texttt{ssid}^{n}_{\text{Prcpl}}, \texttt{Update-Ok}, tx^{n}_{\text{Lock-Prcpl}}) \hookleftarrow \Pi^{n}_{\text{2pSetupPrcpl}}, U_{n+1}$: Enter the deposit-refund phase.

`Deposit-Refund phase`

U_{n+1}:

1) **Refund the payer side deposit.** Invoke: $(\texttt{ssid}^{n}_{\text{RfdDep}}, \texttt{init}, \gamma_n, \texttt{Prcpl}, \texttt{Prm}_n, T, tr^{n}_{\text{rollback}}, tr^{n}_{\text{redeem}}) \hookrightarrow \Pi^{n}_{\text{2pRfdDep}}$
2) **Setup the settle phase.** If $\Pi^{n}_{\text{2pRfdDep}}$ confirms deposit-refund with `Update-OK` message: $(\texttt{ssid}^{n}_{\text{Settle}}, \texttt{setup}, \gamma_n, lb - \texttt{Prcpl} + \texttt{Prm}_n, rb + \texttt{Prcpl} - \texttt{Prm}_n) \hookrightarrow \Pi^{n}_{\text{2pSettle}}$, set a confirmation message $(\texttt{sid}, \texttt{confirm}) \hookrightarrow U_0$, then enter the next phase.

$(\texttt{ssid}^{i}_{\text{RfdDep}}, \texttt{Update-OK}, tx^{i}_{\text{Rfd-Dep}}) \hookleftarrow \Pi^{i}_{\text{2pRfdDep}}, U_{i\in[1,n]}$:

3) **Forward the revoke over γ_{i-1}.** $(\texttt{ssid}^{i-1}_{\text{RfdDep}}, \texttt{init}, \gamma_{i-1}, \texttt{Prcpl}, \texttt{Prm}_{i-1}, T, tr^{i-1}_{\text{rollback}}, tr^{i-1}_{\text{redeem}}) \hookrightarrow \Pi^{i-1}_{\text{2pRfdDep}}$
4) **Setup the settle phase.** If $\Pi^{i-1}_{\text{2pRfdDep}}$ confirms deposit-refund with `Update-OK` message: $(\texttt{ssid}^{i-1}_{\text{Settle}}, \texttt{setup}, \gamma_{i-1}, lb - \texttt{Prcpl} + \texttt{Prm}_n, rb + \texttt{Prcpl} - \texttt{Prm}_n) \hookrightarrow \Pi^{i-1}_{\text{2pSettle}}$ then enter the next phase. Otherwise, terminate.

Upon $(\texttt{ssid}^{0}_{\text{RfdDep}}, \texttt{Update-OK}, tx^{0i}_{\text{Rfd-Dep}}) \hookleftarrow \Pi^{0}_{\text{2pRfdDep}}, U_0$: If U_0 recieved the confirmation from U_{n+1}, then enter the fast settlement phase.

`Settle phase`

U_0:

1) **Settle the payment.** $(\texttt{ssid}^{0}_{\text{Settle}}, \texttt{init}, \gamma_0, lb - x - \sum_{j=1}^{n} f_j, rb + x + \sum_{j=1}^{n} f_j) \hookrightarrow \Pi^{0}_{\text{2pSettle}}$ And once $\Pi^{0}_{\text{2pSettle}}$ confirms with `settle-OK` , then return $(\texttt{sid}, \texttt{settled}, s) \hookrightarrow \mathcal{E}$.
2) Terminate all subprotocols. Return $(\texttt{sid}, \texttt{settled})$ and terminate. then terminate.

$(\texttt{ssid}^{i-1}_{\text{Settle}}, \texttt{settle-OK}) \hookleftarrow \Pi^{i-1}_{\text{2pSettle}}, U_{i\in[1,n]}$:

3) **Continue settlement chain:** $(\mathtt{ssid}^i_{\text{Settle}}, \mathtt{init}, \gamma_i.id, lb - \mathtt{Prcpl} + \mathtt{Prm}_i, rb + \mathtt{Prcpl} - \mathtt{Prm}_i) \hookrightarrow \Pi^i_{\text{2pSettle}}$.

4) Once U_i confirms γ_i is settled: Terminate all subprotocols. Return $(\mathtt{sid}, \mathtt{settled})$ and terminate.

$(\mathtt{ssid}^n_{\text{Settle}}, \mathtt{settle\text{-}OK}) \hookleftarrow \Pi^n_{\text{2pSettle}}$, U_{n+1}:

5) terminates all subprotocols. Return $(\mathtt{sid}, \mathtt{settled})$.

D.2 Formal Descriptions of Sub-Procedures for Channel Updates

In this section, we formalize the sub-protocols used in the main protocol Π_{Zeus}. Specifically, we have two main sub-protocols: Π^i_{2SR} and $\Pi^i_{\text{R-2SR}}$. Π^i_{2SR} is the two-stage revocation channel update over channel γ_i, where U_i is the initializer and U_{i+1} is the follower. The sub-protocol $\Pi^i_{\text{R-2SR}}$ is the reverse version of the two-stage revocation channel update over channel γ_i, where U_{i+1} is the initializer and U_i is the follower. The sub-protocol Π^i_{2SR} is used to lock the deposit and principal, while $\Pi^i_{\text{R-2SR}}$ is used to lock the premium and refund the deposit. For better readability, we use $\Pi^i_{\text{2pSetupDep}}$ and $\Pi^i_{\text{2pSetupPrcpl}}$ to denote the corresponding instances of Π^i_{2SR}, and $\Pi^i_{\text{2pSetupPrm}}$ and Π^i_{2pRfdDep} to denote the corresponding instances of $\Pi^i_{\text{R-2SR}}$.

Π^i_{2SR}

$(\mathtt{sid}, \mathtt{setup}, \gamma, \alpha, \beta, T, tr^i_{\text{rollback}}, tr^i_{\text{redeem}}) \overset{\leq\tau}{\hookleftarrow} \mathcal{E}$, U_{i+1}:

0) If $\mathtt{sid}$ have been observed before, then abort. Otherwise, save $\gamma_i, \alpha, \beta, T, tr^i_{\text{rollback}}, tr^i_{\text{redeem}}$, and set the status of this session as $\mathtt{enabled}$.

$(\mathtt{sid}, \mathtt{init}, \gamma_i, \alpha, \beta, T, tr^i_{\text{rollback}}, tr^i_{\text{redeem}}) \overset{\tau}{\hookleftarrow} \mathcal{E}$, U_i:

1) **Generate the new 2SR state transactions.** If $\mathtt{sid}$ have been observed before, then abort.

$$(tx_s,\ tx^l_{Rvk},\ tx^r_{Rvk}) := \text{gen2SRTx}(\gamma_i, \alpha, \beta, T, tr^i_{\text{rollback}}, tr^i_{\text{redeem}})$$

2) **Approve the revoke transactions.**

- $\sigma_1 := \text{approveTx}(tx^l_{Rvk}, 0)$, $tx^l_{Rvk}.\text{ScriptSigs}[0] := (\sigma_1)$
- If $\beta > 0$, $\sigma_2 := \text{approveTx}(tx^r_{Rvk}, 0)$ and $tx^r_{Rvk}.\text{ScriptSigs}[0] := (\sigma_2)$.

3) **Request for signing new state.**

$$(\mathtt{sid}, \mathtt{request}, \gamma_i, tx_s,\ tx^l_{Rvk}, tx^r_{Rvk}) \overset{\tau}{\hookrightarrow} U_{i+1}$$

$(\mathtt{sid}, \mathtt{request}, \cdots) \overset{\tau+1}{\hookleftarrow} U_i$, U_{i+1}:

4) **Check proposed state.** Check if $status = \mathtt{enabled}$ and

$$\top = \text{check2SRTx}(\gamma_i, \alpha, \beta, T, tr^i_{\text{rollback}}, tr^i_{\text{redeem}}, tx_s,\ tx^l_{Rvk}, tx^r_{Rvk})$$

If any check fails, abort.

5) **Approve revoke transactions.**

- $\sigma_3 := \text{approveTx}(tx^l_{Rvk}, 0)$, add σ_3 in $tx^l_{Rvk}.\text{ScriptSigs}[0]$
- If $\beta > 0$: $\sigma_4 := \text{approveTx}(tx^r_{Rvk}, 0)$, and Add σ_4 in $tx^r_{Rvk}.\text{ScriptSigs}[0]$.

6) **Approve the new state.**

- $\sigma^r_s := \text{approveTx}(tx_s, 0)$.
- Send $(\mathtt{sid}, \mathtt{approved}, tx_s,\ tx^l_{Rvk}, tx^r_{Rvk}, \sigma^r_s) \overset{\tau+1}{\hookrightarrow} U_i$

7) **Setup handlers**

- If $\beta > 0$: $\sigma := \text{approveTx}(tx^r_{Rvk}, 1)$, $tx^r_{Rvk}.\text{ScriptSigs}[1] := (\sigma)$. Sends $(\mathtt{ssid}_3, \mathtt{Setup}, \gamma_i, T, tr^i_{redeem}, tx^r_{Rvk}) \overset{\tau'}{\hookrightarrow} \Pi^i_{\text{redeem-handler}}$
- $(\mathtt{ssid}_4, \mathtt{Setup}, \gamma_i, T) \overset{\tau'}{\hookrightarrow} \Pi^i_{\text{rollback-handler}}$

$(\mathtt{sid}, \mathtt{approved}, \cdots) \overset{\tau+2}{\hookleftarrow} U_{i+1}, U_i$:

8) **Check the signature.** Continue if all txs contains valid signature of U_{i+1}. And σ^r_s is valid. Otherwise, abort.

9) **Approve the new state.** Set $\sigma^l_s := \text{approveTx}(tx_s, 0)$ and $tx_s := \text{genUpdateTx}(tx_s, \sigma^l_s, \sigma^r_s)$

10) **Update the channel state.** Send $(\mathtt{sid}, \mathtt{UPDATE}, \gamma_i.id, tx_s) \overset{\tau+2}{\hookrightarrow} \mathcal{F}_{Channels}$.

11) **Setup handlers.** Upon $(\mathtt{sid}, \mathtt{UPDATED}, \gamma_i.\text{id}, tx_s) \overset{\tau'}{\hookleftarrow} \mathcal{F}_{Channels}$, then:

- $\sigma := \text{approveTx}(tx^l_{Rvk}, 1)$
- $tx^l_{Rvk}.\text{ScriptSigs}[1] := (\sigma)$
- $(\mathtt{ssid}_1, \mathtt{Setup}, \gamma_i, T, tr^i_{rollback}, tx^l_{Rvk}) \overset{\tau'}{\hookrightarrow} \Pi^i_{\text{rollback-handler}}$
- $(\mathtt{ssid}_2, \mathtt{Setup}, \gamma_i, T) \overset{\tau'}{\hookrightarrow} \Pi^i_{\text{redeem-handler}}$
- Return $(\mathtt{sid}, \mathtt{Update\text{-}Ok}, tx_s) \overset{\tau'}{\hookrightarrow} \mathcal{E}$

$(\mathtt{sid}, \mathtt{UPDATED}, \gamma_i.\text{id}, tx_s) \overset{\tau'}{\hookleftarrow} \mathcal{F}_{Channels}$, U_{i+1}:

12) Return $(\mathtt{sid}, \mathtt{Update\text{-}OK}, tx_s) \overset{\tau'}{\hookrightarrow} \mathcal{E}$.

Here, we formalize the critial sub-sub-protocol of Π_{2SR} and reverse sub-sub-protocol of $\Pi_{\text{R-2SR}}$, which are used to handle the rollback and redeem operations. And when such on-chain operations are not triggered, they will help user claim the funds in the channel.

$\Pi^i_{\mathtt{rollback\text{-}handler}}$ executed over γ_i

Parameters owned by U_i for each session:

- T: the expire time to end this handler.
- $tr_{rollback}$: the rollback transaction to be watched.
- tx^l_{Rvk}: Left revocation transactions to be broadcasted.
- $\mathtt{status}$: the status of this session, initially set to $\mathtt{IDLE}$.

$(\mathtt{sid}, \mathtt{Setup}, \gamma_i, T, tr_{rollback}, tx^l_{Rvk}) \overset{\tau}{\hookleftarrow} \tilde{U}_i, U_i$:

1) Abort if $\mathtt{sid}$ is observed, Save those transactions and deadline T.

2) At block $T - t_{close} - \Delta$: Set $\mathtt{status} = \mathtt{revoking}$ if $tr_{rollback}$ is confirmed at $\mathcal{G}_{\text{L}}$.

3) If $\mathtt{status} = \mathtt{revoking}$, and the γ_i's latest state is spendable by tx^l_{Rvk}:

- Close channel γ_i by sending: $(\mathtt{ssid}_C, \mathtt{CLOSE}, \gamma_i.id) \hookrightarrow \mathcal{F}_{Channels}$.
- Upon $\mathcal{F}_{Channels}$ confirming the channel closure, if tx^l_{rvk} is not a valid transaction(e.g., channel have been updated to a new state), then abort. Otherwise, submits the left revocation transaction tx^l_{Rvk} to $\mathcal{G}_{\text{L}}$: $(\mathtt{ssid}_L, \mathtt{POST}, tx^l_{rvk}) \hookrightarrow \mathcal{G}_{\text{L}}$.

Upon $(\mathtt{sid}, \mathtt{Setup}, \gamma_i, T) \hookleftarrow \mathcal{F}_{Channels}, U_i$: at block height $T - t_{close}$, if there are funds locked by U_i wll be spendable after T:

- Close channel γ_i by sending: $(\mathtt{ssid}_C, \mathtt{CLOSE}, \gamma_i.id) \hookrightarrow \mathcal{F}_{Channels}$.

- At T, the $\mathcal{F}_{Channels}$ must have confirmed the channel closure with its latest state tx_s.
- Construct a valid transaction tx_{valid}, which takes all spendable ouputs in tx_s as inputs, and a single output paid to U_{i+1}. Send $(\texttt{ssid}_L, \texttt{POST}, tx_{\text{valid}}) \hookrightarrow \mathcal{G}_{\text{L}}$.

Redeem Handler Protocol: $\Pi^i_{\texttt{redeem-handler}}$ executed over γ_i

Parameters owned by U_{i+1} for each session:

- T: the expire time to end this handler.
- tr_{redeem}: the redeem trigger transaction to be watched.
- tx^r_{Rvk}: The right-side revocation transaction to be broadcasted.
- `status`: the status of this session, initially set to `IDLE`.

$(\texttt{sid}, \texttt{UPDATE}, \gamma_i, T, tr_{redeem}, tx^r_{Rvk}) \overset{\tau}{\hookleftarrow} \tilde{U}_{i+1}, U_{i+1}$:

1) Abort if `sid` is observed. Otherwise, save the parameters $T, tr_{redeem}, tx^r_{Rvk}$ for this session.
2) At block $T - t_{close} - \Delta$: Set `status` := `revoking` if tr_{redeem} is confirmed in $\mathcal{G}_{\text{L}}$.
3) If `status` = `revoking`, and the γ_i's latest state is spendable by tx^r_{Rvk}:
   - Close channel γ_i by sending: $(\texttt{ssid}_C, \texttt{CLOSE}, \gamma_i.id) \hookrightarrow \mathcal{F}_{Channels}$.
   - Upon $\mathcal{F}_{Channels}$ confirming the channel closure, if tx^r_{Rvk} is not a valid transaction (e.g., the channel has been updated to a new state), then abort. Otherwise, submit the right-side revocation transaction tx^r_{Rvk} to $\mathcal{G}_{\text{L}}$: $(\texttt{ssid}_L, \texttt{POST}, tx^r_{Rvk}) \hookrightarrow \mathcal{G}_{\text{L}}$.

Upon $(\texttt{sid}, \texttt{Setup}, \gamma_i, T) \hookleftarrow \mathcal{F}_{Channels}, U_{i+1}$: at block height $T - t_{close} + \Delta$, if there are funds locked by U_{i+1} that will be spendable by U_i after $T + \Delta$:

- Close channel γ_i by sending: $(\texttt{ssid}_C, \texttt{CLOSE}, \gamma_i.id) \hookrightarrow \mathcal{F}_{Channels}$.
- At $T + \Delta$, the $\mathcal{F}_{Channels}$ must have confirmed the channel closure with its latest state tx_s.
- Construct a valid transaction tx_{valid}, which takes all spendable outputs in tx_s as inputs, and a single output paid to U_i. Send $(\texttt{ssid}_L, \texttt{POST}, tx_{\text{valid}}) \hookrightarrow \mathcal{G}_{\text{L}}$.

$\Pi^i_{\text{R-2SR}}$

$(\texttt{sid}, \texttt{setup}, \gamma_i, \alpha, \beta, T, tr^i_{\text{rollback}}, tr^i_{\text{redeem}}) \overset{\leq\tau}{\hookleftarrow} \mathcal{E}, U_i$:

0) If `sid` has been observed before, abort. Otherwise, save $(\gamma_i, \alpha, \beta, T, tr^i_{\text{rollback}}, tr^i_{\text{redeem}})$ and set the session status to `enabled`.

$(\texttt{sid}, \texttt{init}, \gamma_i, \alpha, \beta, T, tr^i_{\text{rollback}}, tr^i_{\text{redeem}}) \overset{\tau}{\hookleftarrow} \mathcal{E}, U_{i+1}$:

1) **Generate the new 2SR state transactions.** If `sid` have been observed before, then abort.

$$(tx_s,\ tx^l_{Rvk},\ tx^r_{Rvk}) := \text{gen2SRTx}(\gamma_i, \alpha, \beta, T, tr^i_{\text{rollback}}, tr^i_{\text{redeem}})$$

2) **Approve the revoke transactions.**
   - $\sigma_1 := \text{approveTx}(tx^l_{Rvk}, 0)$, tx^l_{Rvk}.ScriptSigs[0] := (σ_1)
   - $\sigma_2 := \text{approveTx}(tx^r_{Rvk}, 0)$ and tx^r_{Rvk}.ScriptSigs[0] := (σ_2).
3) **Request for signing new state.**

$$(\texttt{sid}, \texttt{request}, \gamma_i, tx_s,\ tx^l_{Rvk}, tx^r_{Rvk}) \overset{\tau}{\hookrightarrow} U_i$$

$(\texttt{sid}, \texttt{request}, \cdots) \overset{\tau+1}{\hookleftarrow} U_{i+1}, U_i$:

4) **Check proposed state.** Verify that $status$ = `enabled` and

$$\top = \text{checkR2SRTx}(\gamma_i, \alpha, \beta, T, tr^i_{\text{rollback}}, tr^i_{\text{redeem}}, tx_s, tx^l_{Rvk}, tx^r_{Rvk})$$

If any check fails, abort.

5) **Approve revoke transactions.**
   - $\sigma_3 := \text{approveTx}(tx^l_{Rvk}, 0)$, add σ_3 to tx^l_{Rvk}.ScriptSigs[0].
   - If $\beta > 0$: $\sigma_4 := \text{approveTx}(tx^r_{Rvk}, 0)$, add σ_4 to tx^r_{Rvk}.ScriptSigs[0].
6) **Approve the new state.**
   - $\sigma^l_s := \text{approveTx}(tx_s, 0)$.
   - Send $(\texttt{sid}, \texttt{approved}, tx_s, tx^l_{Rvk}, tx^r_{Rvk}, \sigma^l_s) \overset{\tau+1}{\hookrightarrow} U_i$.
7) **Setup handlers.**
   - $\sigma := \text{approveTx}(tx^l_{Rvk}, 1)$, set tx^l_{Rvk}.ScriptSigs[1] := (σ). Send $(\texttt{ssid}_3, \texttt{Setup}, \gamma_i, T, tr^i_{redeem}, tx^l_{Rvk}) \overset{\tau'}{\hookrightarrow} \Pi^i_{\text{rollback-handler}}$.
   - $(\texttt{ssid}_4, \texttt{Setup}, \gamma_i, T) \overset{\tau'}{\hookrightarrow} \Pi^i_{\text{redeem-handler}}$.

$(\texttt{sid}, \texttt{approved}, \cdots) \overset{\tau+2}{\hookleftarrow} U_i, U_{i+1}$:

8) **Check the signature.** Continue if all transactions contain valid signatures from U_i. And σ^l_s is valid.
9) **Approve the new state.** Set $\sigma^r_s := \text{approveTx}(tx_s, 0)$ and $tx_s := \text{genUpdateTx}(tx_s, \sigma^l_s, \sigma^r_s)$.
10) **Update the channel state.** Send $(\texttt{sid}, \texttt{UPDATE}, \gamma_i.id, tx_s) \overset{\tau+2}{\hookrightarrow} \mathcal{F}_{Channels}$.
11) **Setup handlers.** Upon $(\texttt{sid}, \texttt{UPDATED}, \gamma_i.\text{id}, tx_s) \overset{\tau'}{\hookleftarrow} \mathcal{F}_{Channels}$, then:
   - $\sigma := \text{approveTx}(tx^r_{Rvk}, 1)$, set tx^r_{Rvk}.ScriptSigs[1] := (σ).
   - $(\texttt{ssid}_1, \texttt{Setup}, \gamma_i, T, tr^i_{redeem}, tx^r_{Rvk}) \overset{\tau'}{\hookrightarrow} \Pi^i_{\text{redeem-handler}}$.
   - $(\texttt{ssid}_2, \texttt{Setup}, \gamma_i, T) \overset{\tau'}{\hookrightarrow} \Pi^i_{\text{rollback-handler}}$.
   - Return $(\texttt{sid}, \texttt{Update-Ok}, tx_s) \overset{\tau'}{\hookrightarrow} \mathcal{E}$.

$(\texttt{sid}, \texttt{UPDATED}, \gamma_i.\text{id}, tx_s) \overset{\tau'}{\hookleftarrow} \mathcal{F}_{Channels}, U_i$:

12) Return $(\texttt{sid}, \texttt{Update-OK}, tx_s) \overset{\tau'}{\hookrightarrow} \mathcal{E}$.

2pSettle Protocol: Π^i_{2pSettle}

$(\texttt{sid}, \texttt{setup}, lb', rb') \overset{\leq\tau}{\hookleftarrow} \mathcal{E}, U_{i+1}$:

0) If `sid` is not fresh, then abort.

$(\texttt{sid}, \texttt{init}, \gamma_i.id, lb', rb') \overset{\tau}{\hookleftarrow} \mathcal{E}, U_i$:

1) **Generate `paid` state over γ_i.** If `sid` is not fresh, then abort. Otherwise:

$$tx^i_{\texttt{settle}} := \text{genSettleTx}(\gamma_i.id, lb', rb')$$

2) **Request for settle.**

$$(\texttt{sid}, \texttt{request-settle}, \gamma_i.id, tx^i_{\texttt{settle}}) \overset{\tau}{\hookrightarrow} U_{i+1}$$

$(\texttt{sid}, \texttt{request-settle}, \gamma_i.id, tx^i_{\texttt{settle}}) \overset{\tau+1}{\hookleftarrow} U_i, U_{i+1}$:

3) **Check the balance**. Check if $tx^i_{\texttt{settle}}$ reflects the agreed balance of lb' and rb'. If any check fails, abort.
4) **Approve the settlement.**

$$\sigma^{i,r}_{\texttt{settle}} := \text{approveTx}(tx^i_{\texttt{settle}}, 0)$$

5) **Send approval to U_i:**

$$(\texttt{sid}, \texttt{approved-settle}, \sigma^{i,r}_{\texttt{settle}}) \xrightarrow{\tau+1} U_i$$

$\underline{(\texttt{sid}, \texttt{approved-settle}, \sigma^{i,r}_{\texttt{settle}}) \xleftarrow{\tau+2} U_{i+1}, U_i:}$

6) **Check the signature validity.** Verify that $\sigma^{i,r}_{\texttt{settle}}$ is valid. Otherwise, abort.

7) **Generate the settle state of γ_i.**

- $\sigma^{i,l}_{\texttt{settle}} := \text{approveTx}(tx^i_{\texttt{settle}}, 0)$
- $tx^i_{\texttt{settle}} := \text{genUpdateTx}(tx^i_{\texttt{settle}}, \sigma^{i,l}_{\texttt{settle}}, \sigma^{i,r}_{\texttt{settle}})$

8) **Update the channel state.**

$$(\texttt{sid}_C, \texttt{UPDATE}, \gamma_i.id, tx^i_{\texttt{settle}}) \xrightarrow{\tau+2} \mathcal{F}_{Channels}$$

9) If $(\texttt{sid}, \texttt{UPDATED}, \gamma_i.\text{id}, tx^i_{\texttt{settle}}.\text{Outs}) \xleftarrow{\tau'} \mathcal{F}_{Channels}$, then send $(\texttt{sid}, \texttt{settle-Ok}) \xhookrightarrow{\tau'} \mathcal{E}$.

$\underline{(\texttt{sid}, \texttt{UPDATED}, \gamma_i.\text{id}, tx^i_{\texttt{settle}}.\text{Outs}) \xleftarrow{\tau'} \mathcal{F}_{Channels}, U_{i+1}:}$

10) **Return the final state.** $(\texttt{sid}, \texttt{settle-Ok}) \xhookrightarrow{\tau'} \mathcal{E}$
If channel have not been updated within t_{update} blocks, abort.

D.3 Formal Descriptions of sub-procedures for launchers

In addition to the two-stage revocation channel update, U_0 will launch a rollback launcher $\Pi_{\text{rollback-launcher}}$ to handle the rollback transactions, and U_{n+1} will launch a redeem launcher $\Pi_{\text{redeem-launcher}}$ to handle the redeem transactions. The slash launcher $\Pi_{\text{slash-launcher}}$ is launched by U_0 to reclaim its funds in case of a failure in the redeem launcher.

Rollback Launcher Protocol: $\Pi_{\text{rollback-launcher}}$

$\underline{(\texttt{sid}, \texttt{init}, T, tx_{rollback}) \xleftarrow{\tau} \mathcal{E}, U_i:}$

1) If $\texttt{sid}$ is not fresh, then abort.
2) Save T, $tx_{rollback}$.
3) At block height $T - t_{close} - 2\Delta$, send $tx_{rollback}$ on-chain.

$$(\texttt{ssid}, \texttt{POST}, tx_{rollback}) \hookrightarrow \mathcal{G}_{\text{L}}$$

By functionality of $\mathcal{G}_{\text{L}}$, $tx_{rollback}$ appears in $\mathcal{G}_{\text{L}}$ at block height $T - t_{close} - 2\Delta + 1$ when latest block is $T - t_{close} - \Delta$.

$\underline{(\texttt{sid}, \texttt{CLOSE}) \xleftarrow{\tau^* \geq \tau} \mathcal{E}, U_i:}$

4) Terminate the protocol.

Redeem Launcher Protocol: $\Pi_{\text{redeem-launcher}}$

$\underline{(\texttt{sid}, \texttt{init}, T, tx_{rollback}, tx_{redeem}) \xleftarrow{\tau} \mathcal{E}, U_{i+1}:}$

1) If $\texttt{sid}$ is not fresh, then abort.
2) Save T, $tx_{rollback}$, and tx_{redeem}.
3) At every round τ' no later than block $T - t_{close} - \Delta$, if $tx_{rollback}$ confirms on $\mathcal{G}_{\text{L}}$, then send:

$$(\texttt{ssid}, \texttt{POST}, tx_{redeem}) \xhookrightarrow{\tau'} \mathcal{G}_{\text{L}}$$

4) Terminaate after round $T - t_{close} - \Delta$.

Slash Launcher Protocol: $\Pi_{\text{slash-launcher}}$

$\underline{(\texttt{sid}, \texttt{init}, T, tx_{rollback}, tx_{slash}) \xleftarrow{\tau} \mathcal{E}, U_i:}$

1) If $\texttt{sid}$ is not fresh, then abort.
2) Save T, $tx_{rollback}$, and tx_{slash}.
3) At round $T - t_{close}$, if $tx_{rollback}$'s outputs remains unspent, then send slash transaction tx_{slash} to reclaim the funds:
   - $(\texttt{ssid}, \texttt{POST}, tx_{slash}) \hookrightarrow \mathcal{G}_{\text{L}}$
   - At block $T - t_{close} + \Delta$, if tx_{slash} is confirmed at block $T - t_{close} + \Delta$, terminate. Otherwise: generate other tx' to spend the 0-th output $tx_{rollback}$ to U_0's address. Submit:

$$(\texttt{ssid}, \texttt{POST}, tx') \hookrightarrow \mathcal{G}_{\text{L}}$$

E Auxiliary Algorithms

E.1 Helper functions.

This section includes several useful functions that help to construct, approve transactions, and update the state of the payment channel.

$\underline{\text{checkPayeeStake}(U, T):}$

1. **If** $\exists tx \in \mathcal{G}_{\text{L}}.\text{TXs}$, where tx contains an unspent output with index i, such that:
   - $tx.\text{Outs}[i].\text{amt} \geq c_{\text{burn}} + F$, and
   - $tx.\text{Outs}[i].\phi$ requires the spending transaction tx_{spent} satisfy:
     - If $tx.\text{Outs}[i]$ is spent before T_{expire}, then
     $$tx_{\text{spent}}.\text{Outs}[0] = \{\text{amt} : c_{\text{burn}}, \quad \phi = \text{Sig}(\perp))$$
     - After T_{expire}, U can spend the output with a valid signature.
   - $T_{\text{expire}} \gg T$
2. **If** all checks pass, **return** (tx, i).
3. Otherwise, **return** $\perp$.

$\underline{\text{genSetupTxs}(\texttt{UTXO}_{src}, \text{Out}_{\texttt{stake}}, \mathcal{M}):}$

1. Lookup the corresponding channel γ based on $\gamma.id$.
2. Parse $\mathcal{M}$ to obtain the parameters within it: $\mathcal{M} = (U, \Gamma, x, \text{Fees}, \mathcal{S})$.
3. Construct the Rollback transaction tx_{rollback}:
   - $\hat{\theta}_0 := \{\text{amt} : x + (n+1)\varepsilon, \phi : (\text{Sig}(U_{n+1}) \geq rel(\Delta)) \vee (\text{Sig}(U_0) \geq \text{rel}(2\Delta)) \vee (> T - t_{close} - 2\Delta)\}$

- For $i \in [1,n]$, $\hat{\theta}_i := \{\text{amt} : \varepsilon, \phi : \text{Sig}(U_i) \geq \text{rel}(t_{\text{close}} + \Delta)\}$.
- $\theta_0 := \{\text{amt} : lb - x - (2n+2)\varepsilon, \phi : \text{Sig}(U_0)\}$.
- $\theta_1 := \{\text{amt} : rb, \phi : \text{Sig}(U_1)\}$.
- $tx_{\text{rollback}}.\text{Outs} := \{\hat{\theta}_0, \ldots, \hat{\theta}_n, \theta_0, \theta_1\}$.
- $tx_{\text{rollback}}.\text{Ins}[0] := \texttt{UTXO}_{src}$.
- $\sigma_{\text{rollback}} := \texttt{approveTx}(tx_{\text{rollback}}, 0)$.
- $tx_{\text{rollback}}.\text{ScriptSigs}[0] := \sigma_{\text{rollback}}$.

4. Generate the redeem transaction tx_{redeem}:
   - $\theta_0 := \{\text{amt} : x, \phi : (\text{Sig}(U_{n+1}))\}$.
   - For each $i \in [1, n+1]$, $\theta_{i+1} := \{\text{amt} : \varepsilon, \phi : \text{Sig}(U_i) \geq \text{rel}(t_{\text{close}} + \Delta)\}$.
   - $tx_{\text{redeem}} := \{\text{Ins}[0] : \{tx : tx_{\text{rollback}}, \text{idx} : 0\}, \text{Outs} : (\theta_0, \ldots, \theta_{n+1})\}$.
5. Generate the slash transaction tx_{slash}:
   - $\theta_0 := \{\text{amt} : x + n\varepsilon, \phi : \text{Sig}(U_0)\}$.
   - $\theta_1 := \{\text{amt} : c_{\text{burn}}, \phi : \text{Sig}(0)\}$.
   - $tx_{\text{slash}} := \{\text{Ins}[0] : \{tx : tx_{\text{rollback}}, \text{idx} : 0\}, \text{Ins}[1] : \text{Out}_{\texttt{stake}}, \text{Outs} : (\theta_{\text{slash}}, \theta_{\text{return}})\}$.
6. Return $(tx_{\text{rollback}}, tx_{\text{redeem}}, tx_{\text{slash}})$.

approveTx(tx, idx) *called by* U :

1. Parse the "idx"-th input of transaction tx to obtain the spending UTXO, $Out_{\text{spending}} := tx.\text{Ins}[\text{idx}]$.
2. **If** the spending UTXO is locked with $\text{Sig}(U)$, then **return** $\sigma := \text{Sign}(U.\text{sk}, tx)$.
3. Otherwise, **return** $\perp$.

checkSlashSig$(\sigma_{\text{slash}}, tx_{\text{slash}})$:

1. Check if σ_{slash} is a valid signature to spend $tx_{\text{slash}}.\text{Ins}[0]$.
2. **If** yes, **return** $\top$, otherwise, **return** $\perp$.

genUpdateTx$(\gamma_{id}, tx_{\text{state}}, \sigma^l, \sigma^r)$:

1. Construct a channel udpate transaction tx_{update} copying the state of tx_{state}, if σ^l and σ^r are valid signatures to spend the left and right channel states, respectively.
2. Return tx_{update}.

gen2SRTx$(\gamma, \alpha, \beta, T, tr_{rollback}, tr_{redeem})$:

1. Denotes the total balance owned by $\gamma.lu$ as lb(including all deposit locked by $\gamma.lu$ in this MHP) and the total balance owned by $\gamma.ru$ as rb.
2. Fill the left-side locked payment $\theta_0 := \{\text{amt} := \alpha, \phi : (\text{Sig}(\gamma.ru) \geq T) \vee (\text{Sig}(\gamma.ru) \wedge \text{Sig}(\gamma.lu))\}$.
3. If $\beta > 0$, fill the right-side locked payment $\theta_1 := \{\text{amt} := \beta, \phi : (\text{Sig}(\gamma.lu) \geq T + \Delta) \vee (\text{Sig}(\gamma.ru) \wedge \text{Sig}(\gamma.lu))\}$.
4. Construct the remained outputs:

$$\theta_2 := \{\text{amt} : lb - \alpha, \phi : \text{Sig}(\gamma.lu)\}$$

$$\theta_3 := \{\text{amt} : rb - \beta, \phi : \text{Sig}(\gamma.ru)\}$$

5. Construct the ouputs of state tx ($\beta > 0$):

$$tx_s.\text{Outs} := \{\theta_0, \theta_2, \theta_2, \theta_3\}$$

6. Construct the ouputs of state tx ($\beta = 0$):

$$tx_s.\text{Outs} := \{\theta_0, \theta_2, \theta_3\}$$

7. Construct left revoke tx' inputs:

$$tx^l_{Rvk}.\text{Outs}[0] := \left\{ \text{amt} : \gamma.\text{state}[0] + \varepsilon, \quad \phi : \text{Sig}(\gamma.lu) \right\}$$

8. Set the first input to refer to left-side locked amount α:

$$tx^l_{Rvk}.\text{Ins}[0] := \{tx : tx_s, \text{idx} : 0\}$$

9. Link the rollback trigger.

$$tx_s.\text{Ins}[1] := tr_{\text{rollback}}$$

10. Define the right revocation outputs ($\beta > 0$) :

$$tx^r_{Rvk}.\text{Outs}[0] := \left\{ \text{amt} : \gamma.\text{state}[1] + \varepsilon, \quad \phi : \text{Sig}(\gamma.ru) \right\}$$

11. Construct the revoke-premium transaction:

$$tx^r_{Rvk}.\text{Ins}[0] := \{tx : tx_s, \text{ idx} : 1\}$$

$$tx^r_{Rvk}.\text{Ins}[1] := tr_{redeem}$$

12. Set $tx^r_{Rvk} = \perp$ if $\beta = 0$.
13. **Return** $(tx_s, tx^l_{Rvk}, tx^r_{Rvk})$.

genSettleTx$(\gamma.i, lb', rb')$:

1. Lookup the corresponding channel γ based on $\gamma.id$.
2. Define the settlement output, which finalizes the payment.

$$\theta_0 := \left\{ \text{amt} : lb', \quad \phi : \text{Sig}(\gamma.lu) \right\}$$

3. Define the paid output for the counterparty:

$$\theta_1 := \left\{ \text{amt} : rb', \quad \phi : \text{Sig}(\gamma.ru) \right\}$$

4. Generate the settle transaction:

$$tx_{\texttt{settle}}.\text{Outs} := \text{genState}\left(\gamma, \{\theta_0, \theta_1\}\right)$$

5. **Return** $tx_{\texttt{settle}}$.

check2SR$(\gamma, \alpha, \beta, T, tr_{rollback}, tr_{redeem}, tx_s, tx^l_{Rvk}, tx_{Rvk})$:

1. Check if tx_s, tx^l_{Rvk} and tx^r_{Rvk} are correctly constructed Using function gen2SRTx.
2. Check if U_i provides the correct signature over the revoke transactions.
3. **If** all passes, **return** $\top$, **else return** $\bot$.

F UC proof and security analysis

We utilize the global UC framework (GUC) [11] in our formal security analysis. Compare with standard UC, GUC enables a global setup, allowing protocol to interact with pre-configured global functionalities. The definition of GUC is covered in our extended prelimaires in Appendix C

F.1 Ideal Functionalities

In this section, we first shows the main functionality, then we shows the sub-functionalities, which is the corresponding functionalities for the sub-protocols used on the main protocol. With these sub-protocol functionaliies, we can construct several hybrid world models to help me prove the security of the main protocol.

F.1.1 Channel states and revoke transactions

Consider a multi-hop payment $\mathcal{M}$ following the definition within Section 2 with n relays. Here we list all the possible states of each channel from γ_0 to γ_n. We first define the funds involved in *Zeus*.

- Payee's on-chain stake: $\texttt{Stake} \geq c_{\text{burn}} \approx 2.3$ USD.
- Payer's off-chain deposit in channel $\gamma_{i\in[1,n]}$: $\texttt{Dep}_i = c_{\text{payee}} + (n-i) \cdot c_{\text{relay}}$.
- Premium in channel $\gamma_{i\in[1,n]}$: $\texttt{Prm}^i = f_0 + \sum_{k=1}^{i} f_k$.
- Principal in channel $\gamma_{i\in[1,n]}$: $\texttt{Prcpl} = x + \sum_{k=1}^{n} f_k$.

For channel $\gamma_i, i \in [0,n]$, we define the following states:

- $\text{S}^i_{\texttt{IDLE}}$: we use $\text{S}^i_{\texttt{IDLE}}(lb, rb)$ to denote the initial state of γ_i where lb and rb are the initial balances of U_i and U_{i+1} respectively.

$$\theta_0 = \begin{cases} \text{amt}: & lb \\ \phi: & \text{Sig}(U_i) \end{cases} \quad \theta_1 = \begin{cases} \text{amt}: & rb \\ \phi: & \text{Sig}(U_{i+1}) \end{cases}$$

- $\text{S}^i_{\texttt{LOCKED-Dep}}$: we use $\text{S}^i_{\texttt{LOCKED-Dep}}(\texttt{Dep}_i, T)$ to denote the state of γ_i after the deposit is locked by U_i.

$$\theta_0 = \begin{cases} \text{amt}: & \texttt{Dep}_i \\ \phi: & \begin{cases} \text{Sig}(U_i) \wedge \text{Sig}(U_{i+1}) \\ \text{Sig}(U_{i+1}) \geq T \end{cases} \end{cases}$$

$$\theta_1 = \begin{cases} \text{amt}: & lb - \texttt{Dep}_i \\ \phi: & \text{Sig}(U_i) \end{cases} \quad \theta_2 = \begin{cases} \text{amt}: & rb \\ \phi: & \text{Sig}(U_{i+1}) \end{cases}$$

- $\text{S}^i_{\texttt{LOCKED-Prm}}$: we use $\text{S}^i_{\texttt{LOCKED-Prm}}(\texttt{Dep}_i, \texttt{Prm}, T)$ to denote the state of γ_i after the premium is locked by U_{i+1} over the previous $\text{S}^i_{\texttt{LOCKED-Dep}}$ state.

$$\theta_0 = \begin{cases} \text{amt}: & \texttt{Dep}_i \\ \phi: & \begin{cases} \text{Sig}(U_i) \wedge \text{Sig}(U_{i+1}) \\ \text{Sig}(U_{i+1}) \geq T \end{cases} \end{cases}$$

$$\theta_1 = \begin{cases} \text{amt}: & \texttt{Prm}_i \\ \phi: & \begin{cases} \text{Sig}(U_i) \wedge \text{Sig}(U_{i+1}) \\ \text{Sig}(U_i) \geq T + \Delta \end{cases} \end{cases}$$

$$\theta_2 = \begin{cases} \text{amt}: & lb - \texttt{Dep}_i \\ \phi: & \text{Sig}(U_i) \end{cases} \quad \theta_3 = \begin{cases} \text{amt}: & rb - \texttt{Prm}_i \\ \phi: & \text{Sig}(U_{i+1}) \end{cases}$$

- $\text{S}^i_{\texttt{LOCKED-Prcpl}}$: we use $\text{S}^i_{\texttt{LOCKED-Prcpl}}(\texttt{Dep}_i + \texttt{Prcpl}, \texttt{Prm}_i, T)$ to denote the state of γ_i after additional principal is locked by U_i over the previous $\text{S}^i_{\texttt{LOCKED-Prm}}$ state.

$$\theta_0 = \begin{cases} \text{amt}: & \texttt{Dep}_i + \texttt{Prcpl} \\ \phi: & \begin{cases} \text{Sig}(U_i) \wedge \text{Sig}(U_{i+1}) \\ \text{Sig}(U_{i+1}) \geq T \end{cases} \end{cases}$$

$$\theta_1 = \begin{cases} \text{amt}: & \texttt{Prm}_i \\ \phi: & \begin{cases} \text{Sig}(U_i) \wedge \text{Sig}(U_{i+1}) \\ \text{Sig}(U_i) \geq T + \Delta \end{cases} \end{cases}$$

$$\theta_2 = \begin{cases} \text{amt}: & lb - \texttt{Dep}_i - \texttt{Prcpl} \\ \phi: & \text{Sig}(U_i) \end{cases} \quad \theta_3 = \begin{cases} \text{amt}: & rb - \texttt{Prm}_i \\ \phi: & \text{Sig}(U_{i+1}) \end{cases}$$

- $\text{S}^i_{\texttt{REFUNDED-Dep}}$: we use $\text{S}^i_{\texttt{REFUNDED-Dep}}(\texttt{Prcpl}, \texttt{Prm}_i, T)$ to denote the state of γ_i after U_i's locked amount is reduced by $\texttt{Dep}$.

$$\theta_0 = \begin{cases} \text{amt}: & \texttt{Prcpl} \\ \phi: & \begin{cases} \text{Sig}(U_i) \wedge \text{Sig}(U_{i+1}) \\ \text{Sig}(U_{i+1}) \geq T \end{cases} \end{cases}$$

$$\theta_1 = \begin{cases} \text{amt}: & \texttt{Prm}_i \\ \phi: & \begin{cases} \text{Sig}(U_i) \wedge \text{Sig}(U_{i+1}) \\ \text{Sig}(U_i) \geq T + \Delta \end{cases} \end{cases}$$

$$\theta_2 = \begin{cases} \text{amt}: & lb - \texttt{Prcpl} \\ \phi: & \text{Sig}(U_i) \end{cases} \quad \theta_3 = \begin{cases} \text{amt}: & rb - \texttt{Prm}_i + \texttt{Dep}_i \\ \phi: & \text{Sig}(U_{i+1}) \end{cases}$$

- $\text{S}^i_{\texttt{PAID}}$: we use $\text{S}^i_{\texttt{PAID}}(\texttt{Prcpl}, \texttt{Prm}, T)$ to denote the state of γ_i after U_i gives the principal to U_{i+1} while U_{i+1} gives the premium to U_i.

$$\theta_0 = \begin{cases} \text{amt}: & lb - (x + \sum_{k=i+1}^{n} f_k) \\ \phi: & \text{Sig}(U_i) \end{cases}$$

$$\theta_1 = \begin{cases} \text{amt}: & rb + (x + \sum_{k=i+1}^{n} f_k) \\ \phi: & \text{Sig}(U_{i+1}) \end{cases}$$

We define two mappings, which map a channel's state to a valid transaction:

Left Revoke Transaction.

$$\text{RvkTx}_L : (S^i, tr^i_{\text{splicing}}) \mapsto tx_{\texttt{rvk-L}}$$

Given the state of γ_i (S^i) as input, function $\text{RvkTx}_L(S^i)$ returns a valid transaction $tx_{\texttt{rvk_L}}$ which can revoke all the tokens (payer's deposit or principal) locked by U_i once γ_0 is spliced out in state $\text{S}^0_{\texttt{SPLICED}}$, while the UTXO tr^i_{splicing} will be finalized within $\mathcal{F}_C$. This transaction takes the 0-th output of S_i and tr^i_{splicing} as inputs, along with the valid spending signatures ($\text{Sig}(U_i)$), and a single output

$$\begin{cases} \text{amt}: & \text{sum of inputs' amount} - F \\ \phi: & \text{Sig}(U_i) \end{cases}$$

Right Revoke Transaction.

$$\text{RvkTx}_R : (S^i, tr^i_{\text{redeem}}) \mapsto tx_{\texttt{rvk-R}}$$

We denote $\text{RvkTx}_R(S^i)$ as the transaction to revoke the tokens locked by U_{i+1} if the redeem trigger tr^i_{redeem} is finalized on $\mathcal{F}_L$. The revoke transaction $tx_{\texttt{rvk_R}}$ takes the 1-st output of S_i and tr^i_{redeem} as inputs, along with the valid spending signatures ($\text{Sig}(U_{i+1})$), and a single output

$$\begin{cases} \text{amt}: & \text{sum of inputs' amount} - F \\ \phi: & \text{Sig}(U_{i+1}) \end{cases}$$

F.1.2 Ideal functionality of main protocol

$\mathcal{F}_{Zeus}$

Parameters:

- $\texttt{sidSet}$: a set of all payment identifiers. Initially empty.
- For each $\texttt{sid} \in \texttt{sidSet}$, store:
  a) x: payment amount configured by payee.
  b) $\mathcal{M}$: The multi-hop payment metadata configured by the payer.
     $\mathcal{M} = (U, \Gamma, x, \text{Fees}, \mathcal{S})$, where:
     - U is the payment path, $U = (U_0, U_1, \ldots, U_n, U_{n+1})$.
     - Γ is the sequence of payment channels, $\Gamma = (\gamma_0, \ldots, \gamma_n)$, where U_i establishes channel γ_i with U_{i+1} for $i = 0$ to n.
     - Fees is the sequence of relay fees, $\text{Fees} = (f_1, \ldots, f_n)$, where f_i is the fee charged by intermediary U_i.
     - $\mathcal{S}$ is the sequence of channel states.
  c) $T, \bar{T}$: T is the principal timelock set by payer, $\bar{T}$ is the principal timelock set by payee.
  d) $\texttt{status}_i$ for $i \in [0, n+1]$: status flags for each participant.
  e) s^l_i, s^r_i: the proposed channel state for channel γ_i.
  f) $\texttt{tag}_{redeem}, \texttt{tag}_{rollback}, \texttt{tag}_{slash}$: tags for the redeem, rollback, and slash transactions. Intialized as 0.
  g) $\texttt{tag}^{i,l}_{rdm-h}, \texttt{tag}^{i,r}_{rdm-h}$: the tag for redeem handler, initialized as 0. $\texttt{tag}^{i,l}_{rbk-h}, \texttt{tag}^{i,r}_{rbk-h}$: the tag for rollback handler, initialized as 0.
  h) $tx^0_{rollback}, \ldots, tx^{n+1}_{rollback}$: the rollback transaction observed by each participant.
  i) $tx^0_{redeem}, \ldots, tx^{n+1}_{redeem}$: the redeem transaction observed by each participant.
  j) $tx^0_{slash}, \ldots, tx^{n+1}_{slash}$: the slash transaction observed by each participant.

Global Setup. Upon receiving $(\texttt{sid}, \texttt{global-setup}, \mathcal{M}, T, \texttt{Stake})$ from $\mathcal{E}$, where $\texttt{sid} \notin \texttt{sidSet}$, the functionality does the following: $\mathcal{M}$, the payment lifetime T, and the payee's stake $\texttt{Stake}$ are all known to all parties.

$(\texttt{sid}, \texttt{init}, \texttt{UTXO}_{src}) \overset{\tau}{\hookleftarrow} \mathcal{E}$:

`Setup phase`

1. If the $\texttt{Status}$ of $\texttt{sid}$ is not $\texttt{setup}$, then abort.
2. **Construct setup transactions.**
   - If U_0 is honest, **Construct and save setup transactions.** $(tx_{\text{rollback}}, tx_{\text{redeem}}, tx_{\text{slash}}) := \text{genSetupTx}(\texttt{UTXO}_{src}, \text{Out}_{\texttt{Stake}}, \mathcal{M})$. Then set all $tx^i_{rollback} = tx_{\text{rollback}}$, $tx^i_{\text{redeem}} = tx_{\text{redeem}}$, and $tx^i_{\text{slash}} = tx_{\text{slash}}$ for $i \in [0, n+1]$.
   - If U_0 is dishonest, let $\mathcal{S}$ to config the setup transactions at this round: Leak $(tx_{\text{rollback}}, tx_{\text{redeem}}, tx_{\text{slash}})$ to $\mathcal{S}$, and wait for $\mathcal{S}$ to send $(\{tx^i_{\text{rollback}}\}^{n+1}_{i=0}, \{tx^i_{\text{redeem}}\}^{n+1}_{i=0}, \{tx^i_{\text{slash}}\}^{n+1}_{i=0})$ back.
3. **Trigger the redeem launchers.** At round $\tau + 1$: If the payee U_{n+1} is honest, and tx^{n+1}_{slash}, tx^{n+1}_{redeem}, and $tx^{n+1}_{rollback}$ are valid, set $\texttt{tag}_{redeem} = 1$ and set $s^r_n = S^{\tau}_{\texttt{LOCKED-Dep}}$ at round $\tau + 1$. If U_{n+1} is corrupted, $\mathcal{S}$ can choose to set $\texttt{tag}_{redeem} = \perp$ at round $\tau + 1$. If U_{n+1} is corrupted, $\mathcal{S}$ can choose to set $\texttt{tag}_{slash} = \perp$ and set s^r_n to any states.
4. **Set up channel state.** At round $\tau + 1$: If $U_{i \in [1,n]}$ is honest, and tx^i_{slash}, tx^i_{redeem}, and $tx^i_{rollback}$ are valid, this means $tr^i_{rollback}$, $tr^{i-1}_{rollback}$ are valid outputs of $tx^i_{rollback}$. tr^{i-1}_{redeem} and tr^i_{redeem} are valid outputs of tx^i_{redeem}. Set $s^r_{i-1} = S^{i-1}_{\texttt{LOCKED-Dep}}$ at round $\tau + 2$. If $U_{i \in [1,n]}$ is corrupted, $\mathcal{S}$ can set s^r_{i-1}.
5. **Trigger the rollback and slash launchers.** At round $\tau + 2$: If U_0 is honest: (i) if $\texttt{tag}_{slash} = \perp$, terminate. (ii) if $\texttt{tag}_{slash} \neq \perp$, set $\texttt{tag}_{rollback} = 1$ and $\texttt{tag}_{slash} = 1$. If U_0 is corrupted, $\mathcal{S}$ can modify the slash tag and redeem tag.

`Lock-deposit phase`

- If U_0 is honest: **Lock payer's deposit within γ_0.** Set $s^l_0 = S^0_{\texttt{LOCKED-Dep}}$ at round $\tau + 2$. Set $\tau_0 = \tau + 2$.

- At round τ_0+1: if U_1 is honest and $s_0^l = s_0^r$, set $\mathtt{tag}_{\text{rdm-h}}^{0,r} = 1$ and $\mathtt{tag}_{\text{rbk-h}}^{0,r} = 1$. If U_1 is corrupted, let $\mathcal{S}$ to reset s_0^r to any state.
- If $s_0^l = s_0^r$ at round τ_0+2:
  - If both U_0 and U_1 are honest, γ_0 updated to $S^0_{\mathtt{LOCKED\text{-}Dep}}$ within t_{update} blocks.
  - Else if U_0 is corrupted and U_1 is honest, γ_0 can be updated at any time before γ_0 is closed.
  - Else if U_0 is honest and U_1 is corrupted, γ_0 updated to $S^0_{\mathtt{LOCKED\text{-}Dep}}$ within t_{update} blocks.
- If γ_{i-1} is updated to $S^{i-1}_{\mathtt{LOCKED\text{-}Dep}}$ at round τ_i for $i \in [1, n-1]$:
  - If U_{i-1} is honest: Set $\mathtt{tag}_{\text{rdm-h}}^{i-1,r} = 1$ and $\mathtt{tag}_{\text{rbk-h}}^{i-1,l} = 1$.
  - If U_i is honest: **Forward payer's deposit within γ_i.** Set $s_i^l = S^i_{\mathtt{LOCKED\text{-}Dep}}$ at round τ_i.
  - If $s_i^l = s_i^r$ at round τ_i+2:
    - If both U_i and U_{i+1} are honest, γ_i updated to $S^i_{\mathtt{LOCKED\text{-}Dep}}$ within t_{update} blocks.
    - If only U_{i+1} is honest, γ_i can be updated at any time before γ_i is closed.
  - If γ_i is updated to $S^i_{\mathtt{LOCKED\text{-}Dep}}$ at round τ_{i+1}, then set $s_i^l = S^i_{\mathtt{LOCKED\text{-}Prm}}$ at round τ_{i+1}.

`Lock-premium phase`

- If γ_n is updated to $S^n_{\mathtt{LOCKED\text{-}Dep}}$ at some round τ_n:
  - If U_{n+1} is honest: **Start premium locking within γ_n.** Set $s_n^r = S^n_{\mathtt{LOCKED\text{-}Prm}}$ at round τ_n.
  - At round τ_n+1: if U_n is corrupted, let $\mathcal{S}$ to reset s_n^l to any state.
  - If $s_n^l = s_n^r$ at round τ_n+2:
    - If both U_n and U_{n+1} are honest, γ_n is updated to $S^n_{\mathtt{LOCKED\text{-}Prm}}$ within t_{update} blocks.
    - If only U_n is honest, γ_n can be updated at any time before γ_n is closed.
    - Else if U_{n+1} is honest and U_n is corrupted, γ_n updated to $S^n_{\mathtt{LOCKED\text{-}Prm}}$ within t_{update} blocks.
- If γ_i is updated to $S^i_{\mathtt{LOCKED\text{-}Prm}}$ at some round τ_i' for $i \in [1, n]$:
  - If U_i is honest: **Forward premium locking within γ_{i-1}.** Set $s_{i-1}^r = S^{i-1}_{\mathtt{LOCKED\text{-}Prm}}$ at round τ_i'.
  - At round $\tau_i'+1$: if U_{i-1} is corrupted, let $\mathcal{S}$ to reset s_{i-1}^l to any state.
  - If $s_{i-1}^l = s_{i-1}^r$ at round $\tau_i'+2$:
    - If both U_{i-1} and U_i are honest, γ_{i-1} is updated to $S^{i-1}_{\mathtt{LOCKED\text{-}Prm}}$ within t_{update} blocks.
    - If only U_{i-1} is honest, γ_{i-1} can be updated at any time before it is closed.
    - Else if U_i is honest and U_{i-1} is corrupted, γ_{i-1} updated to $S^{i-1}_{\mathtt{LOCKED\text{-}Prm}}$ within t_{update} blocks.

`Lock-principal phase`

- If γ_0 is updated to $S^0_{\mathtt{LOCKED\text{-}Prm}}$ at some round τ_0':
  - If U_0 is honest: **Lock principal within γ_0.** Set $s_0^l = S^0_{\mathtt{LOCKED\text{-}Prcpl}}$ at round τ_0'.
  - At round $\tau_0'+1$: if U_1 is corrupted, let $\mathcal{S}$ to reset s_0^r to any state.
  - If $s_0^l = s_0^r$ at round $\tau_0'+2$:
    - If both U_0 and U_1 are honest, γ_0 is updated to $S^0_{\mathtt{LOCKED\text{-}Prcpl}}$ within t_{update} blocks.
    - If only U_1 is honest, γ_0 can be updated at any time before it is closed.
    - Else if U_0 is honest and U_1 is corrupted, γ_0 updated to $S^0_{\mathtt{LOCKED\text{-}Prcpl}}$ within t_{update} blocks.
- If γ_{i-1} is updated to $S^{i-1}_{\mathtt{LOCKED\text{-}Prcpl}}$ at round τ_i'' for $i \in [1, n]$:
  - If U_i is honest: **Forward principal lock within γ_i.** Set $s_i^l = S^i_{\mathtt{LOCKED\text{-}Prcpl}}$ at round τ_i''.
  - At round $\tau_i''+1$: if U_{i+1} is corrupted, let $\mathcal{S}$ to reset s_i^r to any state.
  - If $s_i^l = s_i^r$ at round $\tau_i''+2$:
    - If both U_i and U_{i+1} are honest, γ_i is updated to $S^i_{\mathtt{LOCKED\text{-}Prcpl}}$ within t_{update} blocks.
    - If only U_{i+1} is honest, γ_i can be updated at any time before it is closed.
    - Else if U_i is honest and U_{i+1} is corrupted, γ_i updated to $S^i_{\mathtt{LOCKED\text{-}Prcpl}}$ within t_{update} blocks.

`Deposit-Refund phase`

- If γ_n is updated to $S^n_{\mathtt{LOCKED\text{-}Prcpl}}$ at round τ_n'':
  - If U_{n+1} is honest: **Refund deposit within γ_n.** Set $s_n^r = S^n_{\mathtt{REFUNDED\text{-}Dep}}$ at round τ_n''.
  - At round $\tau_n''+1$: if U_n is corrupted, let $\mathcal{S}$ to reset s_n^l to any state.
  - If $s_n^l = s_n^r$ at round $\tau_n''+2$:
    - If both U_n and U_{n+1} are honest, γ_n is updated to $S^n_{\mathtt{REFUNDED\text{-}Dep}}$ within t_{update} blocks.
    - If only U_n is honest, γ_n can be updated at any time before it is closed.
    - Else if U_{n+1} is honest and U_n is corrupted, γ_n updated to $S^n_{\mathtt{REFUNDED\text{-}Dep}}$ within t_{update} blocks.
- If γ_i is updated to $S^i_{\mathtt{REFUNDED\text{-}Dep}}$ at round τ_i''' for $i \in [1, n]$:
  - If U_i is honest: **Forward deposit refund within γ_{i-1}.** Set $s_{i-1}^r = S^{i-1}_{\mathtt{REFUNDED\text{-}Dep}}$ at round τ_i'''.
  - If $s_{i-1}^l = s_{i-1}^r$ at round $\tau_i'''+2$:
    - If both U_{i-1} and U_i are honest, γ_{i-1} is updated to $S^{i-1}_{\mathtt{REFUNDED\text{-}Dep}}$ within t_{update} blocks.
    - If only U_{i-1} is honest, γ_{i-1} can be updated at any time before it is closed.
    - Else if U_i is honest and U_{i-1} is corrupted, γ_{i-1} updated to $S^{i-1}_{\mathtt{REFUNDED\text{-}Dep}}$ within t_{update} blocks.

`Settle phase`

- If γ_0 is updated to $S^0_{\mathtt{REFUNDED\text{-}Dep}}$ at round τ_0''':
  - If U_0 is honest: **Settle payment within γ_0.** Set $s_0^l = S^0_{\mathtt{PAID}}$ at round τ_0'''.

– At round $\tau_0''' + 1$: if U_1 is corrupted, let $\mathcal{S}$ to reset s_0^r to any state.
– If $s_0^l = s_0^r$ at round $\tau_0''' + 2$:
  * If both U_0 and U_1 are honest, γ_0 is updated to $S^0_{\texttt{PAID}}$ within t_{update} blocks. Then, return $(\texttt{sid}, \texttt{settled})$ to $\mathcal{E}$.
  * If only U_1 is honest, γ_0 can be updated at any time before it is closed.
  * Else if U_0 is honest and U_1 is corrupted, γ_0 updated to $S^0_{\texttt{PAID}}$ within t_{update} blocks.

- If γ_{i-1} is updated to $S^{i-1}_{\texttt{PAID}}$ at round τ''''_{i-1} for $i \in [1,n]$:
  – If U_{i-1} is honest: Reset the tags:(close the handlers in 2SR sub-protocols)
    * set the handler tags to 0: $\texttt{tag}^{i-1,l}_{\text{rdm-h}} = 0$, $\texttt{tag}^{i-1,l}_{\text{rbk-h}} = 0$.
    * if $i = 1$: set the slash and rollback tags to 0.
  – If U_i is honest: $\texttt{tag}^{i-1,r}_{\text{rdm-h}} = 0$, $\texttt{tag}^{i-1,r}_{\text{rbk-h}} = 0$.
  – If U_i is honest: **Forward settlement within γ_i.** Set $s_i^l = S^i_{\texttt{PAID}}$ at round τ''''_{i-1}.
  – At round $\tau''''_{i-1} + 1$: if U_{i+1} is corrupted, let $\mathcal{S}$ to reset s_i^r to any state.
  – If $s_i^l = s_i^r$ at round $\tau''''_i + 1$:
    * If both U_i and U_{i+1} are honest, γ_i is updated to $S^i_{\texttt{PAID}}$ within t_{update} blocks.
    * If only U_{i+1} is honest, γ_i can be updated at any time before it is closed.
    * Else if U_i is honest and U_{i+1} is corrupted, γ_i updated to $S^i_{\texttt{PAID}}$ within t_{update} blocks.

- If γ_n is updated to $S^n_{\texttt{PAID}}$: and U_{n+1} is honest: $\texttt{tag}^{n,l}_{\text{rdm-h}} = 0$, $\texttt{tag}^{n,l}_{\text{rbk-h}} = 0$, and $\texttt{tag}_{\text{redeem}} = 0$. send $(\texttt{sid}, \texttt{settled})$ to $\mathcal{E}$.

Launcher Service:
- If $\texttt{tag}_{rollback} = 1$ and U_0 is honest: at round corresponding to block height $T - t_{close} - \Delta$, $tx^0_{rollback}$ will be confirmed on $\mathcal{F}_L$. $tx^0_{rollback}$ will be inserted into block with index $T - t_{close} - 2\Delta + 1$.
- If $\texttt{tag}_{redeem} = 1$ and U_{n+1} is honest: for each round τ' up to the block height $T - t_{close} - \Delta$:
  – If transaction $tx^{n+1}_{rollback}$ is confirmed on $\mathcal{F}_L$, then tx_{redeem} is confirmed on $\mathcal{F}_L$ at round $\tau' + \Delta$.
  – At the round corresponding to block height $T - t_{close} - \Delta$, if tx_{redeem} has not been confirmed, set $\texttt{tag}_{redeem} = 0$.
- If $\texttt{tag}_{slash} = 1$ and U_0 is honest:
  – At the round corresponding to block height $T - t_{close}$, if no redeem transaction spending the output of $tx^0_{rollback}$ has been finalized on $\mathcal{F}_L$, then tx_{slash} is confirmed on $\mathcal{F}_L$ by block $T - t_{close} + \Delta$.
  – If tx^0_{slash} fails to be confirmed by $T - t_{close} + \Delta$, an alternative transaction tx' will reclaim U_0's funds in $tx^0_{rollback}$ by block $T - t_{close} + 2\Delta$.

Dispute Resolution.
1. **Rollback Service (for $U_{i\in[0,n]}$):** If $\texttt{tag}^{i,l}_{\text{rbk-h}} = 1$: At the round for block height $T - t_{close} - \Delta$, if U_i is honest and tr^i_{rollback} is finalized:
   - γ_i is closed by $T - t_{close}$.
   - If γ_i closed in state $s \in \{S^i_{\texttt{LOCKED-Dep}}, S^i_{\texttt{LOCKED-Prm}}, S^i_{\texttt{LOCKED-Prcpl}}, S^i_{\texttt{REFUNDED-Dep}}\}$, then the corresponding transaction tx^l_{Rvk} (revoking the funds locked by U_i) is confirmed on $\mathcal{F}_L$ by round T.
2. **Redeem Service (for $U_{i\in[1,n+1]}$):** If $\texttt{tag}^{i-1,r}_{\text{rdm-h}} = 1$: At the round for block height $T - t_{close} - \Delta$, if U_i is honest and tr^{i-1}_{redeem} is finalized:
   - γ_{i-1} is closed by T.
   - If γ_{i-1} closed in state $s \in \{S^{i-1}_{\texttt{LOCKED-Prm}}, S^{i-1}_{\texttt{LOCKED-Prcpl}}, S^{i-1}_{\texttt{REFUNDED-Dep}}\}$ then a corresponding transaction tx^r_{Rvk} (revoking the funds locked by U_i) is confirmed on $\mathcal{F}_L$ by round $T + \Delta$.
3. **Justice Service (Timeout Claim):**
   - If $\texttt{tag}^{i,l}_{\text{rbk-h}} = 1$: At the round for block height $T - t_{close}$, if U_i is honest: (i) γ_i is closed by T, and (ii) all funds spent by U_i in γ_i will be spent before $T + \Delta$.
   - If $\texttt{tag}^{i,r}_{\text{rdm-h}} = 1$: At the round for block height $T - t_{close} + \Delta$, if U_{i+1} is honest: (i) γ_i is closed by $T + \Delta$, and (ii) all funds spent by U_{i+1} in γ_i will be spent before $T + 2\Delta$.

F.1.3 Ideal Functionalities of launchers

$\mathcal{F}_{\text{rollback-launcher}}$

Parameters:
- $\texttt{sidSet}$: the set of all session ids. Initially empty.

$(\texttt{sid}, \texttt{init}, T, tx_{rollback}) \overset{\tau}{\hookleftarrow} \mathcal{E}, U_i$:

1) If $\texttt{sid} \in \texttt{sidSet}$, ignore. Otherwise, add $\texttt{sid}$ to $\texttt{sidSet}$ and record $(T, tx_{rollback})$.
2) If U_i is honest, then at round corresponding to block height $T - t_{close} - \Delta$, $tx_{rollback}$ will be confirmed on $\mathcal{G}_{\text{L}}$. $tx_{rollback}$ will be inserted into block with index $T - t_{close} - 2\Delta + 1$.

$(\texttt{sid}, \texttt{CLOSE}) \overset{\tau^* \geq \tau}{\longleftarrow} \mathcal{E}, U_i$:

3) If U_i is honest, terminate this instance.

$\mathcal{F}_{\text{redeem-launcher}}$

Parameters:
- $\texttt{sidSet}$: the set of all session ids. Initially empty.

$(\texttt{sid}, \texttt{init}, T, tx_{rollback}, tx_{redeem}) \overset{\tau}{\hookleftarrow} \mathcal{E}, U_{n+1}$:

1) If $\texttt{sid} \in \texttt{sidSet}$, ignore. Otherwise, add $\texttt{sid}$ to $\texttt{sidSet}$ and record $(T, tx_{rollback}, tx_{redeem})$.
2) If U_{n+1} is honest, then for each round τ' up to the block height $T - t_{close} - \Delta$:
   - If transaction $tx_{rollback}$ is confirmed on $\mathcal{G}_{\text{L}}$, then tx_{redeem} is confirmed on $\mathcal{G}_{\text{L}}$ at round $\tau' + \Delta$.
3) If U_{n+1} is honest, terminate this instance at the round corresponding to block height $T - t_{close} - \Delta$.

$\mathcal{F}_{\text{slash-launcher}}$

Parameters:
- $\texttt{sidSet}$: the set of all session ids. Initially empty.

$(\texttt{sid}, \texttt{init}, T, tx_{rollback}, tx_{slash}) \overset{\tau}{\hookleftarrow} \mathcal{E}, U_0$:

1) If $\texttt{sid} \in \texttt{sidSet}$, ignore. Otherwise, add $\texttt{sid}$ to $\texttt{sidSet}$ and record $(T, tx_{rollback}, tx_{slash})$.
2) If U_0 is honest, then:

- At the round corresponding to block height $T - t_{close}$, if no redeem transaction spending the output of $tx_{rollback}$ has been finalized on $\mathcal{G}_{\mathrm{L}}$, then tx_{slash} is confirmed on $\mathcal{G}_{\mathrm{L}}$ by block $T - t_{close} + \Delta$.
- If tx_{slash} fails to be confirmed by $T - t_{close} + \Delta$, an alternative transaction tx' will reclaim U_0's funds in $tx_{rllback}$ by block $T - t_{close} + 2\Delta$.

F.1.4 Ideal functionality of handlers

$\mathcal{F}^i_{\text{rollback-handler}}$

Parameters:

- `sidSet`: the set of all session ids. Initially empty.
- A map `sessionData` to store session parameters.

$(\mathtt{sid}, \mathtt{Setup}, \gamma_i, T, tr_{rollback}, tx^l_{Rvk}) \xleftarrow{\tau} \tilde{U}_i$, for party U_i:

1) If $\mathtt{sid} \in \mathtt{sidSet}$, ignore. Otherwise, add `sid` to `sidSet` and record $(\gamma_i, T, tr_{rollback}, tx^l_{Rvk}, U_i)$ in `sessionData`.
2) If U_i is honest, then at the round corresponding to block height $T - t_{close} - \Delta$:
   - If transaction $tr_{rollback}$ is confirmed on $\mathcal{G}_{\mathrm{L}}$, then $\mathcal{F}^i_{\text{rollback-handler}}$ will ensure that the channel γ_i is closed and the transaction tx^l_{Rvk} is confirmed on $\mathcal{G}_{\mathrm{L}}$ by round T, provided that tx^l_{Rvk} is a valid transaction against the latest state of γ_i.

$(\mathtt{sid}, \mathtt{Setup}, \gamma_i, T) \xleftarrow{\tau} \tilde{U}_{i+1}$, for party U_{i+1}:

1) If $\mathtt{sid} \in \mathtt{sidSet}$, ignore. Otherwise, record that U_{i+1} is associated with session `sid`. Save the channel γ_i and the time T in `sessionData`.
2) At the round corresponding to block height $T - t_{close}$ given honest U_{i+1}:
   - If the channel has not been closed, $\mathcal{F}^i_{\text{rollback-handler}}$ will ensure the channel is closed by block T.
   - By the round corresponding to block T, a transaction tx_{valid} spending these funds to U_{i+1}'s counterparty is confirmed on $\mathcal{G}_{\mathrm{L}}$.

$\mathcal{F}^i_{\text{redeem-handler}}$

Parameters:

- `sidSet`: the set of all session ids. Initially empty.
- A map `sessionData` to store session parameters.

$(\mathtt{sid}, \mathtt{Setup}, \gamma_i, T, tr_{redeem}, tx^r_{Rvk}) \xleftarrow{\tau} \tilde{U}_{i+1}$, for party U_{i+1}:

1) If $\mathtt{sid} \in \mathtt{sidSet}$, ignore. Otherwise, add `sid` to `sidSet` and record $(\gamma_i, T, tr_{redeem}, tx^r_{Rvk}, U_{i+1})$ in `sessionData`.
2) If U_{i+1} is honest, then at the round corresponding to block height $T - t_{close} - \Delta$:
   - If transaction tr_{redeem} is confirmed on $\mathcal{G}_{\mathrm{L}}$, then $\mathcal{F}^i_{\text{redeem-handler}}$ will ensure that the channel γ_i is closed and the transaction tx^r_{Rvk} is confirmed on $\mathcal{G}_{\mathrm{L}}$ by round T, provided that tx^r_{Rvk} is a valid transaction against the latest state of γ_i.

$(\mathtt{sid}, \mathtt{Setup}, \gamma_i, T) \xleftarrow{\tau} \tilde{U}_i$, for party U_i:

1) If $\mathtt{sid} \in \mathtt{sidSet}$, ignore. Otherwise, record that U_i is associated with session `sid`. Save the channel γ_i and the time T in `sessionData`.
2) At the round corresponding to block height $T - t_{close} + \Delta$ given honest U_i:
   - If the channel has not been closed, $\mathcal{F}^i_{\text{redeem-handler}}$ will ensure the channel is closed.
   - By the round corresponding to block $T + \Delta$, a transaction tx_{valid} spending these funds to U_i's counterparty is confirmed on $\mathcal{G}_{\mathrm{L}}$.

F.1.5 Ideal functionality of 2 stage revocale channel update

$\mathcal{F}^i_{\text{2SR}}$

Parameters:

- A map `sessionData` to store session parameters and status.
- $\mathcal{F}^i_{\text{2SR}}$ is parameterized by the channel γ_i between U_i and U_{i+1}.

$(\mathtt{sid}, \mathtt{setup}, \mathtt{Meta}_R) \xleftarrow{\leq\tau} \tilde{U}_{i+1}$:

1) If a session with `sid` exists, ignore. Otherwise, create a new session entry in `sessionData`, record $\mathtt{Meta}_R := (\gamma_i, \alpha, \beta, T, tr^i_{\text{rollback}}, tr^i_{\text{redeem}})$ from U_{i+1}, and set $\mathtt{status} := \mathtt{setup}$.

$(\mathtt{sid}, \mathtt{init}, \mathtt{Meta}_L) \xleftarrow{\tau} \tilde{U}_i$:

2) Record $\mathtt{Meta}_L := (\gamma_i, \alpha, \beta, T, tr^i_{\text{rollback}}, tr^i_{\text{redeem}})$ from U_i, set $\mathtt{status} := \mathtt{initialized}$, and leak $(\mathtt{sid}, \mathtt{init}, \mathtt{Meta}_L)$ to the simulator $\mathcal{S}$.
3) At round $\tau + 1$:
   - If U_{i+1} is corrupted, wait for a $(\mathtt{sid}, \mathtt{Met\bar{a}}_R)$ message from $\mathcal{S}$. Upon receiving this message, set $\mathtt{Meta}_R := \mathtt{Met\bar{a}}_R$.
   - Check if $\mathtt{Meta}_L = \mathtt{Meta}_R$. If not, set $\mathtt{status} := \mathtt{aborted}$ and terminate.
   - If they match, set $\mathtt{status} := \mathtt{agreed}$. Leak $(\mathtt{sid}, \mathtt{agreed})$ to $\mathcal{S}$.
   - Generate the corresponding state transaction tx_s based on the agreed metadata.
4) At round $\tau + 2$,if $\mathtt{status} = \mathtt{agreed}$:
   - U_i is honest: $\mathcal{F}_{Channels}$ confirms the channel update with message $(\mathtt{sid}, \mathtt{UPDATE}, \gamma_i.id, tx_s)$. If a `block` message is received, the update will never be confirmed.
   - If U_i is corrupt, then confirmation occurs only upon receiving $(\mathtt{sid}, \mathtt{broadcast})$ from $\mathcal{S}$ prior to channel closure; otherwise never updated.

Upon receiving $(\mathtt{sid}, \mathtt{UPDATED}, \gamma_i.\mathrm{id}, tx_s)$ from $\mathcal{F}_{Channels}$:

5) Set $\mathtt{status} := \mathtt{updated}$. Output $(\mathtt{sid}, \mathtt{Update\text{-}Ok}, tx_s)$ to $\tilde{U}_i$ and $\tilde{U}_{i+1}$.

For an `updated` session, the functionality guarantees:

6) **Rollback Service (for U_i):** At the round for block height $T - t_{close} - \Delta$, if tr^i_{rollback} is confirmed on $\mathcal{G}_{\mathrm{L}}$ and U_i is honest, $\mathcal{F}^i_{\text{2SR}}$ ensures that γ_i is closed and the transaction tx^l_{Rvk} (revoking the α funds to U_i) is confirmed on $\mathcal{G}_{\mathrm{L}}$ by round T, provided tx^l_{Rvk} is valid against state tx_s.
7) **Redeem Service (for U_{i+1}):** At the round for block height $T - t_{close}$, if tr^i_{redeem} is confirmed on $\mathcal{G}_{\mathrm{L}}$ and U_{i+1} is honest, $\mathcal{F}^i_{\text{2SR}}$ ensures that γ_i is closed and the transaction tx^r_{Rvk} (revoking the β funds to U_{i+1}) is confirmed on $\mathcal{G}_{\mathrm{L}}$ by round $T + \Delta$, provided tx^r_{Rvk} is valid against state tx_s.
8) **Justice Service (Timeout Claim):**
   - At the round for block height $T - t_{close}$, if U_{i+1} is honest, the channel has not been closed, and the α-output from tx_s has not been spent, $\mathcal{F}^i_{\text{2SR}}$ ensures the channel is closed and a transaction spending the α funds to U_{i+1} is confirmed on $\mathcal{G}_{\mathrm{L}}$ by block T.
   - At the round for block height $T - t_{close} + \Delta$, if U_i is honest, the channel has not been closed, and the β-output from tx_s has not been spent, $\mathcal{F}^i_{\text{2SR}}$ ensures the channel is closed and a transaction spending the β funds to U_i is confirmed on $\mathcal{G}_{\mathrm{L}}$ by block $T + \Delta$.

$\mathcal{F}^i_{\text{R-2SR}}$

Parameters:

- A map `sessionData` to store session parameters and status.
- $\mathcal{F}^i_{\text{R-2SR}}$ is parameterized by the channel γ_i between U_i and U_{i+1}.

$(\texttt{sid}, \texttt{setup}, \texttt{Meta}_L) \xleftarrow{\leq\tau} \tilde{U}_i$:

1) If a session with `sid` exists, ignore. Otherwise, create a new session entry in `sessionData`, record $\texttt{Meta}_L := (\gamma_i, \alpha, \beta, T, tr^i_{\text{rollback}}, tr^i_{\text{redeem}})$ from U_i, and set $\texttt{status} := \texttt{setup}$.

$(\texttt{sid}, \texttt{init}, \texttt{Meta}_R) \xleftarrow{\tau} \tilde{U}_{i+1}$:

2) Record $\texttt{Meta}_R := (\gamma_i, \alpha, \beta, T, tr^i_{\text{rollback}}, tr^i_{\text{redeem}})$ from U_{i+1}, set $\texttt{status} := \texttt{initialized}$, and leak $(\texttt{sid}, \texttt{init}, \texttt{Meta}_R)$ to the simulator $\mathcal{S}$.

3) At round $\tau + 1$:

   - If U_i is corrupted, wait for a $(\texttt{sid}, \tilde{\texttt{Meta}}_L)$ message from $\mathcal{S}$. Upon receiving this message, set $\texttt{Meta}_L := \tilde{\texttt{Meta}}_L$.
   - Check if $\texttt{Meta}_L = \texttt{Meta}_R$. If not, set $\texttt{status} := \texttt{aborted}$ and terminate.
   - If they match, set $\texttt{status} := \texttt{agreed}$. Leak $(\texttt{sid}, \texttt{agreed})$ to $\mathcal{S}$.
   - Generate the corresponding state transaction tx_s based on the agreed metadata.

4) At round $\tau + 2$,if $\texttt{status} = \texttt{agreed}$:

   - U_{i+1} is honest: $\mathcal{F}_{Channels}$ confirms the channel update with message $(\texttt{sid}, \texttt{UPDATE}, \gamma_i.id, tx_s)$. If a `block` message is received, the update will never be confirmed.
   - If U_{i+1} is corrupt, then confirmation occurs only upon receiving $(\texttt{sid}, \texttt{broadcast})$ from $\mathcal{S}$ prior to channel closure; otherwise never updated.

Upon receiving $(\texttt{sid}, \texttt{UPDATED}, \gamma_i.\text{id}, tx_s)$ from $\mathcal{F}_{Channels}$:

5) Set $\texttt{status} := \texttt{updated}$. Output $(\texttt{sid}, \texttt{Update-Ok}, tx_s)$ to $\tilde{U}_i$ and $\tilde{U}_{i+1}$.

For an `updated` session, the functionality guarantees:

6) **Rollback Service (for U_i):** At the round for block height $T - t_{close} - \Delta$, if tr^i_{rollback} is confirmed on $\mathcal{G}_{\text{L}}$ and U_i is honest, $\mathcal{F}^i_{\text{R-2SR}}$ ensures that γ_i is closed and the transaction tx^l_{Rvk} (revoking the α funds to U_i) is confirmed on $\mathcal{G}_{\text{L}}$ by round T, provided tx^l_{Rvk} is valid against state tx_s.

7) **Redeem Service (for U_{i+1}):** At the round for block height $T - t_{close} - \Delta$, if tr^i_{redeem} is confirmed on $\mathcal{G}_{\text{L}}$ and U_{i+1} is honest, $\mathcal{F}^i_{\text{R-2SR}}$ ensures that γ_i is closed and the transaction tx^r_{Rvk} (revoking the β funds to U_{i+1}) is confirmed on $\mathcal{G}_{\text{L}}$ by round T, provided tx^r_{Rvk} is valid against state tx_s.

8) **Justice Service (Timeout Claim):**

   - At the round for block height $T - t_{close}$, if U_i is honest, the channel has not been closed, and the β-output from tx_s has not been spent, $\mathcal{F}^i_{\text{R-2SR}}$ ensures the channel is closed and a transaction spending the β funds to U_i is confirmed on $\mathcal{G}_{\text{L}}$ by block T.
   - At the round for block height $T - t_{close} + \Delta$, if U_{i+1} is honest, the channel has not been closed, and the α-output from tx_s has not been spent, $\mathcal{F}^i_{\text{R-2SR}}$ ensures the channel is closed and a transaction spending the α funds to U_{i+1} is confirmed on $\mathcal{G}_{\text{L}}$ by block $T + \Delta$.

$\mathcal{F}^i_{\text{2pSettle}}$

Parameters:

- A map `sessionData` to store session parameters and status.
- $\mathcal{F}^i_{\text{2pSettle}}$ is parameterized by the channel γ_i between U_i and U_{i+1}.

$(\texttt{sid}, \texttt{setup}, lb', rb') \xleftarrow{\leq\tau} \tilde{U}_{i+1}$:

1) If a session with `sid` exists, ignore. Otherwise, create a new session entry in `sessionData`, record the expected balances (lb', rb') from U_{i+1}, and set $\texttt{status} := \texttt{setup}$.

$(\texttt{sid}, \texttt{init}, \gamma_i, lb', rb') \xleftarrow{\tau} \tilde{U}_i$:

2) If no session with `sid` exists or its status is not `setup`, ignore. Otherwise, retrieve the follower's balances (lb'_{i+1}, rb'_{i+1}) and compare with the initiator's balances (lb'_i, rb'_i).

3) At round $\tau + 1$:

   - If U_{i+1} is corrupted, allow $\mathcal{S}$ to modify the right balances by sending a message $(\texttt{sid}, \bar{lb}'_{i+1}, \bar{rb}'_{i+1})$. Set $(lb'_{i+1}, rb'_{i+1}) := (\bar{lb}'_{i+1}, \bar{rb}'_{i+1})$.
   - If $(lb'_i, rb'_i) \neq (lb'_{i+1}, rb'_{i+1})$, set $\texttt{status} := \texttt{aborted}$ and terminate.
   - If they match, set $\texttt{status} := \texttt{agreed}$. Leak $(\texttt{sid}, \texttt{agreed})$ to the simulator $\mathcal{S}$.
   - Generate the corresponding settlement transaction $tx_{\texttt{settle}}$ based on the agreed balances.

4) At round $\tau + 2$,if $\texttt{status} = \texttt{agreed}$:

   - U_i is honest: $\mathcal{F}_{Channels}$ confirms the channel update with message $(\texttt{sid}, \texttt{UPDATE}, \gamma_i.id, tx_s)$. If a `block` message is received, the update will never be confirmed.
   - If U_i is corrupt, then confirmation occurs only upon receiving $(\texttt{sid}, \texttt{broadcast})$ from $\mathcal{S}$ prior to channel closure; otherwise never updated.

Upon receiving $(\texttt{sid}, \texttt{UPDATED}, \gamma_i.\text{id}, tx_{\texttt{settle}})$ from $\mathcal{F}_{Channels}$:

5) Output $(\texttt{sid}, \texttt{settle-Ok})$ to both $\tilde{U}_i$ and $\tilde{U}_{i+1}$.

F.2 GUC Realization

In this section, we first show why the sub-protocols GUC-realize their corresponding ideal functionalities. Then we replace the sub-protocols with their ideal functionalities and prove the GUC-realization of the main protocol.

F.2.1 UC Proof for sub-launchers.

In our main protocol, U_0 launches the rollback and slash launchers, while U_{n+1} launches the redeem launchers. These launchers only involve one participant and only interact with the ledger functionality $\mathcal{G}_{\text{L}}$. To this end, we formalize their corresponding functionalities with $\mathcal{F}_{\text{rollback-launcher}}$, $\mathcal{F}_{\text{redeem-launcher}}$, and $\mathcal{F}_{\text{slash-launcher}}$. These functionalities outline the results of the launchers, given that the involved party is honest. Therefore, we can directly produce the following lemmas:

Lemma 3. *The protocol $\Pi_{\text{rollback-launcher}}$ GUC-realizes $\mathcal{F}_{\text{rollback-launcher}}$ in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world.*

Lemma 4. *The protocol $\Pi_{\text{redeem-launcher}}$ GUC-realizes $\mathcal{F}_{\text{redeem-launcher}}$ in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world.*

Lemma 5. *The protocol $\Pi_{\text{slash-launcher}}$ GUC-realizes $\mathcal{F}_{\text{slash-launcher}}$ in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world.*

F.2.2 UC Proof for sub-handlers.

Lemma 6. *The protocol $\Pi^i_{\text{rollback-handler}}$ GUC-realizes $\mathcal{F}^i_{\text{rollback-handler}}$ in the $\mathcal{F}_{prelim}$-hybrid world.*

Proof. We construct a simulator $\mathcal{S}$ for an arbitrary PPT adversary $\mathcal{A}$ such that no environment $\mathcal{E}$ can distinguish between the real world (interacting with $\mathcal{A}$ and $\Pi^i_{\text{rollback-handler}}$) and the ideal world (interacting with $\mathcal{S}$ and $\mathcal{F}^i_{\text{rollback-handler}}$).

Simulator for $\Pi^i_{\text{rollback-handler}}$

The simulator $\mathcal{S}$ runs a copy of $\mathcal{A}$ and handles its communications. Note that in the protocol, U_i and U_{i+1} do not interact with each other. Therefore, we can consider the corruptions separately.
Case 1: U_i is honest. When the environment $\mathcal{E}$ activates the honest U_i via $\tilde{U}_i$ with $(\texttt{sid}, \texttt{Setup}, \gamma_i, T, tr_{rollback}, tx^l_{Rvk})$, $\mathcal{S}$ forwards this exact message to $\mathcal{F}^i_{\text{rollback-handler}}$. Since U_i is honest, $\mathcal{F}^i_{\text{rollback-handler}}$ guarantees the specified behavior (e.g., closing the channel if needed). In the real protocol, the honest U_i would perform these actions by interacting with $\mathcal{G}_{\text{L}}$. The ideal functionality abstracts this interaction. The view of $\mathcal{E}$ is identical as it only sees the initial activation and the final outcome guaranteed by the functionality.
Case 2: U_i is corrupt. The adversary $\mathcal{A}$ controls U_i. Any action taken by the corrupt U_i in the protocol $\Pi^i_{\text{rollback-handler}}$ consists of sending transactions to the ledger functionality $\mathcal{G}_{\text{L}}$. $\mathcal{S}$ simply acts as a passthrough, forwarding any messages from $\mathcal{A}$ (as U_i) to $\mathcal{G}_{\text{L}}$. The view of $\mathcal{E}$ is indistinguishable because in both worlds, the corrupt party's actions are just ledger interactions, and no security guarantees are provided for it.
Case 3: U_{i+1} is honest. Upon receiving the `Setup` message from U_{i+1}, $\mathcal{S}$ simply executes the protocol honestly. The outcome is indistinguishable from the real protocol execution: U_{i+1}'s spendable (by block T) balance will be claimed by U_{i+1}.
Case 4: U_{i+1} is corrupt. For any action taken by the corrupt U_{i+1} in the protocol $\Pi^i_{\text{rollback-handler}}$, $\mathcal{S}$ simply acts as a passthrough, forwarding any messages from $\mathcal{A}$ (as U_{i+1}) to $\mathcal{G}_{\text{L}}$. Meanwhile, the functionality does not enforce any security guarantees for the corrupt party U_{i+1}. Therefore, the view of $\mathcal{E}$ is indistinguishable because in both worlds, the corrupt party's actions are just ledger interactions, and no security guarantees are provided for it.

In all cases, the environment's view is computationally indistinguishable between the real and ideal worlds. □

Symmetrical to that of Lemma 6, we can also construct a simulator $\mathcal{S}_{\text{redeem}}$ that handles the four different corruption cases for the redeem handler $\Pi^i_{\text{redeem-handler}}$ and the ideal functionality $\mathcal{F}^i_{\text{redeem-handler}}$. Therefore, we have:

Lemma 7. *The protocol $\Pi^i_{\text{redeem-handler}}$ GUC-realizes $\mathcal{F}^i_{\text{redeem-handler}}$ in the $\mathcal{F}_{prelim}$-hybrid world.*

F.2.3 UC proof for (Reversed) 2SR protocols.

Now we prove the GUC-realization of the 2SR protocol Π^i_{2SR} and the R-2SR protocol $\Pi^i_{\text{R-2SR}}$ with the ideal functionalities $\mathcal{F}^i_{\text{2SR}}$ and $\mathcal{F}^i_{\text{R-2SR}}$, respectively. We prove this with a middle hybrid world (H1) by replacing the sub-handlers with their corresponding ideal functionalities. We first show the realization in world H1, and then based on Lemma 7 and Lemma 6, we can show the realization in the $\mathcal{F}_{\text{prelim}}$-hybrid world.

Lemma 8. *Let Σ be an EUF-CMA secure signature scheme. The protocol Π^i_{2SR} GUC-realizes the ideal functionality $\mathcal{F}^i_{\text{2SR}}$ in the $\{\mathcal{F}_{prelim}, \mathcal{F}^i_{\text{rollback-handler}}, \mathcal{F}^i_{\text{redeem-handler}}\}$-hybrid world.*

Proof. We construct a simulator $\mathcal{S}$ for an arbitrary PPT adversary $\mathcal{A}$ such that no environment $\mathcal{E}$ can distinguish between an interaction with $\mathcal{A}$ and Π^i_{2SR} in the $\{\mathcal{F}_{\text{prelim}}, \mathcal{F}^i_{\text{rollback-handler}}, \mathcal{F}^i_{\text{redeem-handler}}\}$-hybrid world, and an interaction with $\mathcal{S}$ and the ideal functionality $\mathcal{F}^i_{\text{2SR}}$. The proof proceeds by analyzing the four possible corruption scenarios for parties U_i and U_{i+1}.

Simulator for Π^i_{2SR}

The simulator $\mathcal{S}$ internally runs a copy of the adversary $\mathcal{A}$. It manages all communications between $\mathcal{A}$ and the environment $\mathcal{E}$, and simulates the behavior of honest parties and the hybrid functionalities for $\mathcal{A}$. For each honest party it simulates, $\mathcal{S}$ generates the necessary cryptographic keys.
Case 1: Both U_i and U_{i+1} are honest. In this case, $\mathcal{S}$ forwards all messages between the honest parties and $\mathcal{E}$. Since no parties are corrupt, the view of $\mathcal{E}$ is identical to a real execution.

Case 2: U_i is corrupt, U_{i+1} is honest. $\mathcal{S}$ simulates the honest party U_{i+1} for the corrupted U_i (controlled by $\mathcal{A}$).

- **Setup Phase:** When $\mathcal{E}$ activates the (simulated) honest U_{i+1} with $(\texttt{sid}, \texttt{setup}, \texttt{Meta}_R)$, $\mathcal{S}$ forwards this to $\mathcal{F}^i_{\text{2SR}}$.
- **Request and Approval:** When $\mathcal{A}$ sends a request, $\mathcal{S}$ extracts $\texttt{Meta}_L$, forwards it to $\mathcal{F}^i_{\text{2SR}}$, and waits for the $(\texttt{sid}, \texttt{agreed})$ leak. Upon receipt, $\mathcal{S}$ validates the proposed transactions via 'check2SRTx'. If valid, it generates the necessary signatures for U_{i+1} and sends the 'approved' message to $\mathcal{A}$. If invalid, it aborts, which is indistinguishable from an honest party aborting.
- **Update Phase:** If $\mathcal{A}$ broadcasts the fully signed tx_s to $\mathcal{F}_{Channels}$, $\mathcal{S}$ observes this and sends $(\texttt{sid}, \texttt{broadcast})$ to $\mathcal{F}^i_{\text{2SR}}$. This triggers the confirmation in the ideal world, perfectly aligning it with the real world.
- **Post-Update:** Any subsequent interaction with handlers is already in the ideal world for this hybrid proof, so $\mathcal{S}$ simply forwards calls to the handler functionalities.

The view of $\mathcal{E}$ is indistinguishable. The simulation is sound because $\mathcal{S}$ only provides signatures after the ideal functionality has confirmed agreement on the metadata.

Case 3: U_i is honest, U_{i+1} is corrupt. $\mathcal{S}$ simulates the honest party U_i for the corrupted U_{i+1} (controlled by $\mathcal{A}$).

- **Setup Phase:** When $\mathcal{A}$ provides $(\texttt{sid}, \texttt{setup}, \texttt{Meta}_R)$, $\mathcal{S}$ forwards it to $\mathcal{F}^i_{\text{2SR}}$.
- **Init and Approval:** When $\mathcal{E}$ activates the honest U_i with $(\texttt{sid}, \texttt{init}, \texttt{Meta}_L)$, $\mathcal{S}$ sends this to $\mathcal{F}^i_{\text{2SR}}$ and sends the corresponding real protocol request to $\mathcal{A}$. If $\mathcal{A}$ returns a valid 'approved' message, $\mathcal{S}$ (as U_i) broadcasts the fully signed tx_s at round $\tau+2$, matching the auto-confirmation time of the functionality for an honest initiator. If $\mathcal{A}$ withholds approval, no broadcast occurs, and the lack of an 'UPDATED' event from $\mathcal{F}_{Channels}$ prevents the delivery of 'Update-Ok' in the ideal world, which is indistinguishable from a real-world timeout.

Indistinguishability holds because the honest U_i's behavior is deterministic and perfectly simulated.

Case 4: Both U_i and U_{i+1} are corrupt. $\mathcal{S}$ acts as a passthrough. If $\mathcal{A}$ broadcasts a valid tx_s, $\mathcal{S}$ extracts the parameters, injects the corresponding 'setup' and 'init' messages into $\mathcal{F}^i_{\text{2SR}}$ to reach the 'agreed' state, and then sends the '(sid, broadcast)' message to trigger the confirmation, ensuring the ideal world mirrors the real one.

Indistinguishability Analysis: In all cases, the view of the

environment $\mathcal{E}$ is computationally indistinguishable from a real execution.

- **Message Flows:** As shown in the case analysis, $\mathcal{S}$ generates messages that are computationally indistinguishable from those generated by honest parties in the real protocol.
- **Security Guarantees:** The EUF-CMA security of Σ ensures that $\mathcal{A}$ cannot forge signatures for honest parties, restricting its actions to the protocol paths. $\mathcal{S}$ correctly simulates the honest party responses for all such valid paths.
- **Timeliness:** The simulation correctly models timeouts, leading to indistinguishable outcomes when an adversary deviates.
- **Post-Update Security:** The security guarantees after a successful update are correctly delegated to the handler functionalities.

Therefore, no environment $\mathcal{E}$ can distinguish between the real and ideal worlds, which concludes the proof. □

Lemma 9. *Let Σ be an EUF-CMA secure signature scheme. The protocol Π^i_{2SR} GUC-realizes the ideal functionality $\mathcal{F}^i_{\text{2SR}}$ in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world.*

Proof. The proof follows directly from the UC Composition Theorem.

1. In Lemma 8, we proved that Π^i_{2SR} GUC-realizes $\mathcal{F}^i_{\text{2SR}}$ in the $\{\mathcal{F}_{\texttt{prelim}}, \mathcal{F}^i_{\text{rollback-handler}}, \mathcal{F}^i_{\text{redeem-handler}}\}$-hybrid world.
2. In Lemmas 6 and 7, we proved that the protocols $\Pi^i_{\text{rollback-handler}}$ and $\Pi^i_{\text{redeem-handler}}$ GUC-realize their respective ideal functionalities $\mathcal{F}^i_{\text{rollback-handler}}$ and $\mathcal{F}^i_{\text{redeem-handler}}$ in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world.

By the UC Composition Theorem, we can replace the ideal functionalities $\mathcal{F}^i_{\text{rollback-handler}}$ and $\mathcal{F}^i_{\text{redeem-handler}}$ used in the hybrid proof of Lemma 8 with their securely realizing protocols. Therefore, the protocol Π^i_{2SR} GUC-realizes $\mathcal{F}^i_{\text{2SR}}$ in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world. □

Symmetrically, we have:

Lemma 10. *Let Σ be an EUF-CMA secure signature scheme. The protocol $\Pi^i_{\text{R-2SR}}$ GUC-realizes the ideal functionality $\mathcal{F}^i_{\text{R-2SR}}$ in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world.*

F.2.4 UC proof for Settlement Phase.

In this part, we prove the GUC-realization of the fast confirmation protocol Π^i_{2pSettle} with the ideal functionality $\mathcal{F}^i_{\text{2pSettle}}$.

Lemma 11. *Let Σ be an EUF-CMA secure signature scheme. The protocol Π^i_{2pSettle} GUC-realizes the ideal functionality $\mathcal{F}^i_{\text{2pSettle}}$ in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world.*

Proof. We construct a simulator $\mathcal{S}$ for an arbitrary PPT adversary $\mathcal{A}$ such that no environment $\mathcal{E}$ can distinguish between an interaction with $\mathcal{A}$ and Π^i_{2pSettle} in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world, and an interaction with $\mathcal{S}$ and the ideal functionality $\mathcal{F}^i_{\text{2pSettle}}$. The proof proceeds by analyzing the four possible corruption scenarios for parties U_i and U_{i+1} under a static corruption model.

Simulator for Π^i_{2pSettle}

The simulator $\mathcal{S}$ internally runs a copy of the adversary $\mathcal{A}$. It manages all communications between $\mathcal{A}$ and the environment $\mathcal{E}$, and simulates the behavior of honest parties for $\mathcal{A}$. For each honest party it simulates, $\mathcal{S}$ generates the necessary cryptographic keys.

Case 1: Both U_i and U_{i+1} are honest. In this case, $\mathcal{S}$ simply forwards all activation messages from $\mathcal{E}$ to the ideal functionality $\mathcal{F}^i_{\text{2pSettle}}$. The functionality's outputs to the honest parties are forwarded back to $\mathcal{E}$. Since no parties are corrupt, the adversary $\mathcal{A}$ has no influence. The view of $\mathcal{E}$ is identical to a real execution where honest parties interact, and thus indistinguishable from the ideal execution.

Case 2: U_i is corrupt, U_{i+1} is honest. $\mathcal{S}$ simulates the honest party U_{i+1} for the corrupted U_i (controlled by $\mathcal{A}$).

- **Setup Phase:** When $\mathcal{E}$ activates the (simulated) honest U_{i+1} with $(\texttt{sid}, \texttt{setup}, lb', rb')$, $\mathcal{S}$ forwards this to $\mathcal{F}^i_{\text{2pSettle}}$.
- **Request and Approval:** When $\mathcal{A}$ (as U_i) sends a $(\texttt{sid}, \texttt{request-settle}, \gamma_i.id, tx^i_{\texttt{settle}})$ message, $\mathcal{S}$ extracts the balances (lb'_i, rb'_i) and sends $(\texttt{sid}, \texttt{init}, \gamma_i, lb'_i, rb'_i)$ to $\mathcal{F}^i_{\text{2pSettle}}$. $\mathcal{S}$ then waits. If the balances match, $\mathcal{F}^i_{\text{2pSettle}}$ leaks $(\texttt{sid}, \texttt{agreed})$. Upon receiving this leak, $\mathcal{S}$ generates the follower's signature $\sigma^{i,r}_{\texttt{settle}}$ and sends it to $\mathcal{A}$ in an $(\texttt{sid}, \texttt{approved-settle}, \ldots)$ message. If no 'agreed' leak occurs (due to mismatch), $\mathcal{S}$ sends nothing, which is indistinguishable from an honest follower aborting.
- **Update Phase:** If $\mathcal{A}$ broadcasts the fully signed $tx^i_{\texttt{settle}}$ to $\mathcal{F}_{\textit{Channels}}$, $\mathcal{S}$ observes this and sends $(\texttt{sid}, \texttt{broadcast})$ to $\mathcal{F}^i_{\text{2pSettle}}$. Per its rules for a corrupt initiator, the functionality then confirms the update. If $\mathcal{A}$ never broadcasts, $\mathcal{S}$ never sends the broadcast message, and no update occurs. The real and ideal worlds are perfectly aligned.

Indistinguishability holds because $\mathcal{S}$ only provides the honest party's signature after the functionality confirms agreement, and the final confirmation event is triggered by the adversary's own actions in both worlds. Forgery is prevented by the EUF-CMA security of Σ.

Case 3: U_i is honest, U_{i+1} is corrupt. $\mathcal{S}$ simulates the honest party U_i for the corrupted U_{i+1} (controlled by $\mathcal{A}$).

- **Setup Phase:** When $\mathcal{A}$ (as U_{i+1}) sends a $(\texttt{sid}, \texttt{setup}, lb', rb')$ message, $\mathcal{S}$ forwards this to $\mathcal{F}^i_{\text{2pSettle}}$.
- **Init and Approval:** When $\mathcal{E}$ activates the (simulated) honest U_i with $(\texttt{sid}, \texttt{init}, \gamma_i.id, lb', rb')$, $\mathcal{S}$ sends this to $\mathcal{F}^i_{\text{2pSettle}}$. It also constructs and sends the real-protocol $(\texttt{sid}, \texttt{request-settle}, \ldots)$ message to $\mathcal{A}$. If $\mathcal{A}$ responds with a valid signature in an 'approved-settle' message, $\mathcal{S}$ (as honest U_i) adds its signature and broadcasts the transaction to $\mathcal{F}_{\textit{Channels}}$ at round $\tau+2$. This matches the timing of the functionality, which auto-confirms for an honest initiator at round $\tau+2$. If $\mathcal{A}$ withholds its signature, $\mathcal{S}$ cannot broadcast. The real protocol times out. In the ideal world, the functionality still confirms internally, but since no 'UPDATED' message arrives

from $\mathcal{F}_{Channels}$, no 'settle-OK' is delivered to the parties, which is indistinguishable to the environment.

Indistinguishability holds. The honest U_i's behavior is deterministic. Any attempt by $\mathcal{A}$ to forge a signature on a transaction with different balances would violate EUF-CMA security.

Case 4: Both U_i and U_{i+1} are corrupt. $\mathcal{S}$ acts as a passthrough for messages between $\mathcal{A}$ and $\mathcal{E}$. If $\mathcal{A}$ successfully executes the protocol and broadcasts a valid $tx^i_{\texttt{settle}}$, $\mathcal{S}$ extracts the parameters, injects the corresponding 'setup' and 'init' messages into $\mathcal{F}^i_{\text{2pSettle}}$ to bring it to the 'agreed' state, and then sends the '(sid, broadcast)' message to trigger the confirmation. Since no honest party guarantees are violated, the view of the environment is trivially identical to the real world.

In all cases, the environment's view is computationally indistinguishable between the real and ideal worlds. The simulation correctly handles all message flows and relies on the underlying cryptographic assumptions to ensure security. Therefore, the protocol Π^i_{2pSettle} GUC-realizes $\mathcal{F}^i_{\text{2pSettle}}$. □

F.2.5 UC proof for the main functionality

Theorem 3. *Let Σ be an EUF-CMA secure signature scheme. The protocol Π_{Zeus} GUC-realizes the ideal functionality $\mathcal{F}_{Zeus}$ in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world.*

Proof. The proof proceeds via a sequence of hybrid arguments. We show that an environment $\mathcal{E}$ cannot distinguish between an interaction with the real protocol Π_{Zeus} and an interaction with the ideal functionality $\mathcal{F}_{\text{Zeus}}$. For clarity, we unify the naming of the protocol and functionality to Π_{Zeus} and $\mathcal{F}_{\text{Zeus}}$ respectively.

- **World H_0:** This is the real world execution where the protocol Π_{Zeus} interacts with a PPT adversary $\mathcal{A}$ in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world.

- **World H_1:** This world is identical to H_0, except that all calls to the launcher sub-protocols ($\Pi_{\text{rollback-launcher}}$, $\Pi_{\text{redeem-launcher}}$, $\Pi_{\text{slash-launcher}}$) are replaced with calls to their corresponding ideal functionalities ($\mathcal{F}_{\text{rollback-launcher}}$, $\mathcal{F}_{\text{redeem-launcher}}$, $\mathcal{F}_{\text{slash-launcher}}$).

  By Lemmas 3, 4, and 5, each of these protocols GUC-realizes its ideal functionality in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world. By the UC Composition Theorem, we can replace all instances of these protocols with their functionalities simultaneously. Therefore, for any adversary $\mathcal{A}$, its view in H_1 is computationally indistinguishable from its view in H_0.

- **World H_2:** This world is identical to H_1, except that all calls to the two-stage revocable channel update protocols ($\Pi^i_{\text{2pSetupDep}}$, $\Pi^i_{\text{2pSetupPrcpl}}$, $\Pi^i_{\text{2pSetupPrm}}$, Π^i_{2pRfdDep}, which are instances of Π^i_{2SR} and $\Pi^i_{\text{R-2SR}}$) are replaced with calls to their corresponding ideal functionalities ($\mathcal{F}^i_{\text{2SR}}$ and $\mathcal{F}^i_{\text{R-2SR}}$).

  By Lemma 9 and Lemma 10, the protocols Π^i_{2SR} and $\Pi^i_{\text{R-2SR}}$ GUC-realize their respective ideal functionalities in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world. Since world H_1 is already in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world (as the launcher functionalities are simple wrappers), we can apply the UC Composition Theorem to replace all channel update protocols with their ideal functionalities. Therefore, world H_2 is computationally indistinguishable from world H_1.

- **World H_3:** This world is identical to H_2, except that all calls to the fast settlement protocol (Π^i_{2pSettle}) are replaced with calls to its ideal functionality ($\mathcal{F}^i_{\text{2pSettle}}$).

  By Lemma 11, the protocol Π^i_{2pSettle} GUC-realizes $\mathcal{F}^i_{\text{2pSettle}}$ in the $\mathcal{F}_{\texttt{prelim}}$-hybrid world. By the same logic as the previous step, we apply the UC Composition Theorem. Therefore, world H_3 is computationally indistinguishable from world H_2.

Simulation of H_3 in the Ideal World: We now construct a simulator $\mathcal{S}$ that, given access to the ideal functionality $\mathcal{F}_{\text{Zeus}}$, can simulate the view of the environment $\mathcal{E}$ in an interaction with the protocol in world H_3. In H_3, the only remaining real protocol logic is the orchestration of calls to the ideal sub-functionalities. The simulator $\mathcal{S}$ must translate the adversary's interactions with this orchestration logic into interactions with the monolithic $\mathcal{F}_{\text{Zeus}}$. The view of the environment $\mathcal{E}$ is computationally indistinguishable. In both the H_3 world and the ideal world simulated by $\mathcal{S}$, the sequence of events and conditions for payment success or failure are governed by the same logic, which is encapsulated in $\mathcal{F}_{\text{Zeus}}$. Any deviation by $\mathcal{A}$ in H_3 (e.g., providing an invalid signature) is handled by the ideal sub-functionalities, and $\mathcal{S}$ can replicate this outcome using the corresponding logic in $\mathcal{F}_{\text{Zeus}}$.

Simulator for Π_{Zeus} in H_3

The simulator $\mathcal{S}$ runs an internal copy of the adversary $\mathcal{A}$. It simulates the environment $\mathcal{E}$ for $\mathcal{A}$ and the ideal sub-functionalities that constitute the H_3 world. $\mathcal{S}$ forwards messages between $\mathcal{A}$ and $\mathcal{E}$ and translates the protocol flow into commands for $\mathcal{F}_{\text{Zeus}}$.

1. **Setup Phase:** When $\mathcal{E}$ sends $(\texttt{sid}, \texttt{init}, \ldots)$ to the honest U_0 (simulated by $\mathcal{S}$), $\mathcal{S}$ forwards this activation to $\mathcal{F}_{\text{Zeus}}$.
   - If U_0 is honest, $\mathcal{F}_{\text{Zeus}}$ internally generates the setup transactions. $\mathcal{S}$ uses these to construct the protocol messages $(\texttt{request-acc}, \ldots)$ and $(\texttt{request-setup}, \ldots)$ to send to corrupt relays and the payee on behalf of the simulated U_0.
   - If U_0 is corrupt, $\mathcal{A}$ sends setup transactions to honest parties. $\mathcal{S}$ intercepts these messages and provides them to $\mathcal{F}_{\text{Zeus}}$ as specified by the functionality's rules for a corrupt payer. $\mathcal{F}_{\text{Zeus}}$ allows $\mathcal{S}$ to specify different transactions for each party, mimicking a malicious payer.

   $\mathcal{S}$ then simulates the honest party checks and responses (e.g., U_{n+1} signing tx_{redeem}) based on the instructions from $\mathcal{F}_{\text{Zeus}}$. The activation of the launcher functionalities in H_3 corresponds to the setting of internal tags (e.g., $\texttt{tag}_{redeem} = 1$) inside $\mathcal{F}_{\text{Zeus}}$.
2. **Payment Phases (Lock-Deposit, Lock-Premium, etc.):** $\mathcal{S}$ simply executes the protocols in the H3 world. Note that in each of these

phases, the protocol is only interacting with the sub-ideal functionalities. The $\mathcal{F}_{\text{Zeus}}$ functionality is the wrapped functionality of all the sub-functionalities.

Since world H_0 is indistinguishable from H_1, H_1 from H_2, H_2 from H_3, and H_3 from the ideal world with $\mathcal{F}_{\text{Zeus}}$ and $\mathcal{S}$, by transitivity, the real protocol execution in H_0 is computationally indistinguishable from the ideal execution. This concludes the proof. □

F.3 Proof of security and efficiency goals

In this section, we formally prove the main ideal functionality $\mathcal{F}_{Zeus}$ in the $\{\mathcal{F}_{\texttt{prelim}}, \mathcal{F}_{\texttt{sub}}\}$-hybrid model achieves the design goals. Here, we provide the formal definition of the design goals, and then prove the security of *Zeus*.

Setup. Given a multi-hop payment denoted by $\mathcal{M}$. $\mathcal{M}$ contains the metadata for the multi-hop payment, and s is the credential required by the payer, h is the corresponding commitment.

For the metadata $\mathcal{M}$, $\mathcal{M} := (U, \Gamma, x, \text{Fees}, \mathcal{S})$ where $U = (U_0, U_1, ..., U_n, U_{n+1})$ is an ordered sequence of $n+2$ parties with payer U_0, payee U_{n+1}, and intermediate relays U_1 through U_n. $\Gamma = (\gamma_0, ..., \gamma_n)$ denotes the sequence of payment channels, where U_i establishes channel γ_i with U_{i+1}, for $i = 0$ to n. The payment amount is x, and $\text{Fees} = (f_1, ..., f_n)$ represents the sequence of relay fees where f_i is the fee charged by intermediary U_i. We denote the on-chain transaction cost as F. $\mathcal{S}$ contains all the states of channel, as defined in Appendix F.1.1. In *Zeus*, T denotes the timelock for the principal, while Δ is the blocks required for the on-chain transaction to be confirmed.

Assumptions. Our model follows a synchronous and authenticated communication model, with a PPT adversary $\mathcal{A}$ controlling all but explicitly-specified honest parties.

F.3.1 Balance security

Here, we give the formal definition of balance security in *Zeus*.

Definition 9. *(Balance Security) Given a multi-hop payment $\mathcal{M}$, we say an ideal functionality $\mathcal{F}_{MHP}$ achieves balance security: 1) if for any PPT adversary $\mathcal{A}$, the balance of any honest party $U_{i\in[1,n+1]}$ will not decrease compared with their initial balance; and 2) if payer U_0 and payee U_{n+1} are honest, U_0 pays U_1 in the first channel γ_0 only if U_{n+1} gets paid in γ_n.*

Theorem 4 (*Zeus* achieves balance security)**.** *Given a multi-hop payment $\mathcal{M}$, the main ideal functionality $\mathcal{F}_{Zeus}$ is executed in the $\mathcal{F}_{prelim}$-hybrid world achieves balance security.*

Proof. We prove balance security by analyzing all possible channel states that an honest party might encounter during protocol execution. For each honest party, we demonstrate that their total balance across all channels will not decrease below their initial balance, regardless of adversarial behavior.

For simplicity, we assume all channels have the same initial balance (lb, rb), where lb is the left balance and rb is the right balance for any channel $\gamma_{i\in[0,n]}$. We use the notation $\gamma_i \to S^i_{\texttt{STATE}}$ to denote that channel γ_i is in state $S^i_{\texttt{STATE}}$. When the on-chain rollback-redemption happens, the relays can optimistically revokes its locked funds off-chain. So we omit the cost of revocation in our analysis.

We first shows that if payer and payee are honest, payer pays relay U_1 in channel γ_0 iff payee receives a payment with value x: (i) Case 1: No on-chain dispute. In $\mathcal{F}_{Zeus}$, given U_0 and U_{n+1} are honest, U_0 settles γ_0 to `paid` off-chain only after U_{i+1} confirms $\gamma_n \to$ `Refunded-Dep` state and γ_0 to `paid` state. Once the `Refunded-Dep` state times out, it is equivalent to the `paid` state (in both states, the payee has a new balance of $rb + x$). If γ_n never enters the `Refunded-Dep` state, the honest payer U_0 will never settle γ_0 to `paid`, and the rollback-redemption process will revoke all off-chain payments. (ii) Case 2: $tx_{rollback}$ is finalized on-chain in time. In this case, by the Rollback service of $\mathcal{F}_{Zeus}$, U_0 makes on off-chain payment.

We then analyze balance security for the other parties: honest relays $U_{i\in[21,n]}$, and honest payee U_{n+1} under different cases.

Honest intermediate relays $U_{i\in[1,n]}$. We analyze all possible state combinations of γ_{i-1} and γ_i, considering both on-chain and off-chain scenarios:

- **Idle states**: $\gamma_{i-1} \to S^{i-1}_{\texttt{IDLE}}$ and $\gamma_i \to S^i_{\texttt{IDLE}}$.
  In this baseline case, U_i maintains its initial balance of $lb + rb$.
- **Left channel with deposit locked, right channel idle**: $\gamma_{i-1} \to S^{i-1}_{\texttt{LOCKED-Dep}}$ and $\gamma_i \to S^i_{\texttt{IDLE}}$.
  This analysis has two subcases:
  - If $tx_{rollback}$ is not finalized on chain by block $T - t_{close} - \Delta$ (optimistic case): U_i's balance in γ_{i-1} increases by $\texttt{Dep}_{i-1}$ when the timelock expires, while balance in γ_i remains lb. Total balance increases to $lb + rb + \texttt{Dep}_{i-1}$.
  - If $tx_{rollback}$ is finalized in time (dispute case): The rollback trigger activates $\mathcal{F}^{i-1}_{\text{rollback-handler}}$, allowing U_i to revoke any locked funds in γ_{i-1} through RvkTx_L. Balance remains $lb + rb$.
- **Both channels with deposit locked**: $\gamma_{i-1} \to S^{i-1}_{\texttt{LOCKED-Dep}}$ and $\gamma_i \to S^i_{\texttt{LOCKED-Dep}}$.
  This analysis has two subcases:
  - Without rollback: U_i's balance in γ_{i-1} increases by $\texttt{Dep}_{i-1}$ while decreasing in γ_i by $\texttt{Dep}_i$. By protocol design (as specified in $\mathcal{F}^i_{\text{2pSetupDep}}$), $\texttt{Dep}_{i-1} - \texttt{Dep}_i = c_{\text{relay}} > 0$, ensuring total balance doesn't decrease.

– With rollback: The ideal functionality $\mathcal{F}^{i}_{\text{rollback-handler}}$ allows U_i to recover its locked deposit in γ_i, so balance is at least $lb+rb$. If its left relay U_{i-1} fails to revoke its locked funds in γ_{i-1}, U_i get a bonus of $\texttt{Dep}_{i-1}$ in γ_{i-1}.

- **Left channel with deposit locked, right channel with premium locked**: $\gamma_{i-1} \to S^{i-1}_{\texttt{LOCKED-Dep}}$ and $\gamma_i \to S^{i}_{\texttt{LOCKED-Prm}}$.
This analysis has two subcases:
  – Without rollback: U_i gains $\texttt{Dep}_{i-1}$ in γ_{i-1} and $\texttt{Prm}_i$ in γ_i. So total balance increases to $lb+rb+c_{\text{relay}}+\texttt{Prm}_i$.
  – With rollback: U_i can revoke its deposit in γ_i through $\mathcal{F}^{i}_{\text{rollback-handler}}$, while maintaining balance in γ_{i-1}. If redemption doesn't occur, U_i also retains $\texttt{Prm}_i$. So the balance of U_i is at least $lb+rb$.

- **Both channels with premium locks**: $\gamma_{i-1} \to S^{i-1}_{\texttt{LOCKED-Prm}}$ and $\gamma_i \to S^{i}_{\texttt{LOCKED-Prm}}$.
This analysis has three subcases:
  – Without rollback: U_i's balance changes by $(\texttt{Prm}_i - \texttt{Prm}_{i-1}) + (\texttt{Dep}_{i-1} - \texttt{Dep}_i)$. By protocol design in $\mathcal{F}^{i}_{\text{2pSetupPrm}}$, we have $\texttt{Prm}_i - \texttt{Prm}_{i-1} = f_i$ and $\texttt{Dep}_{i-1} - \texttt{Dep}_i = c_{\text{relay}}$, ensuring non-negative change.
  – With rollback but no redemption: U_i can recover deposit in γ_i and retain premium $\texttt{Prm}_i$, increasing balance by f_i.
  – With rollback and redemption: Both deposit and premium are revoked, maintaining original balance $(lb+rb)$.

- **Left channel with principal, right with premium**: $\gamma_{i-1} \to S^{i-1}_{\texttt{LOCKED-Prcpl}}$ and $\gamma_i \to S^{i}_{\texttt{LOCKED-Prm}}$.
This analysis has three subcases:
  – Without rollback: U_i gains additional principal $\texttt{Prcpl}$ compared to the previous "Both channels with premium locks, no rollback"case, increasing balance further.
  – With rollback but no redemption: U_i can revoke principal in γ_i, with a balance of at least $(lb+rb+f_i)$.
  – With rollback and redemption: All locked funds are revoked, maintaining initial balance.

- **Both channels with principal locks**: $\gamma_{i-1} \to S^{i-1}_{\texttt{LOCKED-Prcpl}}$ and $\gamma_i \to S^{i}_{\texttt{LOCKED-Prcpl}}$.
This analysis has three subcases:
  – Without rollback: $\texttt{Prcpl}$ is identical in both channels by design, so total balance remains $(lb+rb+c_{\text{relay}}+f_i)$.
  – With rollback: U_i can revoke principal in both channels through RvkTx_L transactions facilitated by $\mathcal{F}^{i-1}_{\text{rollback-handler}}$ and $\mathcal{F}^{i}_{\text{rollback-launcher}}$, maintaining balance of $(lb+rb+f_i)$
  – With rollback and redemption: All locked funds are revoked, maintaining initial balance.

- **Left channel with principal, right with deposit refund**: $\gamma_{i-1} \to S^{i-1}_{\texttt{LOCKED-Prcpl}}$ and $\gamma_i \to S^{i}_{\texttt{REFUNDED-Dep}}$.
This analysis has three subcases:
  – Without rollback: Compare with "Both channels with principal locks, no rollback" case, U_i has a better payoff, as its pay less in channel γ_i.
  – With rollback and no redemption: U_i can revoke principal in γ_{i-1} through RvkTx_L, while retaining the deposit refund in γ_i.
  – With rollback and redemption: All locked funds are revoked, maintaining initial balance.

- **Both channels with deposit refunds**: $\gamma_{i-1} \to S^{i-1}_{\texttt{REFUNDED-Dep}}$ and $\gamma_i \to S^{i}_{\texttt{REFUNDED-Dep}}$.
In this state, U_i's balance in γ_{i-1} is $lb-\texttt{Prcpl}+\texttt{Prm}_{i-1}$, and in γ_i it's $rb-\texttt{Prm}_i+\texttt{Dep}_i$. By protocol design, $\texttt{Prm}_{i-1} \geq \texttt{Prcpl}$ and $\texttt{Dep}_i \geq \texttt{Prm}_i$, ensuring total balance doesn't decrease. This holds regardless of whether rollback or redemption occurs.

- **Left channel paid, right with deposit refund**: $\gamma_{i-1} \to S^{i-1}_{\texttt{PAID}}$ and $\gamma_i \to S^{i}_{\texttt{REFUNDED-Dep}}$
When $\gamma_{i-1} \to S^{i-1}_{\texttt{PAID}}$, U_i's balance becomes $rb+\texttt{Prcpl}-\texttt{Prm}_{i-1} = rb+x+\sum_{j=i}^{n} f_j$. In γ_i, the balance is $lb-\texttt{Prcpl}+\texttt{Prm}_i = lb-x-\sum_{j=i+1}^{n} f_j$. Therefore, total balance is $lb+rb+f_i$, which exceeds the initial amount. This remains true regardless of rollback activity since γ_{i-1} has already settled.

- **Both channels paid**: $\gamma_{i-1} \to S^{i-1}_{\texttt{PAID}}$ and $\gamma_i \to S^{i}_{\texttt{PAID}}$
In this optimal case, U_i securely earns relay fee f_i, with total balance increasing to $lb+rb+f_i$.

Therefore, *Zeus* achieves balance security for all honest relays $U_{i\in[1,n]}$ across all possible channel state combinations.

Honest payee U_{n+1}. We analyze the payee's balance security:

- **Without rollback**: Given $\texttt{Dep}_n > \texttt{Prm}_n$, $\texttt{Prm}_n + x = \texttt{Prcpl}$, for all possible states of channel γ_n without rollback, the balance of U_{n+1} never decreases below rb. In optimal case ($\gamma_n \to S^{n}_{\texttt{PAID}}$), the balance increases to $rb+x$.

- **With rollback**: If rollbak transaction submitted, the payee U_{n+1} can redeem the payment on-chain and obtain

the payment amount x, while successfully revoking all off-chain locked funds. The timing guarantees provided by $\mathcal{F}_{Zeus}$ ensure U_{n+1} has sufficient time to complete this redemption within $t_{close} + \Delta$ after rollback.

Through this comprehensive analysis of all possible channel states and transitions, we have demonstrated that *Zeus* achieves balance security for all honest parties under any adversarial behavior. □

F.3.2 Correctness

Definition 10. *Given a multi-hop payment $\mathcal{M}$, we say a MHP ideal functionality $\mathcal{F}_{MHP}$ achieves correctness if the final state of all channels will be in the* `paid` *state when all parties are honest.*

Theorem 5 (*Zeus* achieves correctness)**.** *Given a multi-hop payment $\mathcal{M}$, if the ideal functionality $\mathcal{F}_{Zeus}$ is executed in the $\mathcal{F}_{prelim}$-hybrid model, then $\mathcal{F}_{Zeus}$ achieves correctness.*

The proof of this theorem is straightforward, when every party honest, all channel will be in the `paid` state.

F.3.3 Coin availability

Definition 11. *Given a multi-hop payment $\mathcal{M}$, we say a ideal functionality $\mathcal{F}_{MHP}$ achieves coin availability if for any corrupted PPT $\mathcal{A}$, the locked funds issued from honest U_i will not be locked forever.*

Theorem 6 (*Zeus* achieves coin availability)**.** *Given a multi-hop payment $\mathcal{M}$, if the ideal functionality $\mathcal{F}_{Zeus}$ is executed in the $\{\mathcal{F}_{prelim}, \mathcal{F}_{sub}\}$-hybrid model, then $\mathcal{F}_{Zeus}$ achieves coin availability.*

This theorem is straightforward, by examining all possible states listed in Appendix F.1.1 as the locked funds in any channel will be released after a limited timelock. The timelock is irrelevant with the length of payment path, so no funds will be locked forever.

F.3.4 Fee security

Here, we provide the formal definition of fee security for a MHP ideal functionality. We demonstrate that our main ideal functionality $\mathcal{F}_{Zeus}$ achieves fee security.

Definition 12 (Fee Security)**.** *Given a multi-hop payment $\mathcal{M}$, we say an ideal functionality $\mathcal{F}_{MHP}$ satisfies fee security if for any honest relay U_i ($i \in [1,n]$), in the presence of a PPT adversary $\mathcal{A}$ controlling all other parties, the following guarantee holds: if the payee U_{n+1} receives the payment with the amount x off-chain, then U_i must receive its corresponding relay fee f_i if it has provided liquidity (locked funds) during the protocol execution.*

Theorem 7 (*Zeus* achieves fee security)**.** *Given a multi-hop payment $\mathcal{M}$, if the ideal functionality $\mathcal{F}_{Zeus}$ is executed in the $\mathcal{F}_{prelim}$-hybrid model, then $\mathcal{F}_{Zeus}$ achieves fee security.*

Proof. We analyze three distinct cases to demonstrate that *Zeus* achieves fee security as defined in Definition 12. For each case, we show that when the payee receives the payment x off-chain, any honest relay U_i that has provided liquidity must receive its corresponding fee f_i.

Case 1: No on-chain process occurs. For an honest relay U_i, we show that if the payee receives the payment off-chain, relay U_i will receive its corresponding relay fee f_i. Recall the proof of balance security (Appendix F.3.1), where we analyzed the balance of any honest relay U_i under all possible conditions. We observe that once U_i locks the payer-side deposit off-chain, it can achieve one of the following payoff states:

- $+c_{\text{relay}}$: This occurs when the protocol aborts during the premium-lock phase. Here, U_i has only locked the payer's deposit, with amount $\texttt{Dep}_i = c_{\text{payee}} + (n-i)c_{\text{relay}}$. When both channel γ_{i-1} and γ_i are timeouts with locked deposits, honest U_i can claim a reward c_{relay}. Noting the deposit value $\texttt{Dep}_i$ is defined as $c_{\text{payee}} + (n-i)c_{\text{relay}}$, which is a small value a few USD (see Appendix A.3). This reward is sufficient to compensate for the relay cost associated with the locked $\texttt{Dep}_i$.
- $+c_{\text{relay}} + f_i$: This occurs when the protocol aborts during the deposit-refund phase. In this scenario, U_i receives a payoff of $c_{\text{relay}} + f_i$, which adequately covers the liquidity cost of both the locked principal and the payer's deposit.
- $+f_i$: This represents either successful payment completion or protocol abortion during the settlement phase. In either case, relay U_i receives its designated relay fee f_i, satisfying the fee security requirement.

Case 2: Rollback transaction finalizes on-chain before timeout $T - t_{close} - \Delta$ but on redemption transaction. Based on our analysis in Appendix F.3.1, an honest relay U_i can achieve one of the following payoffs:

- 0: This occurs when the off-chain protocol aborts before U_{i+1} locks the premium, and the rollback transaction is finalized. Although U_i receives no additional compensation in this scenario, the payer has chosen to settle the payment directly on-chain rather than through the relay network. Therefore, the fee security property is not violated since U_i did not contribute to the successful payment routing.
- $+f_i$: This occurs when the premium is successfully locked in γ_i. In this case, relay U_i receives its fee f_i as required by the fee security definition.

In both subcases, the fee security guarantee is maintained: either U_i did not contribute to payment routing (no premium locked) or it receives its fee f_i.

Case 3: Splicing transaction finalizes on-chain before $T - t_{close} - \Delta$, with payee redemption on-chain before $T - t_{close}$. In this final scenario, if all neighbors of the honest relay U_i behave rationally, U_i achieve a zero additional payoff (receiving only its initial balance $lb + rb$). However, this does not violate fee security because the payment is settled directly on-chain rather than through the off-chain relay path. Since Definition 12 specifically addresses fee security when "the payee U_{n+1} receives the payment with the amount x off-chain," this on-chain settlement case falls outside the scope of the guarantee.

Across all three cases, we have demonstrated that whenever the payee receives the payment off-chain, any honest relay that has provided liquidity during protocol execution receives its corresponding fee, thereby satisfying the fee security requirement as defined in Definition 12. □

F.3.5 Griefing resistance against cost-sensitive adversaries

In this part, we prove that our ideal functionality $\mathcal{F}_{Zeus}$ achieves griefing resistance against cost-sensitive adversaries defined in Section 2, which requires that the cost incurred by an adversary to conduct a griefing attack exceeds the financial damage inflicted on honest parties.

Theorem 8 (*Zeus* achieves griefing resistance against cost-sensitive adversaries)**.** *Given a multi-hop payment $\mathcal{M}$, if the ideal functionality $\mathcal{F}_{Zeus}$ is executed in the $\mathcal{F}_{prelim}$-hybrid model, then $\mathcal{F}_{Zeus}$ achieves griefing resistance against 1-cost-sensitive adversaries.*

Proof. We demonstrate that $\mathcal{F}_{Zeus}$ satisfies Theorem 8 by showing that for any possible PPT adversary $\mathcal{A}$ controlling S_A, the griefing damage (liquidity cost imposed on honest parties due to locked funds) is strictly less than the griefing cost (minimum transaction costs incurred by the adversary to execute the attack).

Let us denote by F the cost of executing a standard (1 input, 1 output) Bitcoin transaction, f the relay fee charged by each party, r the time-value interest rate for locked Bitcoin, T the principal timelock duration, and Δ the confirmation time for on-chain transactions. We analyze all possible adversarial collusion patterns:

Case 1: Corrupted payer U_0 We consider two attack strategies:

- *Attack 1: Abort after premium-lock phase.* If U_0 aborts the protocol after the premium-lock phase. If the rollback revocation does not occur in time, each honest relay $U_{i\in[1,n]}$ receive a compensation of $c_{\text{relay}} + f_i$. If rollback occurs, the adversary U_0 incurs a transaction cost of at least F. The total liquidity cost (griefing damage) for honest parties (GD_1) is:

$$r \cdot \left[\left(nc_{\text{payee}} + \frac{n(n+1)}{2} c_{\text{relay}} \right) T + \left(nf_0 + \frac{n(n+1)}{2} f \right) (T + \Delta) \right]$$

  Using the configuration parameters specified in Appendix A.3, even in the worst-case scenario with $T = 2$ hours and $\Delta = 1$ hour, $\text{GD}_1 < 0.01$ USD, which is significantly less than the adversary's griefing cost of $F \approx 0.3$ USD.

- *Attack 2: Revoke payment after reaching refunded-deposit state.* If the payer attempts to revoke an off-chain payment after all channels have reached the refunded-deposit state, the liquidity cost (GD_2) is:

$$r \cdot \left[(nx + nf_0 + n^2 f)(T) + \left(nf_0 + \frac{n(n+1)}{2} f \right) (T + \Delta) \right]$$

  For a rational payment system, the total relay fees must satisfy $nf \leq F$, as otherwise direct on-chain payments would be more economical. With a standard fee rate of $fr = 0.01\%$, we can rewrite the payment amount as $x = \frac{f}{fr}$, yielding:

$$\begin{aligned} \text{GD}_2 &= r \cdot \left[\left(\frac{nf}{fr} + nf_0 + n^2 f \right) (T) + \left(nf_0 + \frac{n(n+1)}{2} f \right) (T + \Delta) \right] \\ &\leq r \cdot \left[\left(\frac{F}{fr} + 2nF \right) T + (nF + (n+1)F/2)(T + \Delta) \right] \\ &\approx 0.23F < F \end{aligned}$$

  Thus, the griefing cost exceeds the griefing damage.

Case 2: Corrupted payee U_{n+1} U_{n+1} has two griefing strategies: 1) griefing the payer's deposit while burning its on-chain stake by aborting *Zeus* after the payer's deposit is fully locked; 2) griefing the entire principal by aborting after the lock-principal phase when locked funds reach their maximum, forcing the honest payer to roll back, and then redeeming the funds. For the first strategy, Appendix C.2 in the manuscript has shown that given a small payer's deposit, we can reduce the griefing factor to less than one. We now focus on the second strategy, where the adversary incurs a griefing cost of at least F, and the griefing damage (GD_3) is:

$$r \cdot \left[(n(x + c_{\text{payee}}) + n^2(f + c_{\text{relay}}))(T) + \left(nf_0 + \frac{n(n+1)}{2} f \right) (T + \Delta) \right]$$

Under the same assumption that $nf \leq 2F$, this griefing damage is approximately 0.25 USD, which is less than the griefing cost of F.

Case 3: Corrupted payer U_0 and payee U_{n+1} With both endpoints corrupted, the griefing damage remains the same as in Case 2:

$$\mathrm{GD}_4 = \mathrm{GD}_3$$

However, the adversary now incurs a significantly higher griefing cost of at least $2F$, as both corrupted endpoints must place on-chain transactions to execute the attack.

Case 4: Corrupted payer U_0 with some relays This case is similar to Case 1, where the griefing cost remains F, but the griefing damage decreases since liquidity provided by corrupted relays is excluded from the damage calculation. Thus, the griefing cost exceeds the griefing damage.

Case 5: Corrupted payee U_{n+1} with some relays Similar to Case 2, the griefing cost is F, but the griefing damage decreases due to the exclusion of corrupted relays' liquidity from the damage calculation. Therefore, the griefing cost exceeds the griefing damage.

Case 6: Corrupted payer U_0, payee U_{n+1}, and some relays This case combines the characteristics of Case 3 and Cases 4-5. The griefing cost remains at least $2F$ due to the necessary on-chain transactions, while the griefing damage is reduced compared to Case 3. Therefore, the griefing cost strictly exceeds the griefing damage.

For all possible adversarial coalitions defined in Section 2, we have demonstrated that the griefing cost consistently exceeds the griefing damage. Therefore, the ideal functionality $\mathcal{F}_{Zeus}$ achieves the griefing resistance property. □

F.3.6 Constant collateral

Definition 13. *Given a multi-hop payment $\mathcal{M}$, we say an ideal functionality $\mathcal{F}_{MHP}$ achieves constant collateral if the locked funds in any channel $\gamma_{i\in[0,n]}$ have constant timelocks, independent of the payment path length n.*

Theorem 9 (*Zeus* achieves constant collateral)**.** *Given a multi-hop payment $\mathcal{M}$, if the ideal functionality $\mathcal{F}_{Zeus}$ is executed in the $\mathcal{F}_{\mathtt{prelim}}$-hybrid model, then $\mathcal{F}_{Zeus}$ achieves constant collateral.*

Similar to the coin availability, the theorem is straightforward, ranging all funds locked in any possible state, the timelock is bounded and irrelevant with the length of payment path, so no funds will be locked forever.