On the Atomicity and Efficiency of Blockchain Payment Channels

Di Wu
Zhejiang University

Shoupeng Ren
Zhejiang University

Yuman Bai
Zhejiang University

Lipeng He
University of Waterloo

Jian Liu*
Zhejiang University

Wu Wen
Zhejiang University

Kui Ren
Zhejiang University

Chun Chen
Zhejiang University

Abstract

Payment channels have emerged as a promising solution to address the performance limitations of cryptocurrencies payments, enabling efficient off-chain transactions while maintaining security guarantees. However, existing payment channel protocols, including the widely-deployed Lightning Network and the state-of-the-art Sleepy Channels, suffer from a fundamental vulnerability: non-atomic state transitions create race conditions that can lead to unexpected financial losses. We first formalize current protocols into a common paradigm and prove that this vulnerability is fundamental—any protocol following this paradigm cannot guarantee balance security due to the inherent race conditions in their design. To address this limitation, we propose a novel atomic paradigm for payment channels that ensures atomic state transitions, effectively eliminating race conditions while maintaining all desired functionalities. Based on this paradigm, we introduce ULTRAVIOLET, a secure and efficient payment channel protocol that achieves both atomicity and high performance, while avoiding the introduction of unimplemented Bitcoin features. ULTRAVIOLET reduces the number of required messages per transaction by half compared to existing solutions, while maintaining comparable throughput. We formally prove the security of ULTRAVIOLET under the universal composability framework and demonstrate its practical efficiency through extensive evaluations across multiple regions. This results in a 37% and 52% reduction in latency compared to the Lightning Network and Sleepy Channels, respectively. Regarding throughput, ULTRAVIOLET achieves performance comparable to the Lightning Network and delivers 2× the TPS of Sleepy Channels.

1 Introduction

Decentralized cryptocurrencies have introduced a revolutionary payment system, gaining widespread popularity in recent years. By the end of 2024, the market cap of cryptocurrencies has surpassed $3 trillion [3]. However, limited by the inherent trade-off between decentralization and scalability, the underlying blockchains powering these digital assets remain inefficient and struggle to meet the growing demands of users. For instance, while traditional payment systems like Visa process transactions with near-instant confirmation and support peak loads of up to 56,000 transactions per second (TPS), Bitcoin [30]—the most prominent blockchain-based cryptocurrency—requires 10 minutes or more to confirm a transaction and supports a maximum throughput of only 7 TPS [14].

Payment Channels (PCs) are among the most promising scalability solutions, significantly reducing payment delays and increasing transaction throughput. PCs enable transactions between two users to take place off-chain in a secure manner while requiring minimal interaction with the blockchain. Ideally, a payment channel involves only two on-chain transactions: one for *opening*, which locks the funds of the participants, and one for *closing*, which finalizes the channel and distributes the funds back to the participants. Early PC protocols [4] only supported unidirectional payments from one party to the other. Later on, bidirectional PCs [15, 16] are introduced, enabling both parties to send payments to each other off-chain for an unlimited number of times. These payments dynamically *update* the state (i.e. balance distribution) within the channel without placing burden on the blockchain. The most well-known bidirectional payment channel protocol is the Lightning Network (LN) [1], deployed on Bitcoin, which hosts bitcoins more than $520 million [2] as of the time of writing. In the off-chain payment process of the LN protocol, both participants must exchange a signed transaction representing the new state and revoke the outdated state they hold. In addition, the state-of-the-art (SOTA) payment channel protocol, Sleepy Channel (SC) [9], follows the Lightning-style design and introduces further improvements, removing reliance on watchtowers [10, 27].

Non-Atomicity Issue. Despite its widespread use, LN has been found to suffer from a design flaw identified in a recent

*Corresponding author

study [33] (CCS'24), leading to the so-called *Payout Race Attack*. The issue arises during off-chain update process, where one party can hold two valid states simultaneously, placing the channel in an ambiguous state. This ambiguity creates the risk that the channel could be finalized in a state unexpected by the counterparty, potentially resulting in financial losses. Both the authors of the study and the LN's team have acknowledged that this issue cannot be mitigated.

We further point out that this design flaw exposes the payer to inevitable financial losses when an anticipated message is not received during the off-chain state update process. Consider a real-world scenario in which LN is used via a wallet application for electronic payments. After the payer confirms the payment in the application, a signed transaction representing the new state is sent to the payee, and then the payer awaits a message in response. However, due to issues such as network partitioning, delays, or malicious behavior by the payee, the payer may fail to receive the expected response. In such cases, the wallet application remains in a prolonged "waiting" status until it eventually times out, indicating that the payment has not been completed. At this point, the payer regards the payment as failed but cannot ascertain whether the new state is successfully sent to the payee or whether the timeout is caused by normal network issues or unresponsive payee. The safest course of action for the payer is to forcibly close the channel by submitting the previously held valid state. However, if the payee is malicious and has received the new state, they can also submit it to the blockchain, resulting in a *race condition* where both parties compete to finalize the channel using different states. If the new state submitted by the malicious payee is confirmed, the payment, which should have failed from the perspective of the payer, will still be forcibly completed, causing the payer to unwittingly lose funds. Conversely, even if the previously valid state submitted by the payer is confirmed, whether by prevailing in the race condition or because the payee is honest, the payer is still forced to bear the transaction fee for closing the channel.

These issues fundamentally arise from the lack of atomicity in payment channel state transitions. Atomicity mandates a direct transition between valid states. In contrast, existing PC protocols employ a multi-step process for state updates, where both parties exchange messages to advance the new state. This results in a window where both the old and new states are valid simultaneously, violating atomicity. This violation enables either party to arbitrarily choose which state to finalize on-chain, thereby creating vulnerabilities. Intuitive solutions, such as grace periods, timeout mechanisms, or acknowledgment messages, fail to address the root cause. As long as the protocol maintains the current multi-step update process, the channel will remain in an ambiguous state during updates. Therefore, ensuring true atomicity necessitates a fundamental redesign of the payment channel protocol.

Efficiency Limitations. In addition, the current design of payment channel protocols faces efficiency challenges. Specifically, both LN and SC require a four-message exchange process to complete a single payment: two messages to update the channel state and two additional messages to revoke the previous state. This design introduces heavy communication overhead, substantially increasing transaction latency in networks with existing delays. The impact becomes particularly severe in cross-regional transactions where network latency is higher. Current protocols cannot reduce communication complexity without compromising security or requiring unsupported Bitcoin features, indicating inherent limitations in existing designs. This suggests the need for a fundamentally different approach to enhance both security and efficiency.

1.1 Our Contributions

The goal of this paper is to solve inherent issues in the SOTA PC protocols and to propose a new protocol that achieves higher security and efficiency.

For the first time, we formalize the SOTA PC protocols, such as LN and SC, into a *common paradigm*. Moreover, we formalize the *balance security* for each phase of a payment channel: *creation*, *update*, and *finalization*.

We demonstrate that the common paradigm inherently lacks *atomicity*, leading to issues caused by *race conditions*, which prevent it from satisfying the balance security definition. Furthermore, we prove that any payment channel protocol adhering to the common paradigm is incapable of mitigating this issue.

To address this limitation, we propose an *atomic paradigm* for payment channels. We prove that the atomic paradigm ensures atomicity, thereby eliminating vulnerabilities introduced by race conditions. Additionally, we argue that the proposed paradigm satisfies the balance security definition.

We present ULTRAVIOLET, a novel payment channel protocol that follows the *atomic paradigm*. To the best of our knowledge, this is the first payment channel protocol that achieves atomicity, as shown in Table 1. Our protocol provides:

- **Security.** Following the atomic paradigm, ULTRAVIOLET inherently prevents race conditions in payment channel operations. Furthermore, we formally prove its security using the Universal Composability (UC) framework [12], with the corresponding ideal functionality and simulator provided in Appendix A.
- **Efficiency.** ULTRAVIOLET achieves its efficiency without relying on any computationally expensive cryptographic primitives. Notably, it requires only two messages per update, effectively halving the communication overhead compared to the LN while maintaining the same functionality. Our evaluation across four regions demonstrates ULTRAVIOLET's superior performance. The results demonstrate that ULTRAVIOLET achieves a latency reduction of 37% compared to LN and 52% compared to SC, while maintaining a comparable TPS

Table 1: Comparison of different payment channel protocols. Flexibility indicates whether a protocol supports bidirectional payments, unrestricted lifetime, and unbound transactions. Compatibility refers to whether the protocol relies only on features currently supported in Bitcoin. SC's flexibility is limited due to its restricted channel lifetime constraint. Eltoo's compatibility is constrained by its dependency on the yet-to-be-implemented `SIGHASH_NOINPUT` feature in Bitcoin, while Generalized Channels (GC) requires a non-deterministic digital signature scheme that is currently not available in Bitcoin. Our proposed solution achieves all three properties, offering atomic operations, full flexibility in payment directions and channel lifetime, while maintaining compatibility with Bitcoin's existing feature set.

PC	Atomicity	Flexibility	Compatibility
LN [1]	✗	✓	✓
SC [9]	✗	✗	✓
Eltoo [15]	✗	✓	✗
GC [6]	✗	✓	✗
ULTRAVIOLET	✓	✓	✓

to LN and delivering a 2× higher TPS than SC.

- **Flexibility.** ULTRAVIOLET supports bidirectional payments with an unrestricted lifetime, meaning the payment channel can remain open indefinitely without a fixed expiration, and unbounded transactions, allowing users to perform an unlimited number of transactions without the need to reset or reopen the channel.
- **Compatibility.** ULTRAVIOLET does not introduce any features not yet implemented in Bitcoin, ensuring seamless integration with existing systems.

2 Preliminaries

2.1 Blockchain and Bitcoin

The blockchain $\mathbb{B}$ is an append-only ledger that ensures consistency for all participants through consensus mechanism. It has a bounded confirmation time [31], denoted as Δ, representing the maximum time required for a valid transaction broadcast by an honest party to be included in the ledger.

In the Bitcoin network, assets are represented as *Unspent Transaction Outputs* (UTXOs), which function as digital checks with a specific amount v and spending conditions. To spend a UTXO, a valid *witness* π must be provided that satisfies its predetermined spending conditions ϕ within the UTXO, denoted as $\pi \models \phi$. These conditions are governed by *scripts*, which are programmable predicates that evaluate to either true or false. The most common script type is the single public key verification, based on a secure digital signature scheme Σ, where a UTXO can be spent by providing a valid digital signature σ corresponding to a specified public key pk, denoted as $\sigma \models pk$. Another prevalent script type is the multi-signature scheme, requiring valid signatures from m out of n designated public keys. More advanced conditions include *relative timelock* constraints, denoted as $\mathsf{RelTime}(t)$, where a UTXO becomes spendable only after a specified time interval t has elapsed since its creation on the ledger: $t_{current} - t_{creation} \geq t \models \mathsf{RelTime}(t)$. Formally, a script can encompass one or multiple conditions $\{\phi_1, \phi_2, ..., \phi_n\}$, combined through logical operators such as AND ($\wedge$) and OR ($\vee$).

The mechanism to spend UTXOs is through *transactions*. A transaction tx consumes one or more UTXOs as inputs $\{utxo_1, ..., utxo_k\}$ and generates one or more new UTXOs as outputs $\{utxo'_1, ..., utxo'_m\}$, where the sum of input values must equal or exceed the sum of output values: $\sum_{i=1}^{k} v_i \geq \sum_{j=1}^{m} v'_j$. Each transaction is uniquely identified by its *transaction identifier* (txid), which is computed as the cryptographic hash of its content: $txid = H(tx)$, using a collision-resistant hash scheme. A UTXO is typically referenced by the txid of its creating transaction and its output index. To validate a transaction, the txid must be submitted to the blockchain along with a set of witnesses $\{\pi_1, ..., \pi_k\}$ satisfying the spending conditions of each input UTXO: $\forall i \in [1,k], \pi_i \models \phi_i$. Once submitted, a valid transaction is expected to be confirmed within a bounded time Δ. However, due to potential *race conditions*, a transaction may be invalidated if a conflicting transaction, which spends the same UTXO, is confirmed earlier on the blockchain.

2.2 Payment Channels

Payment channels enable parties to conduct multiple paymemts off-chain while preserving the security guarantees of the underlying blockchain. By locking funds in a shared UTXO and maintaining valid but unbroadcast transactions that reflect the latest balance state, payment channels minimize on-chain interactions to channel creation and final settlement. Payment channels support various operational features: *bidirectional* payments allow both parties to freely transfer funds within the channel, *unrestricted lifetime* enables the channel to operate indefinitely without expiration, and *unbound transactions* permit an unlimited number of payments without requiring channel resets.

The most widely used payment channel protocol, the *Lightning Network*, operates through a three-phase lifecycle: *opening*, *update*, and *closing*. In the *opening phase*, participants create a funding transaction that locks their shared funds in a multi-signature UTXO. This can be implemented using Bitcoin's native `OP_CHECKMULTISIG` opcode or optimized with Schnorr-based protocols like MuSig/MuSig2. During the *update phase*, participants exchange new commitment transactions to reflect the latest balance state S_i, each representing a balance distribution. To prevent outdated states from being broadcast, the Lightning Network employs a *punishment mechanism*: transitioning from state S_{i-1} to S_i involves ex-

changing revocation keys. These keys allow the participant to claim all channel funds if an outdated state is published by the counterpart. A common implementation is based on preimage: for each payment, a revocation secret and hash pair (rh, rs) are generated. The punishment is enforced using the revocation secret and the participant's signature. In the *closing phase*, participants can either cooperatively broadcast a settlement transaction or forcibly close the channel by publishing the latest commitment transaction they hold.

Recent advances in payment channel research have introduced *Sleepy Channel* [9], a SOTA protocol that eliminates the need for watchtowers through an incentive-based approach while following the Lightning-style design. The key innovation lies in its finite-lifetime design: each channel is created with a predetermined expiration time and additional collateral requirements that incentivise honest settlement behaviour. Unlike traditional protocol that rely on continuous monitoring or third-party watchtowers, Sleepy Channel leverages economic incentives and time-bound constraints to ensure security.

2.3 Universal Composability Framework

The Universal Composability (UC) framework provides a formal methodology for analysing protocol security. Within this framework, the security of a protocol is evaluated by comparing its execution to an ideal functionality, which abstractly captures the desired security properties. A protocol is considered UC-secure if, for every potential adversary in the real protocol execution, there exists a simulator in the ideal-world execution such that no environment can distinguish between the two. This guarantees that the protocol is as secure as its idealized counterpart. In the context of payment channels, UC-secure ensures that the protocol retains its defined security properties even when executed concurrently with other protocols or within complex, adversarial environments. This framework has been extensively applied to analyze various payment channel protocols [6, 20], including the SOTA protocol Sleepy Channels [9].

3 A Common Paradigm for PC Protocols

3.1 Overview

As illustrated in Figure 1, LN and SC both rely on the same overarching structure for off-chain payments. In what follows, we recall how each protocol transitions through three main phases—*creation*, *update*, and *finalization*—and emphasize the *core principle* that each state update is realized via *two* local copies with explicit state revocation.

Core Principle: Dual-State Copies with Explicit Revocation. A defining characteristic of both LN and SC is that any valid channel state is captured by two local copies, one for each participant: S_i^A and S_i^B, held by B and A, respectively. These two copies must remain consistent: $S_i^A = S_i^B$.

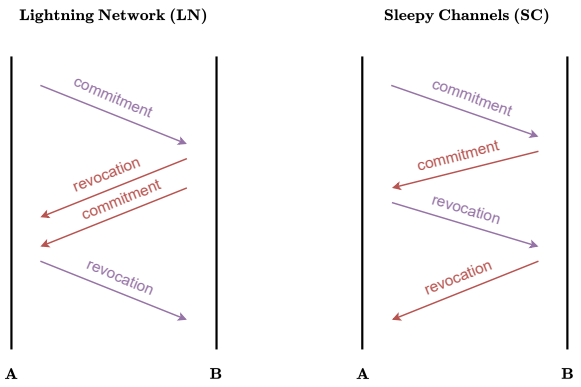

Figure 1: Message flow during the update process in the Lightning Network and Sleepy Channel, both maintaining synchronized local channel states with explicit state revocation, despite differences in sequence and details.

In practice, when updating the channel state, each participant sends the counterparty a commitment—a signed transaction capable of spending the channel funds and reflecting the current state. For clarity, we denote the i-th commitment signed by participant A as S_i^A. This state is intended to be sent to the counterparty, participant B, and can only be broadcast on-chain by B to finalize the channel. If a state is not set to $\perp$, it means the state is valid. By "valid", we mean that the state can be submitted by B to the blockchain to finalize the channel without incurring any financial loss to B. Whenever a new state (S_{i+1}^A, S_{i+1}^B) is introduced, the previous state (S_i^A, S_i^B) must be explicitly revoked by both parties. This explicit revocation step, typically enforced via a punishment mechanism, ensures that outdated states cannot be used by either party to close the channel, thereby completing the *State Replacement*.

Definition 3.1 (State Replacement)**.** *When the state is updated from (S_n^A, S_n^B) at time t_0, after receiving the new state S_{n+1}^A or S_{n+1}^B ($\vdash$), each party must independently revoke ($\perp$) the previous state. The previous state is considered fully replaced only after both parties have explicitly completed the revocation process.*

$$\exists P, Q \in \{A, B\}, \quad \exists t_0 < t_1 < t_2 :$$
$$\vdash S_{n+1}^P, \quad t_1 > t > t_0, \quad S_n^P \to \perp, \quad t \geq t_1$$
$$\vdash S_{n+1}^Q, \quad t_2 > t \geq t_1, \quad S_n^Q \to \perp, \quad t \geq t_2$$

Three-Phase Process: Creation, Update, and Finalization Beyond the core principle, the common paradigm follow a three-phase process over the lifespan of the payment channel:

- **Creation.** Participants A and B jointly establish a payment channel on-chain, locking the initial allocation of funds. We denote their initial local copies as S_0^A and S_0^B.
- **Update.** After channel creation, A and B can perform off-chain payments by *updating* their local states. Suppose

A generates a new state S^A_{i+1} and sends it to B. Then B constructs S^B_{i+1}—consistent with S^A_{i+1}—and returns it to A. Importantly, each party revokes their old state (S^A_i, S^B_i) after receiving and validating the new state, preventing a situation where they have no valid state to finalize.

- **Finalization.** The channel can either be closed cooperatively by both parties or unilaterally by either party submitting their latest held state, S^A_i or S^B_i, to forcefully close the channel. Any attempt to submit a revoked state on-chain typically incurs a significant penalty, allowing the honest party to claim the entire channel balance.

While many PCs have distinct implementation details and message sequences (see Figure 1), we formalize that they are united by a *common paradigm*:

- A *core principle* dictating that each valid state is carried by two synchronized local copies with explicit revocation requirement.
- A *three-phase process*: channel creation, off-chain update that involves sending new states and explicitly revoking old states, and finalization that uses a non-revoked state without punishment.

3.2 Balance Security

Balance security is the most critical property of payment channel protocols. Building on prior work [6,28], we formally define it as follows:

Definition 3.2 (Balance Security)**.** *A payment channel protocol* Π *satisfies balance security if, for any honest participant* P *with balance* b *in the latest consensus-approved state* S_i*, the channel cannot be finalized in any other state* S *with a balance* $b' < b$ *for* P*.*

We examine the requirements of balance security for the three phases of payment channel:

- **Creation.** A payment channel is created only through the mutual agreement of both participants, with the locked funds placed under their joint control.
- **Update.** The channel state can only be updated when both participants reach an agreement on the new payment. Once the update is completed, the new state is considered as the latest *consensus-approved* state.
- **Finalization.** When the channel is ultimately closed, an honest participant must receive at least the balance distributed to them in the latest consensus-approved state.

3.3 Non-Atomicity Issue: Race Condition

The *common paradigm* provides an elegant blueprint for PC protocols. However, it fails to guarantee *balance security* due to the presence of race condition vulnerabilities, as illustrated in Figure 2. Below, we provide a detailed explanation.

Let A be the honest payer and B the malicious payee. Suppose they are currently operating on the valid state (S^A_i, S^B_i),

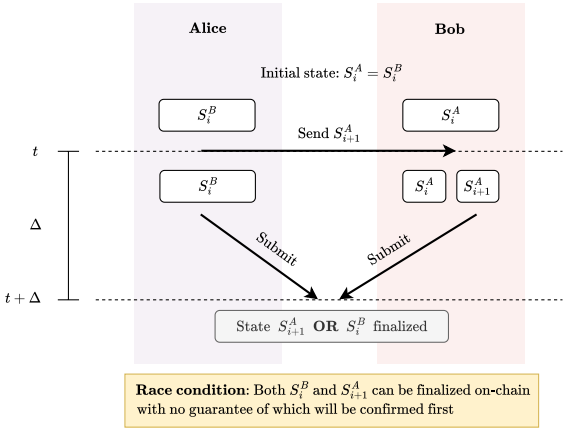

Figure 2: Race condition in the common paradigm: Without atomic updates, either S^A_{i+1} or S^B_i may be finalized, violating balance security.

and A initiates a payment to B to transition the channel to a new state (S^A_{i+1}, S^B_{i+1}). Initially, A sends S^A_{i+1} to B. Upon receiving the new state, B does not respond and instead submits S^A_{i+1} directly on-chain at time t, attempting to finalize the channel using this state. Meanwhile, since A does not receive the expected response, such as a revocation message in the LN protocol or a commitment message representing the new state (S^B_{i+1}) in the SC protocol, A may submit the valid state S^B_i on-chain to forcibly close the channel to protect its funds.

At this point, the update process has not been completed, so (S^A_i, S^B_i) remains the *latest consensus-approved state*. However, due to the asynchronous nature of blockchain transaction ordering and the influence of miners, a race condition window exists during the interval $[t, t+\Delta]$, where Δ represents the upper bound time for transaction confirmation. There is no guarantee as to which of the two states will be finalized first. If S^A_{i+1} is ultimately finalized, party A unwittingly loses funds compared to the latest consensus-approved state (S^A_i, S^B_i), thereby violating the property of balance security.

We highlight that the root cause of this issue lies in the lack of *atomicity* in the update phase of the common paradigm.

Definition 3.3 (Atomicity)**.** *A state transition* $S_i \to S_{i+1}$ *is atomic if and only if:*

$$\forall Q \in \{A, B\}, \forall t : (S^Q_i \neq \bot) \wedge (S^Q_{i+1} \neq \bot) \wedge (S^Q_i \neq S^Q_{i+1}) = \mathsf{false}$$

where t *represents any point in time during the transition process.*

By atomicity, we mean it is impossible for two different valid states (*e.g.*, S^A_i and S^A_{i+1}) to be simultaneously *finalizable* on-chain. We formally prove the impossibility of atomicity in the common paradigm using a proof by contradiction.

Theorem 3.1 (Impossibility of Atomicity in the Common Paradigm)**.** *No protocol strictly adhering to the common paradigm can achieve atomic state update, leaving it vulnerable to race conditions and violating balance security.*

Proof. Assume there exists a protocol Π conforming to the common paradigm, and suppose it satisfy atomicity:

$$\forall t : (S_i^A \neq \bot) \wedge (S_{i+1}^A \neq \bot) \wedge (S_i^A \neq S_{i+1}^A) = \mathsf{false}$$

Here, S_{i+1}^A is the new state generated by A following S_i^A, satisfying $S_i^A \neq S_{i+1}^A$. According to the *State Replacement* principle, once B receives the new state ($\vdash S_{i+1}^A$), it gains the ability to finalize the channel using S_{i+1}^A without incurring any financial loss, meaning $S_{i+1}^A \neq \bot$. However, before B revokes the previous state ($S_i^A \to \bot$), it holds that $S_i^A \neq \bot$. At any time point t_0 between the receipt of the new state and the revocation of the previous state, we have $S_i^A \neq \bot$, $S_{i+1}^A \neq \bot$, and $S_i^A \neq S_{i+1}^A$. These time points t_0 form a time window, which contradicts the assumption. Therefore, the common paradigm fails to satisfy atomicity. □

Note that the processes of receiving new states and revoking old states rely on communication between both parties. Under normal conditions, the duration of this time window is primarily determined by the network latency between the two participants.

The lack of atomicity in the *State Replacement* mechanism during the update phase of the common paradigm introduces the risk of race conditions. Consequently, payment channel protocols adhering to this design are fundamentally unable to satisfy the essential property of *balance security*.

3.4 Remark on Sleepy Channels

The study on Sleepy Channels claims to have formally proven the security of the protocol within the UC framework. However, we point out that since it fundamentally adheres to the common paradigm, its update phase lacks atomicity, resulting in a race condition issue. Therefore, Sleepy Channels fails to achieve the most critical property of payment channel protocols: *balance security*.

Observation 3.2. *In the update phase of Sleepy Channels' ideal functionality, it is explicitly stated that under exceptional circumstances requiring ForceClose, two states may coexist and be used to close the channel without punishment. However, the security definition is insufficiently comprehensive, as it fails to account for the race condition issue. Consequently, the claimed security guarantees are fundamentally flawed.*

4 Atomic Paradigm

4.1 Overview

To fundamentally address the race condition problem in the *common paradigm*, we propose a novel *atomic paradigm* that ensures the protocols adhering to this paradigm satisfy the *balance security* property.

Core Principle: Single-Sided State with Transformation. During each update, the new state is generated solely by a single participant, who then sends it to the other party, while simultaneously all existing states are automatically transformed into this new state.

Definition 4.1 (State Transition)**.** *When participant P sends a new state S_n^P to the other party at t_0, all existing states simultaneously transform into this new state:*

$$\forall Q \in \{A,B\}, \exists P \in \{A,B\}, \forall k, \exists t_0 : S_k^Q \to S_n^P, t \geq t_0$$

where $S_k^Q \to S_n^P$ denotes the automatic transformation of state S_k^Q sent by participant Q into the new state S_n^P.

The channel evolves through three main phases:

- **Creation.** Like common paradigm, both participants jointly establish the channel with initial state (S_0^A, S_0^B).
- **Update.** After creation, any participant can perform updates through *State Transition*. To illustrate this process, consider the following example. Suppose the current states are:
  - Several states generated by A, held by B: $S_0^A, ..., S_i^A$
  - Several states generated by B, held by A: $S_0^B, ..., S_j^B$

  When B initiates a payment by sending a new state S_{j+1}^B to A, all prior states are automatically transformed to the new state: $\forall k \leq i : S_k^A \to S_{j+1}^B$ and $\forall k \leq j : S_k^B \to S_{j+1}^B$.
- **Finalization.** The channel can be closed cooperatively by both parties. Alternatively, either party can finalize the channel using any state they hold, which, due to the *State Transition* during the update phase, will be equivalent to the latest valid state.

We formalize the *atomic paradigm*:

- A *core principle* dictating that each latest valid state is sent by a single participant, with all existing states transforming accordingly.
- A *three-phase process*: channel creation, off-chain updates involving atomic State Transition, and finalization that can utilize any valid state.

4.2 Security Analysis of the Atomic Paradigm

We now prove that our atomic paradigm provides both atomicity and balance security through *State Transition*.

Theorem 4.1 (Atomicity Guarantee)**.** *The atomic paradigm achieves atomic state transitions, preventing any race conditions situation.*

Proof. Assume there exists a protocol Π conforming to the atomic paradigm, and suppose it does not satisfy atomicity:

$$\exists t_0 : (S_i^A \neq \bot) \wedge (S_j^B \neq \bot) \wedge (S_i^A \neq S_j^B) = \mathsf{true}$$

Here, S_j^B is the new state following S_i^A during an update. According to the *State Transition* principle, when B sends S_j^B to A, the state transition $S_i^A \rightarrow S_j^B$ occurs, ensuring that $S_i^A = S_j^B$. Therefore, there cannot exist a time t_0 such that $S_i^A \neq S_j^B$, which directly contradicts the assumption. The paradigm is proven to satisfy atomicity. □

Theorem 4.2 (Balance Security Preservation)**.** *The atomic paradigm maintains balance security across creation, update, and finalization phases according to Definition 3.2.*

Proof. We prove balance security holds for each phase:
Creation. Consistent with the common paradigm, the channel is created only through the mutual agreement of both participants, with the locked funds under joint control. At creation, both parties each hold a valid state reflecting the initial distribution of funds.
Update. An update occurs only when both parties agree on the new state. The update phase adheres to atomicity, satisfying:

$$(S_i^Q \neq \bot) \wedge (S_{i+1}^Q \neq \bot) = \mathsf{false}.$$

This ensures that during the update process, the previous state S_i^Q remains the only valid consensus-approved state. Once the update is completed, the new state S_{i+1}^Q becomes the sole consensus-approved state.
Finalization. In addition to cooperative closure through mutual agreement, either party can finalize the channel using any state they hold. Due to the *State Transition* during the update phase, any such state is guaranteed to be equivalent to the latest consensus-approved state S_n^P, ensuring no loss of funds for the participants.

□

Corollary 4.3 (Security Enhancement)**.** *The atomic paradigm strictly enhances the security of payment channels by providing all security properties while eliminating race conditions through automatic State Transition.*

Therefore, we have shown that our atomic paradigm resolves the race condition vulnerability through its State Transition mechanism while maintaining all security properties.

5 ULTRAVIOLET Protocol

5.1 Protocol Overview

We introduce the ULTRAVIOLET protocol using an incremental construction approach. To develop a protocol that adheres to our new paradigm, we begin with an intuitive toy protocol as a starting point and iteratively refine its design.

(a) Toy protocol

1. B sends the revocation secret for S_{i-1}^A, if not sent yet.
2. A sends the revocation secret for S_j^B, if not sent yet.
3. A sends the new state S_i^A to B.

(b) Remove Punishment Mechanism

1. A sends the new state S_i^A to B.

(c) Introduce Resolve Mechanism

1. A sends the new state S_i^A to B, along with the set of resolve states $\{S_{A,i}^{\hookrightarrow S_n^B} \mid n \in [0, j]\}$.

(d) Final Protocol

1. B sends the revocation secret for S_{i-2}^A, if such a state exists.
2. A sends the new state S_i^A to B, along with the resolve state $S_{A,i}^{\hookrightarrow S_j^B}$ and, if S_{j-1}^B exists, the resolve state $S_{A,i}^{\hookrightarrow S_{j-1}^B}$.

Figure 3: Incremental Construction of the ULTRAVIOLET Protocol, illustrating the evolution from (a) Toy Protocol to (d) Final Protocol, with focus on the Update phase for Party A's i-th payment, assuming B has completed j payments.

For clarity, we focus on the design of Update phase, which differentiates our protocol from the common paradigm followed by traditional Lightning-style channels. We assume that parties A and B have created a channel and possess an initial publishable state, S_0^B and S_0^A, respectively.

Toy Protocol. Figure 3(a) illustrates the state update process of a toy protocol. During each update, if the payee holds a valid state, they first revoke it. We recall that a state is 'valid" means that this state can be submitted to the blockchain and it cannot occur any finance loss (punishment). Then, if the payer also holds a valid state, they revoke it as well and proceed to send the new state to the payee, completing the payment. This simple design ensures that no two valid states exist simultaneously during the update phase, eliminating the race conditions inherent in the common paradigm and guaranteeing atomicity in the state update process.

However, this protocol also introduces significant issues. If A (the payer) is malicious while B (the payee) is honest, A can withhold any response after B revokes their valid state. In this case, B cannot proceed with a cooperative finalization, nor can they perform a forced finalization since they no longer hold any valid state. Similarly, if A is honest and B is malicious, A completes the state update by revoking their valid state and sending the new state to B. At this point, B holds the only valid state, and if B becomes unresponsive, A is left unable to finalize the channel, either cooperatively or forcibly.

Thus, while this toy protocol addresses the race condition issue, it introduces critical fairness and liveness issues. When either party is malicious or unresponsive, the honest party is

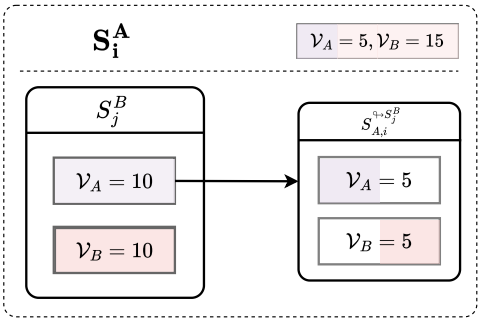

Figure 4: An illustrative example of the resolve mechanism.

vulnerable to a *denial-of-service* (DoS) attack, leaving them unable to finalize the channel forcibly. This could result in funds being indefinitely locked within the channel, rendering the protocol practically unusable.

Remove Punishment Mechanism. To address the issues in the toy protocol, one improvement is to remove the punishment mechanism entirely. As shown in Figure 3(b), the first two steps of the toy protocol related to punishment are removed, simplifying the process. During each state update, the payer directly sends the new state to the payee. In this approach, if one party becomes unresponsive during cooperative finalization, the honest party can commit their latest valid state to finalize the channel forcibly, allowing them to withdraw funds from the channel.

However, this straightforward modification introduces a significant drawback. It allows either party to hold multiple valid states simultaneously, violating the atomicity emphasized in our new paradigm and reintroducing the race condition problem. Consequently, this design fails to ensure that the finalized state is the latest one. A malicious participant could exploit this by competing to submit an outdated but more favorable state for finalization, undermining the security of the protocol.

Introduce Resolve Mechanism. Previous protocol designs exhibit significant shortcomings: either honest users are vulnerable to DoS attacks, leading to locking of funds, or state updates lack atomicity, failing to ensure that the channel is finalized with the latest state. To address these issues, we propose a novel Resolve Mechanism, as depicted in Figure 3(c). In this design, during each state update, the payer sends the new state to the payee along with a set of resolve states, each corresponding to a previously held state by the payer.

To clarify the principle of the resolve mechanism, consider the following example. Suppose B has just completed its j-th payment to A. At this point, A holds a set of valid states, denoted as $S^A_{\text{set}} = \{S^B_n \mid n \in [0, j]\}$, where the latest state is S^B_j. As shown in Figure 4, let S^B_j represent a channel state where A's balance V_A is 10 and B's balance V_B is also 10. Now, If A initiates its i-th payment to B, transferring an amount of 5, the new state S^A_i, reflecting updated balances of $V_A = 5$ and $V_B = 15$, is sent to B together with a set of resolve states constructed for every state in S^A_{set}. These resolve states allow B, within a predefined time window, to redistribute the funds belonging to A from any of states held by A. Specifically, for S^B_j, the corresponding resolve state $S^{\hookrightarrow S^B_j}_{A,i}$ allows B to redistribute the 10 units of funds originally allocated to A, transferring 5 units to B itself as per the updated state S^A_i. In this way, if A commits outdated state S^B_j for finalization, B can immediately use the corresponding resolve state to effectively perform a state transition, ensuring that the channel is finalized with the latest state S^A_i. Conceptually, this can be expressed as: $S^B_n \xrightarrow{S^{\hookrightarrow S^B_n}_{A,i}} S^A_i$, for $n \in [0, j]$. In this manner, the resolve mechanism ensures that outdated states can be transitioned to the latest state during finalization.

It is worth noting that when the balance belonging to A in an outdated state S^B_n is less than A's balance in the latest state S^A_i—for example, when $V_A = 0$ and $V_B = 20$ in S^B_0—the corresponding resolve state cannot redistribute funds to align with the latest state, as it can only reallocate funds originally held by the payer A. In such cases, a rational A would never attempt to finalize the channel with S^B_0, as doing so would lead to a financial loss compared to the latest state. For simplicity, our discussion excludes such state where only irrational participants would commit.

The introduction of the resolve mechanism effectively addresses the security issues identified in the previous protocol designs:

- *DoS Attacks.* If B is honest while A is malicious or unresponsive, B can forcibly finalize the channel by committing the latest state S^A_i. Conversely, if A is honest and B is malicious or unresponsive, the rational A can commit the latest state S^B_j held. During the designated time window, if B commits the corresponding resolve state, the channel will be finalized in the globally latest state S^A_i. Otherwise, the channel will be finalized in the state S^B_j, resulting in a financial loss for the malicious B, while the honest A receives more funds than they are entitled to under the globally latest state S^A_i.
- *Atomicity.* During the state update, the payer sends the new state and the resolve states for all outdated states held. This ensures that if a malicious party attempts to finalize the channel with a more favorable outdated state, the honest party can commit the corresponding resolve state to transition the channel to the latest state. In other words, for every outdated state that a rational user may commit, the counterparty holds the corresponding resolve state, which could transition it to the latest state. In this way, each outdated state can also be considered as having been updated to the new state by sending resolve state, and the update can be seen as a one-time completion, which conforms to atomicity.

This protocol design adheres to our new paradigm, ensuring atomicity while maintaining balance security. However, this

design also presents certain limitations:

- *Storage Cost:* Each party must store resolve states for all historical states. For a channel with n states, this results in $O(n^2)$ storage complexity, as each new state necessitates resolve states for all prior states.
- *Communication Overhead:* The number of resolve states transmitted increases linearly with the channel updates, leading to significant communication overhead.
- *State Management:* Parties must carefully track and organize all historical states and their corresponding resolve states, making the implementation more complex and error-prone.

These concerns make this design impractical for long-lived payment channels with frequent updates.

Final Protocol. To address the limitations of the previous design, we reintroduce the punishment mechanism as a means of pruning, ultimately arriving at the final protocol. As depicted in Figure 3(d), during each state update, the payee first revokes their second-to-last state. Subsequently, the payer sends the resolve states for the latest two states held, along with the new state, to the payee.

This design retains the security guarantees of the previous protocol while significantly reducing storage and communication costs:

- *Storage and Communication.* By requiring the payee to revoke their second-to-last state before receiving the new state, the protocol ensures that no more than two valid states exist at any given time for either party. Consequently, the resolve states only need to be sent and stored for the two latest states. This reduces communication overhead to a constant level per update and decreases the storage complexity of resolve states from $O(n^2)$ to $O(1)$ by merely introducing an $O(n)$ storage requirement for revocation secrets.
- *Balance Security.* Similar to the previous design, the atomicity of state updates is maintained by requiring the payer to send resolve states along with the new state to the payee. Furthermore, when one party is malicious or unresponsive for collaborative finalization, the honest party can commit their latest valid state. If this state is not the globally latest state, the malicious party is forced to commit the corresponding resolve state to avoid financial loss. This ensures that the channel is finalized in the globally latest state. Specifically, even if A receives the revocation secret during a state update but refuses to send the new state to B, B still retains an valid state that has not been revoked, which can be used for finalization. This effectively prevents DoS attacks.

5.2 Details of ULTRAVIOLET

We provide a detailed exposition of the ULTRAVIOLET protocol, designed to enable secure, efficient, flexible, and compatible off-chain payments between two parties. The protocol strictly adheres to the principles of the atomic paradigm, guaranteeing atomicity and balance security across all phases of its execution. It is systematically divided into three phases: *creation*, *update*, and *finalization*, incorporating novel resolve and punishment mechanisms. The complete workflow is depicted in Figure 5. The formal protocol modelled in UC framework can be found in Appendix A.3.

Suppose party A and B fund the payment channel $\mathcal{C}$ using v^A from tx_{A_F} and v^B from tx_{B_F}, creating a channel with a total capacity of $v = v^A + v^B$. We assume that (pk_A, sk_A) and (pk_B, sk_B) serve as the authentication key pairs for tx_{A_F} and tx_{B_F}, respectively, and will also be used in subsequent payments. Let A_i and B_j denote the channel states, where A_i corresponds to party A's i-th payment B_j corresponds to B's j-th payment, each reflecting the fund distribution following that payment.

Creation. To establish the channel, we follow a process similar to Lightning-style channels: both parties lock their funds into a shared funding transaction tx_F, funded from their respective transactions tx_{F_A} and tx_{F_B}. This creates a channel with a total capacity $v = v^A + v^B$, where v^A and v^B represent the funds contributed by party A and party B, respectively. Before broadcasting tx_F, they sign the refund transaction with each other in advance, ensuring their funds can be reclaimed if necessary after the channel is established. These refund transactions, which distribute the channel funds according to the initial balances v^A and v^B, are effectively equivalent to zero-value payments executed between the parties prior to channel creation, consistent with the structure of the payment transaction during the update phase.

For better readability, we assume each party executes two such refund transactions in advance, ensuring both parties have two valid states during the update phase. Note that this is for illustrative purposes only, a single such transaction suffices in practice.

Update. The state update design is the core of our protocol, enabling secure and efficient off-chain payments. The following explanation is presented from the perspective of party A's i-th payment, assuming party B has completed j payments.

The update process begins with revoking the outdated state. Party B, the payee, generates a new revocation secret and hash pair (rh_{B_i}, rs_{B_i}) and sends rh_{B_i} along with the revocation secret $rs_{B_{i-2}}$ for revoking the outdated state A_{i-2} to party A, the payer.

Upon receiving $rs_{B_{i-2}}$ and rh_{B_i}, A generates the i-th payment transaction $tx_{A_i} := \mathsf{tx}(tx_F, [\langle sk_A, v^A_{A_i}\rangle, \langle (rs_{B_i} \wedge sk_A) \vee (sk_A \wedge sk_B) \vee (\mathsf{RelTime}(T) \wedge sk_B), v^B_{A_i}\rangle])$. This payment transaction updates the channel state to reflect the balances corresponding to state A_i, where $v^A_{A_i}$ and $v^B_{A_i}$ are the respective balances of A and B after this payment. As illustrated in Figure 6, tx_{A_i} contains two outputs: one representing $v^A_{A_i}$, which A can claim immediately upon confirmation on the blockchain, and the other representing $v^B_{A_i}$, which can be spent by different conditional branches. Under normal circumstances, B must

Creation. Before establishing the channel, both parties must first obtain the refund transaction along with respective signature. The following describes the process from party A's perspective, with party B following a symmetric procedure. The steps are as follows:

1. Generate two revocation secret and hash pairs $(rh_{A_{-1}}, rs_{A_{-1}})$, (rh_{A_0}, rs_{A_0}), sends $rh_{A_{-1}}$ and rh_{A_0} to party B.
2. Upon receiving the revocation hash $rh_{A_{-1}}$ and rh_{A_0} from the B, generate funding transaction $tx_F := tx([tx_{A_F}, tx_{B_F}], \langle sk_A \wedge sk_B, v \rangle)$, along with $tx_{A_{-1}} := \mathrm{tx}(tx_F, [\langle sk_A, v^A \rangle, \langle (rs_{B_{-1}} \wedge sk_A) \vee (sk_A \wedge sk_B) \vee (\mathsf{RelTime}(T) \wedge sk_B), v^B \rangle])$ and $tx_{A_0} := \mathrm{tx}(tx_F, [\langle sk_A, v^A \rangle, \langle (rs_{B_0} \wedge sk_A) \vee (sk_A \wedge sk_B) \vee (\mathsf{RelTime}(T) \wedge sk_B), v^B \rangle])$.
3. Generate signatures σ_F^A, $\sigma_{A_{-1}}^A$ and $\sigma_{A_0}^A$ for transactions tx_F, $tx_{A_{-1}}$ and tx_{A_0}, respectively, using sk_A.
4. Send $tx_{A_{-1}}$, tx_{A_0} and corresponding signatures $\sigma_{A_{-1}}^A$ and $\sigma_{A_0}^A$ to B.
5. Upon receiving $tx_{B_{-1}}$, tx_{B_0} and corresponding signatures $\sigma_{B_{-1}}^B$ and $\sigma_{B_0}^B$ from B, send σ_F^A to B.
6. Upon receiving σ_F^B from B, post $(tx_F, \{\sigma_F^A, \sigma_F^B\})$ on $\mathbb{B}$. Once tx_F is confirmed on $\mathbb{B}$, the channel is successfully created.

***i*-th Payment.** For party A's i-th payment, assuming party B has completed j payments, once both parties have agreed on this payment, the process unfolds as follows:

First, party B, the payee:

1. generate a new revocation secret and hash pair (rh_{B_i}, rs_{B_i}), then send rh_{B_i} to party A for the current payment, along with $rs_{B_{i-2}}$ to revoke the outdated payment transaction $tx_{A_{i-2}}$.

Then, party A, the payer:

1. Upon receiving rh_{B_i} and $rs_{B_{i-2}}$, generate i-th payment transaction $tx_{A_i} := \mathrm{tx}(tx_F, [\langle sk_A, v_{A_i}^A \rangle, \langle (rs_{B_i} \wedge sk_A) \vee (sk_A \wedge sk_B) \vee (\mathsf{RelTime}(T) \wedge sk_B), v_{A_i}^B \rangle])$, reflecting the updated balance state.
2. Generate resolve transaction $tx_{A_i}^{\curvearrowright tx_{B_j}} := tx(tx_{B_j}, [\langle sk_A, v_{A_i}^A \rangle, \langle sk_B, v_{B_j}^A - v_{A_i}^A \rangle])$ corresponding to j-th payment tx_{B_j} from party B.
3. If $v_{A_i}^A \leq v_{B_{j-1}}^A$, generate resolve transaction $tx_{A_i}^{\curvearrowright tx_{B_{j-1}}} := tx(tx_{B_{j-1}}, [\langle sk_A, v_{A_i}^A \rangle, \langle sk_B, v_{B_{j-1}}^A - v_{A_i}^A \rangle])$. Otherwise, generate resolve transaction $tx_{A_i}^{\curvearrowright tx_{B_{j-1}}} := tx(tx_{B_{j-1}}, [\langle sk_A, v_{B_{j-1}}^A \rangle, \langle sk_B, 0 \rangle])$ corresponding to $(j$-1)-th payment $tx_{B_{j-1}}$ from party B
4. Generate signatures $\sigma_{A_i}^A$, $\sigma_{A_i(B_j)}^A$ and $\sigma_{A_i(B_{j-1})}^A$ for tx_{A_i}, $tx_{A_i}^{\curvearrowright tx_{B_j}}$ and $tx_{A_i}^{\curvearrowright tx_{B_{j-1}}}$, respectively, using sk_A.
5. Send the payment transaction tx_{A_i}, along with the resolve transactions $tx_{A_i}^{\curvearrowright tx_{B_j}}$ and $tx_{A_i}^{\curvearrowright tx_{B_{j-1}}}$ to party B.

Finalization. The channel can be finalized either collaboratively or forcibly by any party. Assuming that party A has completed i payments and party B has completed j payments, with tx_{A_i} as the latest payment transaction and A_i as the globally latest state. Collaborative finalization proceeds as follows:

1. Party A generates $tx_C := tx(tx_F, [\langle sk_A, v_{A_n}^A \rangle, \langle sk_B, v_{A_n}^B \rangle])$, sign it with sk_A to produce the signature σ_C^A, and sends (tx_C, σ_C^A) to B.
2. Party B signs tx_C to produce σ_C^B, and post $(tx_C, \{\sigma_C^A, \sigma_C^B\})$ on $\mathbb{B}$, finalizing the channel in the globally latest state A_i.

For party A, the forced finalization proceeds as follows:

1. Party A uses sk_A to generate signature $\sigma_{B_j}^A$ for transaction tx_{B_j} and post $(tx_{B_j}, \{(\sigma_{B_j}^A, \sigma_{B_j}^B)\})$ on $\mathbb{B}$.
2. Upon tx_{B_j} is confirmed on $\mathbb{B}$, party B uses sk_B to generate the signature $\sigma_{A_i(B_j)}^B$ for the transaction $tx_{A_i}^{\curvearrowright tx_{B_j}}$, then posts $(tx_{A_i}^{\curvearrowright tx_{B_j}}, \{\sigma_{A_i(B_j)}^A, \sigma_{A_i(B_j)}^B\})$ on $\mathbb{B}$. If $tx_{A_i}^{\curvearrowright tx_{B_j}}$ is confirmed on $\mathbb{B}$ before the relative time T, the channel finalizes in the globally latest state A_i.

For party B, the forced finalization proceeds as follows:

1. Party B uses sk_B to generate $\sigma_{A_i}^B$ for tx_{A_i} and post $(tx_{A_i}, \{(\sigma_{A_i}^A, \sigma_{A_i}^B)\})$ on $\mathbb{B}$, finalizing the channel in the latest state A_i.

Punishment. If party A observes that a revoked payment tx_{A_k}, corresponding to the k-th payment, has been posted by B and confirmed on the ledger $\mathbb{B}$, it proceeds with the following steps to execute the punishment:

1. Generate the punishment transaction $tx_{pnsh}^{A_k} := tx(tx_{A_k}, \langle sk_A, v_{A_k}^B \rangle)$ corresponding to tx_{A_k}.
2. Generate signature $\sigma_{pnsh}^{A_k}$ for punishment transaction $tx_{pnsh}^{A_k}$ using sk_A and rs_{B_k}.
3. Post $(tx_{pnsh}^{A_k}, \sigma_{pnsh}^{A_k})$ on $\mathbb{B}$ and to get all funds within the channel.

Figure 5: ULTRAVIOLET Protocol

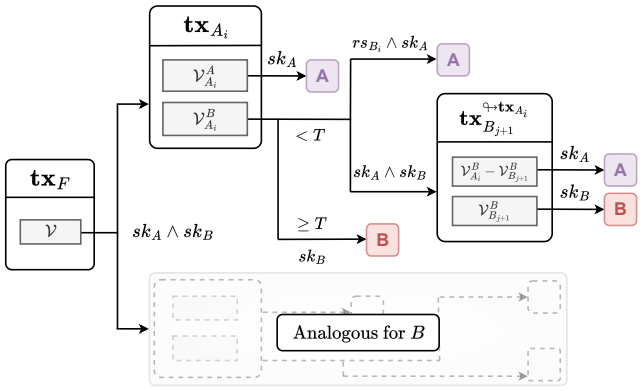

Figure 6: Flow for payment and resolve transaction.

wait for a relative time T to claim $v^B_{A_i}$. During this period, either due to the punish mechanism, A can punish and take away the funds $v^B_{A_i}$ that originally belonged to B, or due to the resolve mechanism, this portion of funds can be redistributed to A and B by a subsequent resolve transaction.

In addition to the payment transaction tx_{A_i}, party A generates resolve transactions for the two valid payments it holds, tx_{B_j} and $tx_{B_{j-1}}$ corresponding to state A_i. the resolve transaction $tx_{A_i}^{\curvearrowright tx_{B_j}}$ for tx_{B_j} is structured as $tx(tx_{B_j}, [\langle sk_A, v^A_{A_i}\rangle, \langle sk_B, v^A_{B_j} - v^A_{A_i}\rangle])$. Figure 6 illustrates the flow of the resolve transaction $tx_{B_{j+1}}^{\curvearrowright tx_{A_i}}$ generated by B for tx_{A_i} in a potential future scenario where B initiates the $(j+1)$-th payment to A. In our current scenario, the structure of $tx_{A_i}^{\curvearrowright tx_{B_j}}$ and $tx_{A_i}^{\curvearrowright tx_{B_{j-1}}}$ are symmetric to $tx_{B_{j+1}}^{\curvearrowright tx_{A_i}}$, redistributing the portion of funds in outdated payment tx_{B_j} and $tx_{B_{j+1}}$ that originally belong to A to transition their state to the globally latest state A_i.

It should be noted that for the latest valid payment tx_{B_j} held by A, the funds originally belonging to A can always be redistributed to match the globally latest state A_i. This is because the payment between tx_{B_j} and tx_{A_i} can only be a payment from A to B; otherwise, j would not represent the latest payment from B to A. Hence, it is guaranteed that $v^A_{A_i} \leq v^A_{B_j}$. However, due to the existence of the payment $tx_{B_{j-1}}$ from B to A between payments $tx_{B_{j-1}}$ and tx_{A_i}, it follows that $v^A_{B_{j-1}} < v^A_{B_j}$. Consequently, there may be cases where $v^A_{A_i} > v^A_{B_{j-1}}$, preventing $tx_{B_{j-1}}$ from being redistributed to match the state A_i. In this case, the corresponding resolve transaction $tx_{A_i}^{\curvearrowright tx_{B_{j-1}}}$ is structured as $tx(tx_{B_{j-1}}, [\langle sk_A, v^A_{B_{j-1}}\rangle, \langle sk_B, 0\rangle])$.

After signing the newly generated payment transaction and the two resolve transactions, A sends them together to B. These innovative payment and resolve transactions in the update phase will play a crucial role in the finalization phase.

Finalization. Similar to Lightning-style channels, the established channel can be efficiently finalized collaboratively with a single transaction, tx_C, when both parties agree on the fund distribution. Here, we focus on scenarios where one party is unresponsive or malicious, and the other party attempts to finalize the channel forcibly. Assuming that party A has completed i payments and party B has completed j payments, with tx_{A_i} representing the latest payment transaction between them.

For party B, to finalize channel forcibly, B can directly post tx_{A_i}, which represents the globally latest state A_i. Once the transaction is confirmed on the blockchain, A will immediately receive their share of the funds, $V^A_{A_i}$, while B will receive their share, $V^B_{A_i}$, after waiting for the relative time T, ensuring the channel finalizes in the latest state A_i.

For a rational party A, to finalize channel forcibly, A will post the latest valid transaction tx_{B_j} they possess. Once the transaction is confirmed on the blockchain, B will immediately receive their share of the funds, $V^B_{B_j}$. Based on the earlier proof that $v^A_{A_i} \leq v^A_{B_j}$, it follows that $v^B_{A_i} \geq v^B_{B_j}$. Consequently, a rational B will post the resolve transaction $tx_{A_i}^{\curvearrowright tx_{B_j}}$ corresponding to tx_{B_j} within the relative time T, redistributing the funds originally belonging to A in tx_{B_j} to prevent any financial loss. This ensures the channel ultimately finalizes in the globally latest state A_i.

Punishment. According to our protocol design, each party holds at most two valid states during the existence of the channel. If a party attempts to post a revoked transaction, the counterparty can utilize the corresponding revocation secret obtained during the update phase to post a punishment transaction, thereb claiming the entire funds of the channel.

5.3 Analysis

Security. We informally discussed protocol security in Section 5.1. To formally model and prove the security of our protocol, we employ the synchronous Global Universal Composability (GUC) framework [13], following prior work [6–8, 18, 19, 21]. This framework enables us to model our protocol within a global setup that incorporates a global ledger $\mathbb{B}(\Delta, \Sigma, \mathcal{V})$, a global clock $\mathcal{F}_{\text{clock}}$, and $\mathcal{F}_{\text{GDC}}$. Due to space constraints, we provide a high-level overview of our security model here, with the complete formal description presented in Appendix A.

Firstly, we define the ideal functionality $\mathcal{F}$ that abstracts the expected operations within our protocol and ensures balance security during different phase. $\mathcal{F}$ stipulates the input/output behavior from our protocol, along with its impact on the global setup and potential adversarial actions. In the ideal world, all parties interact solely with $\mathcal{F}$, which acts as a trusted third party. We then formally define the real-world protocol Π in the UC framework. To prove that our protocol realizes $\mathcal{F}$, we construct a simulator S that translates any potential attack on the protocol Π into an equivalent attack on the ideal functionality $\mathcal{F}$. In essence, this means that Π is as secure as $\mathcal{F}$, as any actions that an adversary could take in the real protocol

can also be simulated in the ideal setting. Our proof hinges on showing that no probabilistic polynomial-time (PPT) environment $\mathcal{E}$ can distinguish between interactions with the real-world protocol Π and interactions with the ideal functionality $\mathcal{F}$ with non-negligible probability. To achieve this indistinguishability, we ensure that the simulator S replicates the behavior of Π by producing identical outputs and ledger transactions in matching rounds for both real and ideal executions. This guarantees that messages, transactions, and their timing align in both worlds. Consequently, the environment $\mathcal{E}$ observes the same information in both scenarios, preventing it from distinguishing between real and ideal protocol executions.

Efficiency. ULTRAVIOLET achieves high efficiency by avoiding the use of any computationally expensive cryptographic primitives. Structurally, each state update requires only two communication messages: a revocation message from the payee to the payer and a state update message from the payer to the payee. This design reduces the communication overhead by half compared to the traditional PCs, e.g., LN and SC, which requires four communication messages per update.

Flexibility. ULTRAVIOLET demonstrates remarkable flexibility by supporting general bidirectional payment channels, allowing either party to initiate payments to the other. Additionally, it avoids the use of absolute time locks in transaction construction, enabling an unrestricted channel lifetime—so long as both parties agree to keep the channel open, it remains valid without requiring any additional operations. Furthermore, the absence of counters or similar mechanisms ensures the protocol supports unbounded transactions, accommodating an indefinite number of updates.

Compatibility. Our protocol is designed with strong compatibility, relying solely on existing Bitcoin features, such as multi-signature and relative time locks, to achieve its functionality. By avoiding the introduction of any extra complex cryptographic primitives or unimplemented Bitcoin features, ULTRAVIOLET ensures seamless integration with existing blockchain systems, maintaining compatibility with the current ecosystem.

6 Evaluation

6.1 Implementation

We implement the complete ULTRAVIOLET protocol described in Section 5 using approximately 8 000 lines of Go code. The implementation leverages gRPC for network communication, employs 2 048-bit ECDSA for digital signatures, and uses 256-bit SHA256 as the hash function. Additionally, it utilizes the `OP_CHECKMULTISIG` opcode to construct multisignature addresses as payment channels and adopts a revocation and punishment mechanism based on providing the preimage (revocation secret) and signatures.

To ensure a fair comparison, we also implement the Lightning Network and Sleepy Channels using the same technology stack (Golang, gRPC, 2 048-bit ECDSA, and 256-bit SHA256) and similar mechanisms (`OP_CHECKMULTISIG` for constructing multisignature addresses and the preimage and signatures for revocation).

6.2 Performance Evaluation

Experimental Setup. To evaluate the performance of ULTRAVIOLET, we conduct experiments and compare it against two SOTA protocols: Lightning Network and Sleepy Channels. The experiment involves performing 10 000 sequential payments for each protocol, repeated five times. Sequential payments mean that each payment is initiated only after the previous one is fully completed.

During the experiments, we measure the latency of each payment, defined as the time from initiating a transaction to completing it. From these measurements, we compute the average latency and the 95th-percentile latency for each protocol. Additionally, we calculate the transactions per second (TPS) achieved by each protocol.

The experiments were conducted on Azure D8s_v4 virtual machines, distributed across four distinct regions: West US (Virginia), East US (Phoenix), South UK (London), and Southeast Asia (Singapore). Each node is equipped with eight virtual CPUs (vCPUs) powered by Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz processors, 32 GB of memory, and a network bandwidth of 12 500 Mbps. All nodes run on Ubuntu 24.04 LTS system.

In our experiments, we designate the node in Virginia as the payer, initiating transactions to nodes in other regions (Phoenix, London, and Singapore) which serve as payees.

Latency Analysis. Figure 7 shows the average transaction latency across different regions. ULTRAVIOLET achieves significantly lower latency compared to existing solutions, with average transaction completion times of 54.38ms, 132.11ms, and 180.84ms in Phoenix, London, and Singapore respectively. This represents a 1.59× improvement over LN and a 2.10× improvement over SC. The 95th percentile error bars demonstrate ULTRAVIOLET's stability, with maximum variations of 0.6ms, 1.96ms, and 1.69ms across regions, compared to higher variations in LN (up to 4.18ms) and SC (up to 8.37ms). The increasing latency from Phoenix to Singapore reflects the dominant impact of network latency on transaction times. ULTRAVIOLET's superior performance stems from its optimized commitment scheme that reduces round-trip communications, requiring fewer message exchanges for each transaction.

Throughput Analysis. Figure 8 presents the throughput achieved by each system. ULTRAVIOLET achieves 18.39 TPS in Phoenix, 7.57 TPS in London, and 5.53 TPS in Singapore, showing comparable performance to LN (18.23 TPS, 7.66 TPS, 5.46 TPS). Both systems significantly outperform SC

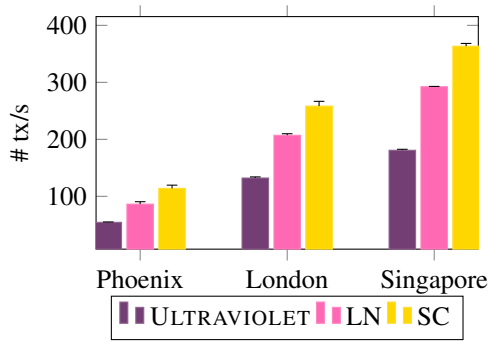

Figure 7: Average transaction latency across different regions with 95th percentile error bars.

(8.79 TPS, 3.87 TPS, 2.75 TPS) by approximately 2×.

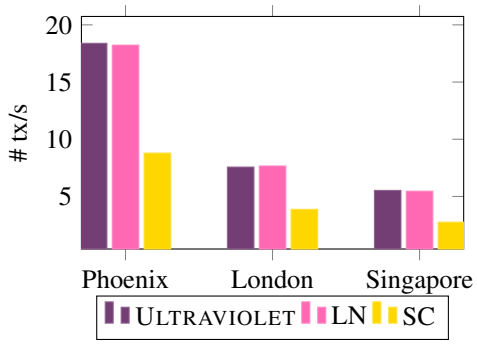

Figure 8: Average throughput across different regions.

6.3 On-chain Cost Analysis

In the context of unilateral channel closure without cooperation, where both parties withdraw their funds from the channel (i.e., no punishment is triggered), ULTRAVIOLET and LN require a single on-chain transaction, while SC require two on-chain transactions.

For unilateral closure transactions, when aligning the implementations of the three protocols to a unified approach (i.e., using `op_multisig` to create a multisig address and utilizing revocation secrets and signatures for revocation and punishment mechanisms), the transaction size in ULTRAVIOLET is slightly larger than that of the LN. This is due to an additional conditional branch needed to support resolving transactions. On the other hand, the transaction size in SC is even larger, as it introduces not only an additional conditional branch but also an additional output to accommodate the specific design of its protocol.

7 Related Works

There are several research directions in off-chain payments as a scalability solution for first-generation blockchains. In this section, we provide a brief overview of related work and position ULTRAVIOLETwithin this context.

Payment Channel Protocols. Numerous protocols have been proposed to enable efficient and secure off-chain transactions. Early efforts focused on *unidirectional* channels, supporting transfers in a single direction, while later *bidirectional* designs allowed participants to act as both sender and receiver [16]. Subsequent research addressed challenges such as privacy [24], multi-party support [23], and reliance on watchtowers [9, 34]. Recently, [5] tackled security issues involving rational miners, differing from our focus. Beyond traditional payment channels, state channels [20] introduced support for state transitions but depend heavily on blockchain scripting capabilities. Generalized channels [6] further expanded functionality by lifting blockchain-supported operations to the off-chain setting, enabling broader applications.

Payment Channel Networks (PCNs) and Hubs (PCHs). The concept of payment channels can be extend to payment channel networks where two users without a direct channel can connect with each other through a path of other people's payment channels via a routing mechanism. One of the most popular implementation is the Basis of Lightning Technology (BOLTs) powering the Lightning Network. Moreover, Payment Channel Hubs deploy a star topology of users enabling them to pay each other via payment channels established with an intermediary called the *tumbler*. PCNs and PCHs have been widely studied in the past, and multiple constructions have been created to challenges such as privacy-protection [17, 25, 32], channel re-balancing [22], better payment routing [29], etc.

8 Conclusion

In this paper, we formalize existing payment channel protocols, including the Lightning Network and Sleepy Channels, into a common paradigm and prove that non-atomic state transitions create a fundamental vulnerability with race conditions that can lead to financial losses. To address this fundamental limitation, we propose a novel atomic paradigm that ensures atomic state transitions while preserving essential functionalities. Based on this paradigm, we develop ULTRAVIOLET, a secure and efficient payment channel protocol that achieves both atomicity and high performance while avoiding the introduction of unimplemented Bitcoin features. Due to reducing the number of required messages per transaction by half, ULTRAVIOLET achieves significant latency improvements of 37% and 52% compared to the Lightning Network and Sleepy Channels, respectively, while achieving throughput comparable to the Lightning Network and twice that of Sleepy Channels. Our formal security analysis under the Universal Composability framework and extensive practical evaluations demonstrate that ULTRAVIOLET provides a secure and efficient solution for scaling blockchain systems.

References

[1] Lightning Network, 2020. https://github.com/lightningnetwork/lnd.

[2] Bitcoin explorer, explore the full bitcoin ecosystem. https://mempool.space/lightning, 2024.

[3] CoinMarketCap. https://coinmarketcap.com/, 2024.

[4] Payment channels wiki. https://en.bitcoin.it/wiki/Payment_channels, 2024.

[5] Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, and Subhra Mazumdar. Securing lightning channels against rational miners. Cryptology ePrint Archive, Paper 2024/826, 2024.

[6] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized channels from limited blockchain scripts and adaptor signatures. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part II 27*, pages 635–664. Springer, 2021.

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

[8] Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Blitz: Secure {Multi-Hop} payments without {Two-Phase} commits. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 4043–4060, 2021.

[9] 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*, pages 179–192, 2022.

[10] Georgia Avarikioti, Felix Laufenberg, Jakub Sliwinski, Yuyi Wang, and Roger Wattenhofer. Towards secure and efficient payment channels. *arXiv preprint arXiv:1811.12740*, 2018.

[11] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. *Journal of Cryptology*, 37(2):18, 2024.

[12] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[13] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007. Proceedings 4*, pages 61–85. Springer, 2007.

[14] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains: (a position paper). In *International conference on financial cryptography and data security*, pages 106–125. Springer, 2016.

[15] Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. eltoo: A simple layer2 protocol for bitcoin. *White paper: https://blockstream. com/eltoo. pdf*, 2018.

[16] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Stabilization, Safety, and Security of Distributed Systems: 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings 17*, pages 3–18. Springer, 2015.

[17] Maya Dotan, Saar Tochner, Aviv Zohar, and Yossi Gilad. Twilight: A differentially private payment channel network. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 555–570, 2022.

[18] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse, and Kristina Hostáková. Multi-party virtual state channels. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*, pages 625–656. Springer, 2019.

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

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

[21] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 949–966, 2018.

[22] Zhonghui Ge, Yi Zhang, Yu Long, and Dawu Gu. Shaduf: Non-cycle payment channel rebalancing. In *NDSS*, 2022.

[23] Zhonghui Ge, Yi Zhang, Yu Long, and Dawu Gu. Magma: Robust and flexible multi-party payment channel. *IEEE Transactions on Dependable and Secure Computing*, 20(6):5024–5042, 2023.

[24] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 473–489, 2017.

[25] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *Network and distributed system security symposium*, 2017.

[26] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *Theory of Cryptography Conference*, pages 477–498. Springer, 2013.

[27] Bowen Liu, Pawel Szalachowski, and Siwei Sun. Failsafe watchtowers and short-lived assertions for payment channels. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 506–518, 2020.

[28] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 455–471, 2017.

[29] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. In *International conference on financial cryptography and data security*, pages 508–526. Springer, 2019.

[30] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[31] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 643–673. Springer, 2017.

[32] Xianrui Qin, Shimin Pan, Arash Mirzaei, Zhimei Sui, Oğuzhan Ersoy, Amin Sakzad, Muhammed F Esgin, Joseph K Liu, Jiangshan Yu, and Tsz Hon Yuen. Blindhub: Bitcoin-compatible privacy-preserving payment channel hubs supporting variable amounts. In *2023 IEEE symposium on security and privacy (SP)*, pages 2462–2480. IEEE, 2023.

[33] Ben Weintraub, Satwik Prabhu Kumble, Cristina Nita-Rotaru, and Stefanie Roos. Payout races and congested channels: A formal analysis of security in the lightning network. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 2562–2576, 2024.

[34] Yankai Xie, Ruian Li, Yan Huang, Chi Zhang, Lingbo Wei, and Yani Sun. A monitoring-free bitcoin payment channel scheme with support for real-time settlement. *IEEE Transactions on Services Computing*, 2024.

A UC Modeling

In this section, we formalize the security of our protocol using the Universal Composability (UC) framework [12], specifically employing the Global UC (GUC) extension [13], an extension of the standard UC framework that enables a global setup.

A.1 Preliminaries

Since our model closely aligns with model established by previous off-chain payment protocols [6–8, 18, 19, 21], we begin by introducing key concepts that have been widely adopted in these earlier works.

Protocols and Adversarial Model. In the real world, a protocol Π is executed by a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ in the presence of an adversary $\mathcal{A}$, who receives a security parameter $\lambda \in \mathbb{N}$ and an auxiliary input $z \in \{0,1\}^*$. We consider a static corruption model, where $\mathcal{A}$ can corrupt any party $P_i \in P$ at the beginning of the protocol execution. Corruption allows $\mathcal{A}$ to take full control over P_i, learning all its internal state. Both the parties and the adversary $\mathcal{A}$ receive their inputs from a special entity, the environment $\mathcal{E}$, which models everything that happens external to the protocol execution. The environment not only provides inputs but also observes all outputs generated by the parties throughout the execution. We consider that our model operates within a hybrid setting where the protocol may access additional ideal functionalities, denoted by $\mathcal{H}_1, \ldots, \mathcal{H}_m$. In this case, we say that the protocol Π works in the $(\mathcal{H}_1, \ldots, \mathcal{H}_m)$-*hybrid* model.

Modeling Time and Communication. In our model, we assume a synchronous communication network where protocol execution unfolds in discrete rounds. This abstraction of rounds allows for clear reasoning about time during the protocol execution. The notion of rounds is formalized by the

global ideal functionality $\mathcal{F}_{clock}$ [26], which acts as a global clock that advances to the next round only when all honest parties are prepared to proceed, ensuring that every party is aware of the current round. We assume that communication between parties in $\mathcal{P}$ occurs over authenticated channels with a strict delivery guarantee of one round, formalized via an ideal functionality $\mathcal{F}_{GDC}$ [18]. This means that if a party P sends a message to party Q in round τ, Q receives it at the beginning of round $\tau+1$, with certainty that P is the sender. While the adversary $\mathcal{A}$ can observe message content and reorder messages sent within the same round, it cannot modify, delay, or drop messages, nor can it introduce new messages into the protocol. All other communications, such as those involving the adversary $\mathcal{A}$, the environment $\mathcal{E}$, are assumed to take zero rounds.

Modeling Global Ledger. We model the mechanics of UTXO-based cryptocurrencies, like Bitcoin, using a global ideal functionality $\mathbb{B}(\Delta,\Sigma,\mathcal{V})$ under the Global UC framework. This functionality is parameterized by a bounded delay Δ, indicating the maximum number of rounds required to confirm a valid transaction, a digital signature scheme Σ, and a set $\mathcal{V}$ that defines the valid spending conditions, such as signature verification with respect to Σ. We assume that the ledger interacts with a fixed set of parties $\mathcal{P}$. Initially, the ledger functionality $\mathbb{B}$, initiated per the instructions of environment $\mathcal{E}$, generates key pairs (pk_P, sk_P) for every party $P \in \mathcal{P}$, registers each public key pk_P to the ledger and establishes the initial state as a publicly accessible set of all posted transactions. Any party $P \in \mathcal{P}$ can post a transaction to $\mathbb{B}$, which, if deemed valid after verification, will be appended to the ledger after up to Δ rounds. Our ledger model is simplified for clarity. For a more detailed description and comprehensive formalization of the ledger model, we refer the reader to prior works [6, 11]. This simplified model suffices for our work and improves the readability of our channel protocol.

The GUC-security definition. We define a *hybrid* protocol Π that operates with access to ideal functionality $\mathcal{F}_{\text{prelim}}$ consisting of the global ledger $\mathbb{B}(\Delta,\Sigma,\mathcal{V})$, the global clock $\mathcal{F}_{clock}$, and $\mathcal{F}_{GDC}$. The output of an environment $\mathcal{E}$ interacting with Π and an adversary $\mathcal{A}$, given λ as the security parameter and z as the auxiliary input to $\mathcal{A}$, is denoted as $\text{EXEC}^{\mathcal{F}^{\text{prelim}}}_{\Pi,\mathcal{A},\mathcal{E}}(\lambda,z)$. Let $\phi_{\mathcal{F}}$ be the ideal protocol for an ideal functionality $\mathcal{F}$ with access to the $\mathcal{F}_{\text{prelim}}$. This means that the parties in $\mathcal{P}$ simply forward their inputs to the ideal functionality $\mathcal{F}$. The output of an environment $\mathcal{E}$ interacting with a protocol $\phi_{\mathcal{F}}$ and a simulator $\mathcal{S}$, given λ as the security parameter and z as the auxiliary input to $\mathcal{S}$, is denoted as $\text{EXEC}^{\mathcal{F}^{\text{prelim}}}_{\phi_{\mathcal{F}},\mathcal{S},\mathcal{E}}(\lambda,z)$. We say that if a protocol Π GUC-realizes an ideal functionality $\mathcal{F}$, then any attack that can be carried out against the real-world protocol Π can also be carried out against the ideal protocol $\phi_{\mathcal{F}}$ and vice versa. In other words, Π is at least as secure as $\mathcal{F}$.

Definition A.1. *A protocol Π GUC-realizes an ideal functionality $\mathcal{F}$, w.r.t. $\mathcal{F}_{prelim}$, if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for any environment $\mathcal{E}$, we have*

$$EXEC^{\mathcal{F}^{prelim}}_{\Pi,\mathcal{A},\mathcal{E}}(\lambda,z) \overset{c}{\approx} EXEC^{\mathcal{F}^{prelim}}_{\varphi^{\mathcal{F}},\mathcal{S},\mathcal{E}}(\lambda,z)$$

(where "$\overset{c}{\approx}$" denotes computational indistinguishability).

A.2 Ideal functionality

In this section, we will describe the ideal functionality $\mathcal{F}$ in detail. Following [6], we also represent payment channels through an ideal functionality $\mathcal{F}(T_p,k)$, which operates on top of the ledger functionality $\mathbb{B}(\Delta,\Sigma,\mathcal{V})$. This functionality, parameterized by two values T_p and k, models channel operations such as creation, update, and closure. Here, T_p represents an upper bound on the number of consecutive off-chain communication rounds between channel participants, while k defines the number of ways a channel state can be published. Note that we consider only protocols that realize $\mathcal{F}$ without producing an `ERROR` output, as any occurrence of `ERROR` implies a loss of security guarantees.

To simplify our notation and improve readability, we omit explicit calls to $\mathcal{F}_{clock}$ and $\mathcal{F}_{GDC}$. We use shorthand notation to represent message passing: $m \overset{\tau}{\hookrightarrow} P$ denotes sending message m to party P in round τ, and $m \overset{\tau}{\hookleftarrow} P$ denotes receiving it from P in round τ. A message m generally consists of (`MESSAGE-ID`, *parameters*). We also omit several natural checks that one would expect $\mathcal{F}$ to make. These checks could be formally handled by combining a functionality wrapper. For a formal definition of such wrappers, we refer the reader to [6].

We structure $\mathcal{F}$ into four parts: (i) Create, (ii) Pay, (iii) Close, and (iv) Punish. The formal description of $\mathcal{F}$ is shown in Figure 9.

Create. In the Create phase, both participants A and B must first send a $(\texttt{CREATE}, \gamma, tid_p)$ message to $\mathcal{F}$. An attribute tuple γ represents the channel, defined by its unique identifier γ.id, participants γ.users, total funds γ.cash, and the latest state of channel γ.st (latest funds distribution between participants). tid_p refers the party P's input for the funding transaction of the channel. Upon receiving these messages from both participants, $\mathcal{F}$ checks $\mathbb{B}$ for the presence of a funding transaction tx_F, which spend both input tid_A and tid_B with output amounts equal to γ.cash. If tx_F is confirmed within Δ rounds, $\mathcal{F}$ initializes ω, a structure that stores two latest valid states for each participant, denoted as θ_A, θ'_A for A and θ_B, θ'_B for B, as well as an additional state θ^*, representing the older of the two states held by the payee in the latest payment. The tuple ω ensures that sufficient history is available to resolve disputes if needed. Finally, $\mathcal{F}$ sends a `CREATED` message to both users, indicating successful channel creation, and updates the map Γ to link γ.id with (γ, tx_F, ω). Since the channel is created only after receiving `CREATED` message from both parties and the spending condition ϕ of tx_F is mutually agreed upon, *balance security in creation* holds.

Pay. In the Pay phase, where A is the payer and B is the payee, A initiates a payment by sending (PAY, id, $\overrightarrow{\theta}$, t_{stp}) to $\mathcal{F}$, where $\overrightarrow{\theta}$ represents the new balance state and t_{stp} is the number of rounds needed by the parties to set up off-chain applications. $\mathcal{F}$ then sends (REVOKE-REQ, id, $\overrightarrow{\theta}$, t_{stp}) to B, requesting B to revoke previous state. If $\mathcal{F}$ receives a (REVOKE, id) message from B, this indicates that B has agreed to the new state corresponding to A's payment. Otherwise, B rejects this payment, and the Pay process is aborted. Once both parties have agreed on the payment, $\mathcal{S}$ informs $\mathcal{F}$ with a vector of transaction identifiers $\overrightarrow{tid}$. $\mathcal{F}$ then sends a (PAY-REQ, id, $\overrightarrow{tid}$) message to A, requesting the finalization of the payment. If $\mathcal{F}$ receives a (PAY-OK, id) message from A, it updates the corresponding γ and ω in the map Γ, and sends a (PAID, id, $\overrightarrow{\theta}$) message to both users. Otherwise, if no PAY-OK is received, $\mathcal{F}$ executes the subprocedure ForceClose, which expects the funding transaction of the channel to be spent within Δ rounds. The new state $\overrightarrow{\theta}$ is considered valid and updated only after $\mathcal{F}$ receives both the REVOKE message from the payee and the PAY-OK message from the payer. Until this point, the previous state remains the sole valid state, thereby upholding *balance security in update*.

Close. Either participant can initiate the channel closure by sending a (CLOSE, id) message to $\mathcal{F}$. If the counterparty also sends the same message within T_p rounds, $\mathcal{F}$ expects a closing transaction tx_C, reflecting the latest channel state, to appear on $\mathbb{B}$ within Δ rounds. Upon observing tx_C, $\mathcal{F}$ updates $\Gamma(\text{id}) := \perp$ and notifies both users with (CLOSED, id). Otherwise, if one party is unresponsive or dishonest, $\mathcal{F}$ triggers the ForceClose subprocedure.

Punish. This phase, triggered at the end of each round τ_0, is designed to enforce accountability and resolve disputes using the stored states in ω. For each active channel, $\mathcal{F}$ checks if tx' appears on the ledger $\mathbb{B}$ that spends the corresponding funding transaction tx_F. If tx' represents the latest valid state γ.st that aligns with the expected balances for both participants, $\mathcal{F}$ finalizes the channel by updating $\Gamma(\text{id})$ to $\perp$ and sends both participants with a (CLOSED, id) message if not sent yet. This indicates either a peaceful cooperative channel closure by both parties or a forced closure by the payee in the latest payment. If tx' does not match the latest valid state but aligns with a permitted previous state in ω, $\mathcal{F}$ expects a follow-up transaction tx_{res} to appear on the ledger, making the participants' balances with what the latest state γ.st dictates. Upon seeing this resolving transaction, $\mathcal{F}$ updates $\Gamma(\text{id})$ to $\perp$, then sends (RESOLVED, id) and (CLOSED, id) to both users. This case indicates a forced closure of the channel by the payer in the latest payment. However, if tx' reflects a revoked state, $\mathcal{F}$ expects a punish transaction tx_{pnsh} to appear on the ledger, allowing the honest party to claim the entire channel funds. In every scenario, for an honest and rational participant, the channel will never finalize in a state that allocates less than the balance specified in the latest globally valid state γ.st. Thus, *balance security in finalization* satisfies.

A.3 Protocol

In this section, we formally present the protocol Π as outlined in Section 5.2. The protocol builds on the high-level concepts discussed earlier, now incorporating detailed UC formalism, allowing for clear interactions within a global environment $\mathcal{E}$ and time-bound communication rounds. To enhance readability and distinguish between the communication between parties and input/outputs from/to the environment $\mathcal{E}$, we denote messages involved $\mathcal{E}$ in uppercase, e.g., "CREATE," while messages between parties use lowercase, e.g., "createInfo." Similar to the ideal functionality, the pseudocode presented excludes several checks that an honest user would naturally perform, which can instead be handled through a protocol wrapper. The protocol is structured into four parts, each carefully designed to handle the various stages of payment channel. Additionally, we incorporate two subprocedures to streamline the protocol: one for the force closure mechanism, and another for generating necessary transactions at each payment.

ULTRAVIOLET protocol Π

Create

Party A upon (CREATE, γ, tid_A) $\xleftarrow{\tau_0}$ $\mathcal{E}$:

1. Set id = γ.id. Generate (pk_A, sk_A), $(rh_{A_{-1}}, rs_{A_{-1}})$, (rh_{A_0}, rs_{A_0}). Construct $auth^A_{set} := \{pk_A, rh_{A_{-1}}, rh_{A_0}\}$, a set containing the public keys and revocation secret hash owned by A, and $rs^A_{set} := \{rs_{A_{-1}}, rs_{A_0}\}$, comprising revocation secrets generated by A.
2. Extract initial balances v^A and v^B from $\gamma.st$ and set $v := v^A + v^B$, $v^A_{A_{-1}} := v^A, v^B_{A_{-1}} := v^B, v^A_{A_0} := v^A, v^B_{A_0} := v^B$.
3. Send (createInfo, id, $tid_A, auth^A_{set}$) $\xhookrightarrow{\tau_0}$ B.
4. If (createInfo, id, $tid_B, auth^B_{set}$) $\xhookleftarrow{\tau_0+1}$ B, $auth^A_{set} := auth^A_{set} \cup auth^B_{set}$, continue. Else, go idle.
5. Generate the funding transaction $tx_F := tx([tid_A, tid_B], \langle sk_A \wedge sk_B, v\rangle)$.
6. Generate $tx_{A_{-1}} := tx(tx_F, [\langle sk_A, v^A_{A_{-1}}\rangle, \langle (rs_{B_{-1}} \wedge sk_A) \vee (sk_A \wedge sk_B) \vee (\mathsf{RelTime}(T) \wedge sk_B), v^B_{A_{-1}}\rangle])$.
7. Generate $tx_{A_0} := tx(tx_F, [\langle sk_A, v^A_{A_0}\rangle, \langle (rs_{B_0} \wedge sk_A) \vee (sk_A \wedge sk_B) \vee (\mathsf{RelTime}(T) \wedge sk_B), v^B_{A_0}\rangle])$.
8. Construct $tx^A_{set} := \{tx_{A_{-1}}, tx_{A_0}\}$, a set containing the transactions owned by A.
9. A uses sk_A to generate signatures σ_{tid_A}, $\sigma^A_{A_{-1}}$ and $\sigma^A_{A_0}$ for transactions tx_F, $tx_{A_{-1}}$ and tx_{A_0} respectively. Construct $sig^A_{set} := \{\sigma^A_{A_{-1}}, \sigma^A_{A_0}\}$.
10. Send (prepareInfo, id, tx^A_{set}, sig^A_{set}) $\xhookrightarrow{\tau_0+1+\tau_{pre}}$ B.

11. If (prepareInfo, id, tx^B_{set}, sig^B_{set}) $\xleftarrow{\tau_0+2+\tau_{pre}}$ B, set tx^A_{set}:= tx^A_{set} $\cup$ tx^B_{set}, sig^A_{set} := $sig^A_{set} \cup sig^B_{set}$. Else, go idle.
12. Send (createFund, id, σ_{tid_A}) $\xhookrightarrow{\tau_0+2+\tau_{pre}}$ B.
13. If (createFund, id, σ_{tid_B}) $\xleftarrow{\tau_0+3+\tau_{pre}}$ B, post (tx_F, $\{\sigma_{tid_A}, \sigma_{tid_B}\}$) on $\mathbb{B}$.
14. If tx_F is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + 3 + \tau_{pre} + \Delta$, set $\Gamma^A(\text{id}) := (tx_F, auth^A_{set}, rs^A_{set}, tx^A_{set}, sig^A_{set})$, (CREATED, id) $\xhookrightarrow{\tau_1}$ $\mathcal{E}$.

Pay

Party A's i-th payment, assuming party B has completed j payments.

Party A upon (PAY, id, $\overrightarrow{\theta}$, t_{stp}) $\xhookleftarrow{\tau_0}$ $\mathcal{E}$:

1. Extract $(tx_F, auth^A_{set}, rs^A_{set}, tx^A_{set}, sig^A_{set}) = \Gamma^A(\text{id})$.
2. Send (revokeReq, id, $\overrightarrow{\theta}$, t_{stp}) $\xhookrightarrow{\tau_0}$ B.

Party B upon (revokeReq, id, $\overrightarrow{\theta}$, t_{stp}) $\xhookleftarrow{t_0}$ A:

1. Send (REVOKE-REQ, id, $\overrightarrow{\theta}$, t_{stp}) $\xhookrightarrow{t_0}$ $\mathcal{E}$
2. If not (REVOKE, id) $\xhookleftarrow{t_0}$ $\mathcal{E}$, go idle.
3. Extract $(tx_F, auth^B_{set}, rs^B_{set}, tx^B_{set}, sig^B_{set}) = \Gamma^B(\text{id})$.
4. Generate (rh_{B_i}, rs_{B_i}), let $auth^B_{set} := auth^B_{set} \cup \{rh_{B_i}\}$, rs^B_{set} := $rs^B_{set} \cup \{rs_{B_i}\}$.
5. Retrieve $rs_{B_{i-2}}$ from rs^B_{set}, (revokeInfo, id, rh_{B_i}, $rs_{B_{i-2}}$) $\xhookrightarrow{t_0+t_g}$ A.

Party A upon (revokeInfo, id, rh_{B_i}, $rs_{B_{i-2}}$) $\xleftarrow{\tau_0+2+t_g}$ B:

1. Let $auth^A_{set} := auth^A_{set} \cup \{rh_{B_i}\}$. Retrieve $tx_{A_{i-2}}$ from tx^A_{set} and extract $v^A_{A_{i-2}}$ and $v^B_{A_{i-2}}$ from it.
2. Generate the punishment transaction $tx^{A_{i-2}}_{pnsh} := tx(tx_{A_{i-2}}, \langle sk_A, v^B_{A_{i-2}} \rangle)$ corresponding to $tx_{A_{i-2}}$.
3. A uses sk_A and $rs_{B_{i-2}}$ to generate signature $\sigma^{A_{i-2}}_{pnsh}$ for punishment transaction $tx^{A_{i-2}}_{pnsh}$.
4. Let $\mathcal{T}^A_{revoke} := \mathcal{T}^A_{revoke} \cup \{tx_{A_{i-2}}\}$, $\mathcal{T}^A_{pnsh} := \mathcal{T}^A_{pnsh} \cup \{tx^{A_{i-2}}_{pnsh}\}$, $\Sigma^A_{pnsh} := \Sigma^A_{pnsh} \cup \{\sigma^{A_{i-2}}_{pnsh}\}$.
5. Let $\Theta^A(\text{id}) := (\mathcal{T}^A_{revoke}, \mathcal{T}^A_{pnsh}, \Sigma^A_{pnsh})$.
6. Extract $v^A_{A_i}$ and $v^B_{A_i}$, representing the balance states of A and B after this payment, from $\overrightarrow{\theta}$.
7. Let $tx^{A_i}_{pay} \leftarrow$ GenerateTxs(tx_F, tx^A_{set}, $auth^A_{set}$, $v^A_{A_i}$, $v^B_{A_i}$), tx^A_{set} := $tx^A_{set} \cup tx^{A_i}_{pay}$.
8. A uses sk_A to sign each transaction in $tx^{A_i}_{pay}$, generating a set of signatures $\sigma^{A_i}_{pay}$.
9. Let $\overrightarrow{tid}$ be the tuple of ids corresponding to each transaction in $tx^{A_i}_{pay}$, (PAY-REQ, id, $\overrightarrow{tid}$) $\xhookrightarrow{\tau_1 \leq \tau_0+2+t_g+t_{\text{stp}}}$ $\mathcal{E}$.
10. Send (payInfo, id, $tx^{A_i}_{pay}$, $\sigma^{A_i}_{pay}$) $\xhookrightarrow{\tau_1 \leq \tau_0+2+t_g+t_{\text{stp}}}$ B.

Party B in round $t_1 \leq t_0 + t_g + 2 + t_{\text{stp}}$:

1. If (payInfo, id, $tx^{A_i}_{pay}$, $\sigma^{A_i}_{pay}$) $\xhookleftarrow{t_1}$ A, go to next step. Else, execute ForceClose(id).
2. Let $tx^B_{set} := tx^B_{set} \cup tx^{A_i}_{pay}$, $sig^B_{set} := sig^B_{set} \cup \sigma^{A_i}_{pay}$.
3. Let $\Gamma^B(\text{id}) := (tx_F, auth^B_{set}, rs^B_{set}, tx^B_{set}, sig^B_{set})$.
4. Send (payCom, id) $\xhookrightarrow{t_1}$ A and (PAID, id, $\overrightarrow{\theta}$) $\xhookrightarrow{t_1}$ $\mathcal{E}$.

Party A in round $\tau_1 + 2$:

1. If (payCom, id) $\xleftarrow{\tau_1+2}$ B, continue. Else, execute ForceClose(id).
2. If not (PAY-OK, id) $\xleftarrow{\tau_1+2}$ $\mathcal{E}$, go idle.
3. Let $\Gamma^A(\text{id}) := (tx_F, auth^A_{set}, rs^A_{set}, tx^A_{set}, sig^A_{set})$.
4. (PAID, id, $\overrightarrow{\theta}$) $\xhookrightarrow{\tau_1+2}$ $\mathcal{E}$.

Close

Let A's n-th payment be the latest payment between A and B.

Party A upon (CLOSE, id) $\xhookleftarrow{\tau_0}$ $\mathcal{E}$:

1. Extract $(tx_F, auth^A_{set}, rs^A_{set}, tx^A_{set}, sig^A_{set}) = \Gamma^A(\text{id})$.
2. Retrieve tx_{A_n} from tx^A_{set} and extract $v^A_{A_n}$ and $v^B_{A_n}$ from tx_{A_n}.
3. Generate transaction $tx_C := tx(tx_F, [\langle sk_A, v^A_{A_n} \rangle, \langle sk_B, v^B_{A_n} \rangle])$.
4. A uses sk_A to generate signature σ^A_C for transaction tx_C.
5. Send (closeInfo, id, tx_C, σ^A_C) $\xhookrightarrow{\tau_0+\tau_g}$ B.
6. If (closeInfo, id, tx_C, σ^B_C) $\xleftarrow{\tau_0+\tau_g+1}$ B, post (tx_C, $\{\sigma^A_C, \sigma^B_C\}$) on $\mathbb{B}$. Else, go idle.
7. If tx_C is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + \tau_g + 1 + \Delta$, set $\Theta^A(\text{id}) := \perp$, $\Gamma^A(\text{id}) := \perp$ and (CLOSED, id) $\xhookrightarrow{\tau_1}$ $\mathcal{E}$. Else, execute ForceClose(id).

Punish

Party A upon PUNISH $\xhookleftarrow{\tau_0}$ $\mathcal{E}$:

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

1. Extract $(\mathcal{T}^A_{revoke}, \mathcal{T}^A_{pnsh}, \Sigma^A_{pnsh}) = \Theta^A(\text{id})$.
2. If any revoked payment $tx_{A_n} \in \mathcal{T}^A_{revoke}$ appears on $\mathbb{B}$, retrieve $tx^{A_n}_{pnsh}$, $\sigma^{A_n}_{pnsh}$ from $\mathcal{T}^A_{pnsh}$ and Σ^A_{pnsh} respectively. Post ($tx^{A_n}_{pnsh}$, $\sigma^{A_n}_{pnsh}$) on $\mathbb{B}$.
3. After $tx^{A_n}_{pnsh}$ is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + \Delta$, set $\Theta^A(\text{id}) := \perp$, $\Gamma^A(\text{id}) := \perp$ and (PUNISHED, id) $\xhookrightarrow{\tau_1}$ $\mathcal{E}$.

Subprotocols

`ForceClose`(id):

Let τ_0 be the current round, assuming that party P has completed i payments and party Q has completed j payments.

Party P do the following:

1. Extract $(tx_F, auth^P_{set}, rs^P_{set}, tx^P_{set}, sig^P_{set}) = \Gamma^P(\text{id})$.
2. Retrieve tx_{Q_j}, $\sigma^Q_{Q_j}$ from tx^P_{set} and sig^P_{set} respectively.
3. P uses sk_P to generate signature $\sigma^P_{Q_j}$ for transaction tx_{Q_j}. Post $(tx_{Q_j}, \{(\sigma^P_{Q_j}, \sigma^Q_{Q_j})\})$ on $\mathbb{B}$.
4. After tx_{Q_j} is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + \Delta$, set $\Theta^P(\text{id}) := \bot$, $\Gamma^P(\text{id}) := \bot$. (forceClose, id, tx_{Q_j}) $\overset{\tau_1}{\hookrightarrow} Q$ and (`CLOSED`, id) $\overset{\tau_1}{\hookrightarrow} \mathcal{E}$.

Party Q upon (`forceClose`, id, tx_{Q_j}) $\overset{t_0}{\hookleftarrow} P$:

1. After tx_{Q_j} appears on $\mathbb{B}$, extract (tx_F, $auth^Q_{set}$, rs^Q_{set}, tx^Q_{set}, sig^Q_{set}) $= \Gamma^Q(\text{id})$.
2. If there exists $tx_{P_i}^{\curvearrowright tx_{Q_j}} \in tx^Q_{set}$, continue. Else, (`CLOSED`, id) $\overset{t_0}{\hookrightarrow} \mathcal{E}$ and stop.
3. Retrieve $tx_{P_i}^{\curvearrowright tx_{Q_j}}$, $\sigma^P_{P_i(Q_j)}$ from tx^Q_{set} and sig^Q_{set} respectively.
4. Q uses sk_Q to generate signature $\sigma^Q_{P_i(Q_j)}$ for transaction $tx_{P_i}^{\curvearrowright tx_{Q_j}}$. Post $(tx_{P_i}^{\curvearrowright tx_{Q_j}}, \{\sigma^P_{P_i(Q_j)}, \sigma^Q_{P_i(Q_j)}\})$ on $\mathbb{B}$.
5. After $tx_{P_i}^{\curvearrowright tx_{Q_j}}$ is confirmed on $\mathbb{B}$ in round $t_0 + \Delta$, set $\Theta^Q(\text{id}) := \bot$, $\Gamma^Q(\text{id}) := \bot$.
6. (`RESOLVED`, id) $\overset{t_0+\Delta}{\hookrightarrow} \mathcal{E}$ and (`CLOSED`, id) $\overset{t_0+\Delta}{\hookrightarrow} \mathcal{E}$.

`GenerateTxs`(tx_F, tx^P_{set}, $auth^P_{set}$, $v^P_{P_i}$, $v^Q_{P_i}$):

Party P's i-th payment, assuming the other party Q has completed j payments.

1. Retrieve $\{pk_P, pk_Q, rh_{Q_i}\}$ from $auth^P_{set}$.
2. Generate the payer P's i-th payment transaction tx_{P_i} := $\text{tx}(tx_F, [\langle sk_P, v^P_{P_i}\rangle, \langle (rs_{Q_i} \wedge sk_P) \vee (sk_P \wedge sk_Q) \vee (\mathsf{RelTime}(T) \wedge sk_Q), v^Q_{P_i}\rangle])$.
3. Retrieve $\{tx_{Q_j}, tx_{Q_{j-1}}\}$ from tx^P_{set}, extract $v^P_{Q_j}$ and $v^P_{Q_{j-1}}$ respectively.
4. Let $v^P_{res_1} := v^P_{P_i}$, $v^Q_{res_1} := v^P_{Q_j} - v^P_{P_i}$.
5. Generate resolve transaction $tx_{P_i}^{\curvearrowright tx_{Q_j}} := tx(tx_{Q_j}, [\langle sk_P, v^P_{res_1}\rangle, \langle sk_Q, v^Q_{res_1}\rangle])$.
6. If $v^P_{P_i} \leq v^P_{Q_{j-1}}$, set $v^P_{res_2} := v^P_{P_i}$, $v^Q_{res_2} := v^P_{Q_{j-1}} - v^P_{P_i}$. Else, set $v^P_{res_2} := v^P_{Q_{j-1}}$, $v^Q_{res_2} := 0$.
7. Generate resolve transaction $tx_{P_i}^{\curvearrowright tx_{Q_{j-1}}} := tx(tx_{Q_{j-1}}, [\langle sk_P, v^P_{res_2}\rangle, \langle sk_Q, v^Q_{res_2}\rangle])$.
8. Return $tx^{A_i}_{pay} := \{tx_{P_i}, tx_{P_i}^{\curvearrowright tx_{Q_j}}, tx_{P_i}^{\curvearrowright tx_{Q_{j-1}}}\}$.

A.4 Proof

In this section, we present the simulator and the formal proof that the ULTRAVIOLET protocol Π, described in Appendix A.3, GUC-realizes the ideal functionality $\mathcal{F}$, defined in Appendix A.2.

Simulator for Create

Case A is honest and B is corrupted

Upon A sending (`CREATE`, γ, tid_A) $\overset{\tau_0}{\hookrightarrow} \mathcal{F}$, if B does not send (`CREATE`, id, γ, tid_B) $\overset{\tau}{\hookrightarrow} \mathcal{F}$ where $|\tau_0 - \tau| \leq T_P$, then distinguish the following case:

(1) If B sends(createInfo, id, $tid_B, auth^B_{set}$) $\overset{\tau_0}{\hookrightarrow} A$, then send (`CREATE`, γ, tid_A) $\overset{\tau_0}{\hookrightarrow} \mathcal{F}$ on behalf of B.

(2) Otherwise stop.

Do the following:

1. Set id = γ.id. Generate (pk_A, sk_A), $(rh_{A_{-1}}, rs_{A_{-1}})$, (rh_{A_0}, rs_{A_0}). Construct $auth^A_{set} := \{pk_A, rh_{A_{-1}}, rh_{A_0}\}$, a set containing the public keys and revocation secret hash owned by A, and $rs^A_{set} := \{rs_{A_{-1}}, rs_{A_0}\}$, comprising revocation secrets generated by A.
2. Extract initial balances v^A and v^B from $\gamma.st$ and set $v := v^A + v^B$, $v^A_{A_{-1}} := v^A, v^B_{A_{-1}} := v^B, v^A_{A_0} := v^A, v^B_{A_0} := v^B$.
3. Send (createInfo, id, $tid_A, auth^A_{set}$) $\overset{\tau_0}{\hookrightarrow} B$.
4. If (createInfo, id, $tid_B, auth^B_{set}$) $\overset{\tau_0+1}{\hookleftarrow} B$, $auth^A_{set} := auth^A_{set} \cup auth^B_{set}$, do the following. Else, go idle.
5. Generate the funding transaction $tx_F := tx([tid_A, tid_B], \langle sk_A \wedge sk_B, v\rangle)$.
6. Generate $tx_{A_{-1}} := tx(tx_F, [\langle sk_A, v^A_{A_{-1}}\rangle, \langle (rs_{B_{-1}} \wedge sk_A) \vee (sk_A \wedge sk_B) \vee (\mathsf{RelTime}(T) \wedge sk_B), v^B_{A_{-1}}\rangle])$.
7. Generate $tx_{A_0} := tx(tx_F, [\langle sk_A, v^A_{A_0}\rangle, \langle (rs_{B_0} \wedge sk_A) \vee (sk_A \wedge sk_B) \vee (\mathsf{RelTime}(T) \wedge sk_B), v^B_{A_0}\rangle])$.
8. Construct $tx^A_{set} := \{tx_{A_{-1}}, tx_{A_0}\}$, a set containing the transactions owned by A.
9. Generate signatures σ_{tid_A}, $\sigma^A_{A_{-1}}$ and $\sigma^A_{A_0}$ on behalf of A, for transactions tx_F, $tx_{A_{-1}}$ and tx_{A_0} respectively. Construct $sig^A_{set} := \{\sigma^A_{A_{-1}}, \sigma^A_{A_0}\}$.
10. Send (prepareInfo, id, tx^A_{set}, sig^A_{set}) $\overset{\tau_0+1+\tau_{pre}}{\hookrightarrow} B$.
11. If (prepareInfo, id, tx^B_{set}, sig^B_{set}) $\overset{\tau_0+2+\tau_{pre}}{\hookleftarrow} B$, $tx^A_{set} := tx^A_{set} \cup tx^B_{set}$, $sig^A_{set} := sig^A_{set} \cup sig^B_{set}$. Else, go idle.
12. Send (createFund, id, σ_{tid_A}) $\overset{\tau_0+2+\tau_{pre}}{\hookrightarrow} B$.
13. If (createFund, id, σ_{tid_B}) $\overset{\tau_0+3+\tau_{pre}}{\hookleftarrow} B$, post $(tx_F, \{\sigma_{tid_A}, \sigma_{tid_B}\})$ on $\mathbb{B}$.

14. If tx_F is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + 3 + \tau_{pre} + \Delta$, set $\Gamma^A(\text{id}) := (tx_F, auth^A_{set}, rs^A_{set}, tx^A_{set}, sig^A_{set})$. $\mathcal{S}$ instruct $\mathcal{F}$ to (CREATED, id) $\overset{\tau_1}{\hookrightarrow} \mathcal{E}$.

Simulator for Pay

Party A's i-th payment, assuming party B has completed j payments.

Case A is honest and B is corrupted

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

1. Extract $(tx_F, auth^A_{set}, rs^A_{set}, tx^A_{set}, sig^A_{set}) = \Gamma^A(\text{id})$.
2. Send (revokeReq, id, $\overrightarrow{\theta}, t_{\text{stp}}$) $\overset{\tau_0}{\hookrightarrow} B$.
3. Upon (revokeInfo, id, $pk_{B_i}, sk_{B_{i-2}}$) $\overset{\tau_0+2+t_g}{\hookleftarrow} B$, do the following. Else, go idle.
4. Let $auth^A_{set} := auth^A_{set} \cup \{pk_{B_i}\}$. Retrieve $tx_{A_{i-2}}$ from tx^A_{set} and extract $v^A_{A_{i-2}}$ and $v^B_{A_{i-2}}$ from it.
5. Generate the punishment transaction $tx^{A_{i-2}}_{pnsh} := tx(tx_{A_{i-2}}, \langle sk_A, v^B_{A_{i-2}} \rangle)$ corresponding to $tx_{A_{i-2}}$.
6. Generate signature $\sigma^{A_{i-2}}_{pnsh}$ for punishment transaction $tx^{A_{i-2}}_{pnsh}$ on behalf of A.
7. Let $\mathcal{T}^A_{revoke} := \mathcal{T}^A_{revoke} \cup \{tx_{A_{i-2}}\}$, $\mathcal{T}^A_{pnsh} := \mathcal{T}^A_{pnsh} \cup \{tx^{A_{i-2}}_{pnsh}\}$, $\Sigma^A_{pnsh} := \Sigma^A_{pnsh} \cup \{\sigma^{A_{i-2}}_{pnsh}\}$.
8. Let $\Theta^A(\text{id}) := (\mathcal{T}^A_{revoke}, \mathcal{T}^A_{pnsh}, \Sigma^A_{pnsh})$.
9. Extract $v^A_{A_i}$ and $v^B_{A_i}$, representing the balance states of A and B after this payment, from $\overrightarrow{\theta}$.
10. Let $tx^{A_i}_{pay} \leftarrow$ GenerateTxs($tx_F, tx^A_{set}, auth^A_{set}, v^A_{A_i}, v^B_{A_i}$), $tx^A_{set} := tx^A_{set} \cup tx^{A_i}_{pay}$.
11. Sign each transaction in $tx^{A_i}_{pay}$ on behalf of A, generating a set of signatures $\sigma^{A_i}_{pay}$.
12. Let $\overrightarrow{tid}$ be the tuple of ids corresponding to each transaction in $tx^{A_i}_{pay}$. Instruct $\mathcal{F}$ of $\overrightarrow{tid}$ and (PAY-REQ, id, $\overrightarrow{tid}$) $\overset{\tau_1 \leq \tau_0+2+t_g+t_{\text{stp}}}{\hookleftarrow} \mathcal{E}$ via A.
13. Send (payInfo, id, $tx^{A_i}_{pay}, \sigma^{A_i}_{pay}$) $\overset{\tau_1 \leq \tau_0+2+t_g+t_{\text{stp}}}{\hookrightarrow} B$.
14. If (payCom, id) $\overset{\tau_1+2}{\hookleftarrow} B$, continue. Else, execute ForceClose(id).
15. If A does not send (PAY-OK, id) $\overset{\tau_1+2}{\hookrightarrow} \mathcal{F}$, go idle.
16. Let $\Gamma^A(\text{id}) := (tx_F, auth^A_{set}, rs^A_{set}, tx^A_{set}, sig^A_{set})$.
17. Instruct $\mathcal{F}$ to (PAID, id, $\overrightarrow{\theta}$) $\overset{\tau_2+2}{\hookrightarrow} \mathcal{E}$.

Case B is honest and A is corrupted

Upon A sending (revokeReq, id, $\overrightarrow{\theta}, t_{\text{stp}}$) $\overset{\tau_0}{\hookrightarrow} B$, send (PAY, id, $\overrightarrow{\theta}, t_{\text{stp}}$) $\overset{\tau_0}{\hookrightarrow} \mathcal{F}$ on behalf of A, if A has not sent this message. Proceed as follows:

1. Upon (revokeReq, id, $\overrightarrow{\theta}, t_{\text{stp}}$) $\overset{t_0}{\hookleftarrow} A$, instruct $\mathcal{F}$ to (REVOKE-REQ, id, $\overrightarrow{\theta}, t_{\text{stp}}$) $\overset{t_0}{\hookrightarrow} \mathcal{E}$.
2. If B does not send (REVOKE, id) $\overset{t_0}{\hookrightarrow} \mathcal{F}$, go idle.
3. Extract $(tx_F, auth^B_{set}, rs^B_{set}, tx^B_{set}, sig^B_{set}) = \Gamma^B(\text{id})$.
4. Generate (rh_{B_i}, rs_{B_i}), let $auth^B_{set} := auth^B_{set} \cup \{pk_{B_i}\}$, $rs^B_{set} := rs^B_{set} \cup \{sk_{B_i}\}$.
5. Retrieve $rs_{B_{i-2}}$ from rs^B_{set}, send (revokeInfo, id, $rh_{B_i}, rs_{B_{i-2}}$) $\overset{t_0+t_g}{\hookrightarrow} A$.
6. If (payInfo, id, $tx^{A_i}_{pay}, \sigma^{A_i}_{pay}$) $\overset{t_1}{\hookleftarrow} A$ in round $t_1 \leq t_0 + t_g + 2 + t_{\text{stp}}$, continue. Else, execute ForceClose(id).
7. Let $tx^B_{set} := tx^B_{set} \cup tx^{A_i}_{pay}, sig^B_{set} := sig^B_{set} \cup \sigma^{A_i}_{pay}$.
8. Let $\Gamma^B(\text{id}) := (tx_F, auth^B_{set}, rs^B_{set}, tx^B_{set}, sig^B_{set})$.
9. Send (payCom, id) $\overset{t_1}{\hookrightarrow} A$ and instruct $\mathcal{F}$ to (PAID, id, $\overrightarrow{\theta}$) $\overset{t_1}{\hookrightarrow} \mathcal{E}$.

Simulator for Close

Case A is honest and B is corrupted

Let A's n-th payment be the latest payment between A and B.

Upon A sending (CLOSE, id) $\overset{\tau_0}{\hookrightarrow} \mathcal{F}$, do the following:

1. Extract $(tx_F, auth^A_{set}, rs^A_{set}, tx^A_{set}, sig^A_{set}) = \Gamma^A(\text{id})$.
2. Retrieve tx_{A_n} from tx^A_{set} and extract $v^A_{A_n}$ and $v^B_{A_n}$ from tx_{A_n}.
3. Generate transaction $tx_C := tx(tx_F, [\langle sk_A, v^A_{A_n} \rangle, \langle sk_B, v^B_{A_n} \rangle])$.
4. Generate signature σ^A_C for transaction tx_C on behalf of A.
5. Send (closeInfo, id, tx_C, σ^A_C) $\overset{\tau_0+\tau_g}{\hookrightarrow} B$.
6. If (closeInfo, id, tx_C, σ^B_C) $\overset{\tau_0+\tau_g+1}{\hookleftarrow} B$, post ($tx_C, \{\sigma^A_C, \sigma^B_C\}$) on $\mathbb{B}$. Else, go idle.
7. If tx_C is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + \tau_g + 1 + \Delta$, set $\Theta^A(\text{id}) := \perp$, $\Gamma^A(\text{id}) := \perp$ and instruct $\mathcal{F}$ to (CLOSED, id) $\overset{\tau_1}{\hookrightarrow} \mathcal{E}$ Else, execute ForceClose(id).

Simulator for Punish

Case A is honest and B is corrupted

Upon A sending PUNISH $\overset{\tau_0}{\hookrightarrow} \mathcal{F}$, for each id $\in \{0,1\}^*$ s.t. $\Theta^A(\text{id}) \neq \perp$, do the following:

1. Extract $(\mathcal{T}^A_{revoke}, \mathcal{T}^A_{pnsh}, \Sigma^A_{pnsh}) = \Theta^A(\text{id})$.
2. If any revoked payment $tx_{A_n} \in \mathcal{T}^A_{revoke}$ is on $\mathbb{B}$, retrieve $tx^{A_n}_{pnsh}$, $\sigma^{A_n}_{pnsh}$ from $\mathcal{T}^A_{pnsh}$ and Σ^A_{pnsh} respectively. Post ($tx^{A_n}_{pnsh}, \sigma^{A_n}_{pnsh}$) on $\mathbb{B}$.

3. After $tx^{A_n}_{pnsh}$ is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + \Delta$, set $\Theta^A(\text{id}) := \bot$, $\Gamma^A(\text{id}) := \bot$ and instruct $\mathcal{F}$ to (PUNISHED, id) $\overset{\tau_1}{\hookrightarrow} \mathcal{E}$.

Simulator for ForceClose(id)

Let τ_0 be the current round, assuming that party P has completed i payments and party Q has completed j payments.

Case P is honest and Q is corrupted

1. Extract $(tx_F, auth^P_{set}, rs^P_{set}, tx^P_{set}, sig^P_{set}) = \Gamma^P(\text{id})$.
2. Retrieve tx_{Q_j}, $\sigma^Q_{Q_j}$ from tx^P_{set} and sig^P_{set} respectively.
3. Generate signature $\sigma^P_{Q_j}$ for transaction tx_{Q_j} on behalf of P. Post $(tx_{Q_j}, \{(\sigma^P_{Q_j}, \sigma^Q_{Q_j})\})$ on $\mathbb{B}$.
4. After tx_{Q_j} is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + \Delta$, set $\Theta^P(\text{id}) := \bot$, $\Gamma^P(\text{id}) := \bot$. Send (forceClose, id, tx_{Q_j}) $\overset{\tau_1}{\hookrightarrow}$ Q and instruct $\mathcal{F}$ to (CLOSED, id) $\overset{\tau_1}{\hookrightarrow} \mathcal{E}$.

Case Q is honest and P is corrupted

Upon P sending (forceClose, id, tx_{Q_j}) $\overset{\tau_0}{\hookrightarrow} Q$, proceed as follows:

1. Upon (forceClose, id, tx_{Q_j}) $\overset{t_0}{\hookleftarrow} P$, if tx_{Q_j} appears on $\mathbb{B}$, extract $(tx_F, auth^Q_{set}, rs^Q_{set}, tx^Q_{set}, sig^Q_{set}) = \Gamma^Q(\text{id})$.
2. If there exists $tx_{P_i}^{\curvearrowright tx_{Q_j}} \in tx^Q_{set}$, continue. Else, instruct $\mathcal{F}$ to (CLOSED, id) $\overset{t_0}{\hookrightarrow} \mathcal{E}$ and stop.
3. Retrieve $tx_{P_i}^{\curvearrowright tx_{Q_j}}$, $\sigma^P_{P_i(Q_j)}$ from tx^Q_{set} and sig^Q_{set} respectively.
4. Generate signature $\sigma^Q_{P_i(Q_j)}$ for transaction $tx_{P_i}^{\curvearrowright tx_{Q_j}}$ on behalf of Q. Post $(tx_{P_i}^{\curvearrowright tx_{Q_j}}, \{\sigma^P_{P_i(Q_j)}, \sigma^Q_{P_i(Q_j)}\})$ on $\mathbb{B}$.
5. After $tx_{P_i}^{\curvearrowright tx_{Q_j}}$ is confirmed on $\mathbb{B}$ in round $t_0 + \Delta$, set $\Theta^Q(\text{id}) := \bot$, $\Gamma^Q(\text{id}) := \bot$.
6. (RESOLVED, id) $\overset{t_0+\Delta}{\longrightarrow} \mathcal{E}$ and (CLOSED, id) $\overset{t_0+\Delta}{\longrightarrow} \mathcal{E}$.

To proof that the protocol Π GUC-realizes the ideal functionality $\mathcal{F}$, we demonstrate that, from the perspective of the environment $\mathcal{E}$, the transcript arising from interactions between the simulator $\mathcal{S}$ and the ideal functionality $\mathcal{F}$ is indistinguishable from the transcript produced during the execution of the protocol Π in the presence of an adversary $\mathcal{A}$. Formally, we aim to prove that the execution ensembles $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{E}}$ and $\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are indistinguishable for any environment $\mathcal{E}$. We demonstrate this indistinguishability across the various phases of the protocol, including Create, Pay, Close, and Punish, along with the subprotocol ForceClose. Note that we require an EUF-CMA secure signature scheme Σ to prevent forgery and a collision-resistant hash scheme to ensure the security of the revocation mechanism.

In our description, we use $m[\tau]$ to indicate that a message m is observed by the environment $\mathcal{E}$ in round τ. Each message is uniquely identified by its message identifier (e.g., CREATE or createInfo), while specific parameters are omitted for clarity. Furthermore, the protocol interacts with other ideal functionalities, which may in turn interact with the environment $\mathcal{E}$ or other parties, including those under adversarial control. These interactions may also affect publicly observable variables such as the ledger $\mathbb{B}$. To formalize this, we define obsSet(a_n, τ) as the set of all observable side effects resulting from action a_n executed in round τ.

In the following analysis, we consider different corruption cases, examining the view of environment $\mathcal{E}$ in the real world protocol Π and the view of $\mathcal{E}$ in the ideal world as simulated by $\mathcal{S}$. Notably, in the real world, the environment $\mathcal{E}$ controls the adversary $\mathcal{A}$, and thus equivalently controls all corrupted parties. Therefore, we do not analyze cases where both parties are corrupted, as such scenarios reduce to the environment communicating with itself, which is trivially identical in the ideal and the real world. Similarly, messages from corrupted parties to $\mathcal{E}$ are not considered for the same reason. Moreover, we omit the case where both parties are honest, as the simulator merely needs to follow the protocol execution faithfully, guaranteeing indistinguishability between the real and ideal world.

Lemma 1. *The Create phase of protocol* Π *GUC-realizes the Create phase of functionality* $\mathcal{F}$.

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

Real World: After receiving CREATE in round τ_0, A sends createInfo to B. If A receives createInfo from B in round $\tau_0 + 1$, A will perform action a_0 := "generate and sign transactions" for channel creation in round $\tau_0 + 1$. If this is successful, A will send prepareInfo to B in round $\tau_0 + 1 + \tau_{pre}$. If A receives prepareInfo in round $\tau_0 + 2 + \tau_{pre}$, A will send the signature for the funding transaction tx_F via createFund to B. If A receives createFund from B in round $\tau_0 + 3 + \tau_{pre}$, it will perform action a_1 := "post tx_F on $\mathbb{B}$". If tx_F is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + 3 + \tau_{pre} + \Delta$, finally A will send CREATED in round τ_1. Thus, the execution ensemble is $\text{EXEC}^{\text{create}}_{\Pi,\mathcal{A},\mathcal{E}}$:= {createInfo[τ_0], obsSet($a_0, \tau_0 + 1$), prepareInfo[$\tau_0 + 1 + \tau_{pre}$], createFund[$\tau_0 + 2 + \tau_{pre}$], obsSet($a_1, \tau_0 + 3 + \tau_{pre}$), CREATED[$\tau_1$]}.

Ideal World: After sending CREATE in round τ_0 to $\mathcal{F}$, the simulator $\mathcal{S}$ sends createInfo to B. If A receives createInfo from B in round $\tau_0 + 1$, $\mathcal{S}$ informs $\mathcal{F}$ and performs a_0 on behalf of A in round $\tau_0 + 1$. If this is successful, $\mathcal{S}$ sends prepareInfo to B in round $\tau_0 + 1 + \tau_{pre}$. If B sends prepareInfo to A, $\mathcal{S}$ sends createFund to B in round $\tau_0 + 2 + \tau_{pre}$. If B sends createFund to A, received in $\tau_0 + 3 + \tau_{pre}$, $\mathcal{S}$ performs a_1 on behalf of A. If the funding transaction tx_F is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + 3 + \tau_{pre} + \Delta$, finally $\mathcal{S}$ informs $\mathcal{F}$ to send CREATED in round τ_1. Thus, the execution ensemble is $\text{EXEC}^{\text{create}}_{\mathcal{F},\mathcal{S},\mathcal{E}}$:= {createInfo[τ_0], obsSet($a_0, \tau_0 + 1$), prepareInfo[$\tau_0 + 1 + \tau_{pre}$], createFund[$\tau_0 +$

$2+\tau_{pre}]$, obsSet$(a_1, \tau_0+3+\tau_{pre})$, CREATED$[\tau_1]\}$. □

Lemma 2. *The ForceClose subprotocol of* Π *GUC-realizes the ForceClose subprocedure of* $\mathcal{F}$.

Proof. We start by considering the case where P is honest and Q is corrupted.

Real World: Taking the latest valid payment transaction tx_{Q_j} which has been signed by Q, party P performs action a_0 := "sign tx_{Q_j} and post it on $\mathbb{B}$" in round τ_0. If tx_{Q_j} is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0+\Delta$, P will send forceClose to Q and CLOSED to $\mathcal{E}$ in round τ_1. Thus, the execution ensemble is $\text{EXEC}^{\text{forceclose}}_{\Pi,\mathcal{A},\mathcal{E}}$:= {obsSet(a_0,τ_0), forceClose$[\tau_1]$, CLOSED$[\tau_1]\}$.

Ideal World: Taking the latest valid payment transaction tx_{Q_j} owned by P, the simulator $\mathcal{S}$ performs action a_0 in round τ_0. If tx_{Q_j} is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0+\Delta$, it will send forceClose and CLOSED in round τ_1. Thus, the execution ensemble is $\text{EXEC}^{\text{forceclose}}_{\mathcal{F},\mathcal{S},\mathcal{E}}$:= {obsSet(a_0,τ_0), forceClose$[\tau_1]$, CLOSED$[\tau_1]\}$.

Now we consider the case where Q is honest and P is corrupted.

Real World: Upon receiving forceClose from P in round t_0 and observing that tx_{Q_j} has appeared on $\mathbb{B}$, party Q checks if there exists $tx_{P_i}^{\hookrightarrow tx_{Q_j}}$, which is the latest resolve transaction for tx_{Q_j} and signed by P. If no such transaction exists, Q immediately sends CLOSED in round t_0. Otherwise, Q performs action a_0 := "sign $tx_{P_i}^{\hookrightarrow tx_{Q_j}}$ and post it on $\mathbb{B}$" in round t_0. If $tx_{P_i}^{\hookrightarrow tx_{Q_j}}$ is confirmed on $\mathbb{B}$ in round $t_1 \leq t_0+\Delta$, Q will send RESOLVED and CLOSED in round t_1. Thus, the execution ensemble is either $\text{EXEC}^{\text{forceclose}}_{\Pi,\mathcal{A},\mathcal{E}}$:= {CLOSED$[t_0]\}$ or $\text{EXEC}^{\text{forceclose}}_{\Pi,\mathcal{A},\mathcal{E}}$:= {obsSet(a_0,t_0), RESOLVED$[t_1]$, CLOSED$[t_1]\}$.

Ideal World: Upon receiving forceClose from P in round t_0 and observing that tx_{Q_j} has appeared on $\mathbb{B}$, the simulator $\mathcal{S}$ checks whether a valid $tx_{P_i}^{\hookrightarrow tx_{Q_j}}$ exists. If no such transaction is found, $\mathcal{S}$ instructs $\mathcal{F}$ to output CLOSED in round t_0. Otherwise, $\mathcal{S}$ performs action a_0 in round t_0. If $tx_{P_i}^{\hookrightarrow tx_{Q_j}}$ is confirmed on $\mathbb{B}$ in round $t_1 \leq t_0+\Delta$, $\mathcal{S}$ instructs $\mathcal{F}$ to output RESOLVED and CLOSED in round t_1. Thus, the execution ensemble is either $\text{EXEC}^{\text{forceclose}}_{\mathcal{F},\mathcal{S},\mathcal{E}}$:= {CLOSED$[t_0]\}$ or $\text{EXEC}^{\text{forceclose}}_{\mathcal{F},\mathcal{S},\mathcal{E}}$:= {obsSet(a_0,t_0), RESOLVED$[t_1]$, CLOSED$[t_1]\}$. □

Lemma 3. *The Pay phase of protocol* Π *GUC-realizes the Pay phase of functionality* $\mathcal{F}$.

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

Real World: Upon receiving PAY in round τ_0, A performs the following steps: informs B of the new payment, generates and signs punishment transactions for the revoked payment, generates and signs payment and resolve transactions for the new payment, and sends these transactions to B to complete the payment. For better readability, the execution ensemble $\text{EXEC}^{\text{Pay}}_{\Pi,\mathcal{A},\mathcal{E}}$, which captures the steps visible to $\mathcal{E}$, along with their dependencies, is presented as a list below.

- revokeReq to B in round τ_0
- generate and sign the punishment transaction in round τ_0+2 (if received revokeInfo from B)
- generate and sign the payment transactions in round $\tau_0+2+\tau_g$
- PAY-REQ to $\mathcal{E}$ in round $\tau_1 \leq \tau_0+2+t_g+t_{\text{stp}}$
- payInfo to B in round τ_1
- PAID to $\mathcal{E}$ in round τ_1+2 (if received payCom from B)

Ideal World: Similarly, to enhance readability, the execution ensemble $\text{EXEC}^{\text{Pay}}_{\mathcal{F},\mathcal{S},\mathcal{E}}$, which captures the steps visible to $\mathcal{E}$, along with their dependencies and whether they are executed by $\mathcal{S}$ or $\mathcal{F}$, is presented as a list below.

- revokeReq to B in round τ_0 ($\mathcal{S}$)
- generate and sign the punishment transaction in round τ_0+2 (if received revokeInfo from B) ($\mathcal{S}$)
- generate and sign the payment transactions in round $\tau_0+2+\tau_g$ ($\mathcal{S}$)
- PAY-REQ to $\mathcal{E}$ in round $\tau_1 \leq \tau_0+2+t_g+t_{\text{stp}}$ ($\mathcal{F}$)
- payInfo to B in round τ_1 ($\mathcal{S}$)
- PAID to $\mathcal{E}$ in round τ_1+2 (if received payCom from B) ($\mathcal{F}$)

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

Real World: Upon A receiving PAY in round τ_0, if B recieves revokeReq in round t_0, B proceeds as follows: generates the new revocation secret and corresponding hash, revokes the previous payment transaction, and informs A to finalize the payment. Analogous to previous case, the execution ensemble $\text{EXEC}^{\text{Pay}}_{\Pi,\mathcal{A},\mathcal{E}}$, which captures the steps visible to $\mathcal{E}$, along with their dependencies, is presented as a list below to improve readability.

- REVOKE-REQ to $\mathcal{E}$ in round t_0 (if received revokeReq from A)
- generate revocation secret and hash in round t_0
- revokeInfo to A in round t_0+t_g
- payCom to A in round $t_1 \leq t_0+t_g+2+t_{\text{stp}}$ (after receiving payInfo in that round)
- PAID to $\mathcal{E}$ in round t_1

Ideal World: Similarly, to enhance readability, the execution ensemble $\text{EXEC}^{\text{Pay}}_{\mathcal{F},\mathcal{S},\mathcal{E}}$, which captures the steps visible to $\mathcal{E}$, along with their dependencies and whether they are executed by $\mathcal{S}$ or $\mathcal{F}$, is presented as a list below.

- REVOKE-REQ to $\mathcal{E}$ in round t_0 (if received revokeReq from A) ($\mathcal{F}$)
- generate revocation secret and hash in round t_0 ($\mathcal{S}$)
- revokeInfo to A in round t_0+t_g ($\mathcal{S}$)
- payCom to A in round $t_1 \leq t_0+t_g+2+t_{\text{stp}}$ (after receiving payInfo from A in that round) ($\mathcal{S}$)
- PAID to $\mathcal{E}$ in round t_1 ($\mathcal{F}$)

□

Lemma 4. *The Close phase of protocol* Π *GUC-realizes the Close phase of functionality* $\mathcal{F}$.

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

Real World: After receiving `CLOSE` in round τ_0, A performs action a_0 := "generate and sign closing transaction tx_C". In case of success, A will send closeInfo to B in round $\tau_0 + \tau_g$. Upon receiving closeInfo from B in round $\tau_0 + \tau_g + 1$, A proceeds with action a_1 := "post tx_C on $\mathbb{B}$". If tx_C is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + \tau_g + 1 + \Delta$, A sends `CLOSED`. Otherwise, A executes action a_2 := `ForceClose`. Thus, the execution ensemble is either $\text{EXEC}^{\text{close}}_{\Pi,\mathcal{A},\mathcal{E}}$:= {obsSet(a_0, τ_0), closeInfo[$\tau_0 + \tau_g$], obsSet($a_1, \tau_0 + \tau_g + 1$), `CLOSED`[τ_1]} or $\text{EXEC}^{\text{close}}_{\Pi,\mathcal{A},\mathcal{E}}$:= {obsSet(a_0, τ_0), closeInfo[$\tau_0 + \tau_g$], obsSet($a_1, \tau_0 + \tau_g + 1$), obsSet(a_2, τ_1)}.

Ideal World: After sending `CLOSE` in round τ_0, $\mathcal{S}$ performs a_0 in round τ_0 and sends closeInfo to B in round $\tau_0 + \tau_g$. Upon receiving closeInfo in round $\tau_0 + \tau_g + 1$, $\mathcal{S}$ performs a_1 on behalf of A. If the closing transaction tx_C is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + \tau_g + 1 + \Delta$, $\mathcal{S}$ instructs $\mathcal{F}$ to send `CLOSED` in round τ_1. Otherwise, $\mathcal{S}$ executes action a_2 and instruct $\mathcal{F}$ to do the same. Thus, the execution ensemble is either $\text{EXEC}^{\text{close}}_{\mathcal{F},\mathcal{S},\mathcal{E}}$:= {obsSet(a_0, τ_0), closeInfo[$\tau_0 + \tau_g$], obsSet($a_1, \tau_0 + \tau_g + 1$), `CLOSED`[τ_1]} or $\text{EXEC}^{\text{close}}_{\Pi,\mathcal{A},\mathcal{E}}$:= {obsSet(a_0, τ_0), closeInfo[$\tau_0 + \tau_g$], obsSet($a_1, \tau_0 + \tau_g + 1$), obsSet(a_2, τ_1)}. □

Lemma 5. *The Punish phase of protocol* Π *GUC-realizes the Punish phase of functionality* $\mathcal{F}$.

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

Real World: Upon receiving `PUNISH` from $\mathcal{E}$ in round τ_0, A checks whether a transaction appears on $\mathbb{B}$ that belongs to any revoked payment of one of its channels. If such a transaction exists, A performs action a_0 := "post punishment transaction tx_{pnsh} on $\mathbb{B}$" in round τ_0 for the revoked transaction. If tx_{pnsh} is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + \Delta$, A sends `PUNISHED` in round τ_1. Thus, the execution ensemble is $\text{EXEC}^{\text{punish}}_{\Pi,\mathcal{A},\mathcal{E}}$:= {obsSet(a_0, τ_0), `PUNISHED`[τ_1]}.

Ideal World: At the end of every round τ_0, the ideal functionality $\mathcal{F}$ checks if there exists a transaction on $\mathbb{B}$ corresponding to any revoked payment associated with an active channel. If such a transaction exists, $\mathcal{S}$ performs action a_0 on behalf of the honest party A. Once tx_{pnsh} is confirmed on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + \Delta$, send `PUNISHED` in round τ_1. Thus, the execution ensemble is $\text{EXEC}^{\text{punish}}_{\mathcal{F},\mathcal{S},\mathcal{E}}$:= {obsSet(a_0, τ_0), `PUNISHED`[τ_1]}. □

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

Proof. This theorem follows directly from Lemmas 1 to 5 via a standard hybrid argument. □

Ideal Functionality $\mathcal{F}^{\mathbb{B}(\Delta,\Sigma,\mathcal{V})}(T_p,k)$

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

Both agreed: If already received (CREATE, γ, tid_B) $\overset{\tau}{\hookleftarrow} B$, where $\tau_0 - \tau \leq T_p$: If $tx_F := tx([tid_A, tid_B], \langle \phi, \gamma.\text{cash} \rangle)$ for some ϕ, appears on $\mathbb{B}$ in round $\tau_1 \leq \tau + \Delta + T_p$, set $\omega := (\theta_A, \theta'_A, \theta_B, \theta'_B, \theta^*)$ with each element as a copy of γ.st, $\Gamma(\gamma.\text{id}) := (\gamma, tx_F, \omega)$ and (CREATED, γ.id) $\overset{\tau_1}{\hookrightarrow}$ γ.users. Else stop.

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

Pay: Upon (PAY, id, $\overrightarrow{\theta}$, t_{stp}) $\overset{\tau_0}{\hookleftarrow} A$, (REVOKE-REQ, id, $\overrightarrow{\theta}$, t_{stp}) $\overset{\tau_1 \leq \tau_0 + T_p}{\longleftarrow} B$, parse $(\gamma, tx_F, \omega) := \Gamma(\text{id})$, set $\gamma' := \gamma, \gamma'.\text{st} := \overrightarrow{\theta}, \omega' := \omega, \omega'.\theta_B := \omega.\theta'_B, \omega'.\theta'_B := \overrightarrow{\theta}, \omega'.\theta^* = \omega.\theta'_B$:

(1) If (REVOKE, id) $\overset{\tau_2 \leq \tau_1 + T_p}{\longleftarrow} B$, then let $\mathcal{S}$ define $\overrightarrow{tid}$ and (PAY-REQ, id, $\overrightarrow{tid}$) $\overset{\tau_3 \leq \tau_2 + T_p}{\longleftarrow} A$. Else stop *(reject)*.

(2) If (PAY-OK, id) $\overset{\tau_4 \leq \tau_3 + t_{\text{stp}}}{\longleftarrow} A$, update $\Gamma(\text{id}) := (\gamma', tx_F, \omega')$, send (PAID, id, $\overrightarrow{\theta}$) $\overset{\tau_5 \leq \tau_4 + T_p}{\longleftarrow}$ γ.users and stop *(accept)*. Else run ForceClose(id) and stop.

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

Both agreed: If already received (CLOSE, id) $\overset{\tau}{\hookleftarrow} B$, where $\tau_0 - \tau \leq T_p$, let $(\gamma, tx_F, \omega) := \Gamma(\text{id})$ and distinguish:

- If $tx_C := tx(tx_F, \langle \phi_A, \gamma.\text{st.bal}(A) \rangle, \langle \phi_B, \gamma.\text{st.bal}(B) \rangle)$ appears on $\mathbb{B}$ in round $\tau_1 \leq \tau_0 + \Delta$, set $\Gamma(\text{id}) := \perp$, send (CLOSED, id) $\overset{\tau_1}{\hookrightarrow}$ γ.users and stop.
- Else, if at least one of the parties is not honest, run ForceClose(id). Else, output (ERROR) $\overset{\tau_0 + \Delta}{\longrightarrow}$ γ.users and stop.

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

Punish: (executed at the end of every round τ_0) For each id $\in \{0,1\}^*$ s.t. $\Gamma(\text{id}) \neq \perp$, parse $(\gamma, tx_F, \omega) := \Gamma(\text{id})$, check if $\mathbb{B}$ contains $tx' := tx(tx_F, \langle \phi'_A, v_A \rangle, \langle \phi'_B, v_B \rangle)$ with $v_A + v_B = \gamma.\text{cash}$. If yes, let $\overrightarrow{\theta}$ be the current state, $\overrightarrow{\theta}.\text{bal}(A) := v_A$, $\overrightarrow{\theta}.\text{bal}(B) := v_B$, $\tau = \tau_0 + \Delta$, distinguish:

Close: If $\overrightarrow{\theta} \in \omega$, distinguish:

- If $\overrightarrow{\theta} = \gamma.\text{st}$, set $\Gamma(\text{id}) := \perp$, (CLOSED, id) $\overset{\tau_1 \leq \tau}{\longrightarrow}$ γ.users if not sent yet.
- Else, if $\overrightarrow{\theta} \neq \omega.\theta^*$ and $tx_{res}(tx', \langle \phi_A, v'_A \rangle, \langle \phi_B, v'_B \rangle)$ appears on the $\mathbb{B}$ within round τ, then set $\Gamma(\text{id}) := \perp$, (RESOLVED, id) $\overset{\tau}{\hookrightarrow}$ γ.users, (CLOSED, id) $\overset{\tau}{\hookrightarrow}$ γ.users and stop.
- Otherwise, output (ERROR) $\overset{\tau}{\hookrightarrow}$ γ.users and stop.

Punish: Else, if $tx_{pnsh}(tx', \langle \phi_X, \gamma.\text{cash} - \overrightarrow{\theta}.\text{bal}(X) \rangle)$ appears on $\mathbb{B}$ within round τ, then set $\Gamma(\text{id}) := \perp$, and for the honest party X, (PUNISHED, id) $\overset{\tau}{\hookrightarrow} X$ and stop.

Subprocedure ForceClose(id): Let τ_0 be the current round and $(\gamma, tx_F, \omega) := \Gamma(\text{id})$. If within Δ rounds tx_F is still an unspent transaction on $\mathbb{B}$, (ERROR) $\overset{\tau_0 + \Delta}{\longrightarrow}$ γ.users and stop. Else, latest in round $\tau_0 + 2 \cdot \Delta$, message $m \in \{\text{CLOSED}, \text{PUNISHED}, \text{ERROR}\}$ is output via Punish.

Figure 9: The Ideal Functionality.