USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

On the Atomicity and Efficiency of Blockchain Payment Channels

Di Wu, Shoupeng Ren, and Yuman Bai, *The State Key Laboratory of Blockchain and Data Security, Zhejiang University;* Lipeng He, *University of Waterloo;* Jian Liu, *The State Key Laboratory of Blockchain and Data Security, Zhejiang University;* Wu Wen, *International Business School, Zhejiang University;* Kui Ren and Chun Chen, *The State Key Laboratory of Blockchain and Data Security, Zhejiang University*

https://www.usenix.org/conference/usenixsecurity25/presentation/wu-di

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

On the Atomicity and Efficiency of Blockchain Payment Channels

Di Wu[1], Shoupeng Ren[1], Yuman Bai[1], Lipeng He[3], Jian Liu[1]*, Wu Wen[2], Kui Ren[1], Chun Chen[1]
[1]*The State Key Laboratory of Blockchain and Data Security, Zhejiang University*
[2]*International Business School, Zhejiang University*
[3]*University of Waterloo*
{wu.di, spren, baiyuman}@zju.edu.cn, lipeng.he@uwterloo.ca
liujian2411@zju.edu.cn, wuwen@intl.zju.edu.cn, {kuiren, chenc}@zju.edu.cn

Abstract

Payment channels are a promising solution for scaling cryptocurrency payments by enabling secure off-chain transactions. However, existing protocols, including the widely-deployed Lightning Network and the state-of-the-art Sleepy Channels, suffer from a fundamental flaw: non-atomic state transitions can result in multiple valid states coexisting, introducing race conditions and ambiguity in protocol execution. This ambiguity can be exploited to cause unexpected financial loss. We first formalize existing protocols into a common paradigm and prove that such flaws are inherent to their design, preventing balance security. To overcome this, we propose an atomic paradigm that guarantees atomic state transitions while preserving all desired functionality. Based on this paradigm, we design ULTRAVIOLET, the first payment channel protocol that achieves atomicity by introducing the novel Resolve Mechanism. We formally prove that ULTRAVIOLET satisfies balance security under the Universal Composability framework. In addition, ULTRAVIOLET reduces the number of required messages per transaction by half compared to existing solutions. Our evaluation across multiple regions shows that ULTRAVIOLET reduces latency by 37% and 52% compared to the Lightning Network and Sleepy Channels, respectively, and achieves comparable throughput to the Lightning Network and 2× that of Sleepy Channels.

1 Introduction

Decentralized cryptocurrencies have introduced a revolutionary payment system, gaining 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 [25]—the most prominent blockchain-based cryptocurrency—requires 10 minutes or more to confirm a transaction and supports a maximum throughput of only 7 TPS [12].

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. The most well-known PC protocol is the Lightning Network (LN) [1], deployed on Bitcoin, which hosts bitcoins worth 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) protocol, Sleepy Channel (SC) [7], follows the Lightning-style design and introduces further improvements, removing reliance on watchtowers [8, 22].

Non-Atomicity Issue. Despite its widespread use, LN has been found to suffer from a design flaw identified in a recent study [28] (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. Such 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 allows a malicious payee to inflict unavoidable financial loss to a payer by simply being unresponsive. Consider a buyer (payer) using LN to purchase product from a seller (payee). Following the protocol, the buyer first sends a signed transaction representing the post-payment state and then awaits the protocol-specified response to proceed. At this point, the new state is created,

*Jian Liu is the corresponding author

but the prior state is not revoked, leaving two valid states. A malicious seller, upon receiving the transaction, can retain it and withhold the response. Consequently, due to the presence of two simultaneously valid states and the absence of the expected response, the buyer cannot determine the actual execution status of the protocol.

- If the buyer considers the payment is proceeding, assumes the lack of response is due to network issues, then takes no action, the malicious seller can broadcast the received transaction to finalize the channel with post-payment state, in an incomplete protocol execution. As a result, the buyer unwittingly loses funds.
- Alternatively, the buyer consider the payment is aborted then unilaterally closes the channel using a prior valid state. However, this risks a *race condition* with the new transaction that the malicious seller actually has received. If the seller's transaction confirms first, the outcome mirrors the earlier scenario. Even if the buyer's transaction confirms first, they incur unavoidable transaction fees for finalizing channel.

Notably, this type of financial loss, where the buyer unwittingly loses money due to uncertainty about whether the payment has completed, should be distinguished from scenarios in which the buyer is certain that the transaction succeeded but the seller refuses to deliver the product. The latter issue, which also occurs in traditional payment methods, falls outside the scope of what the payment protocol itself can address. We focus on the former, a protocol-level design flaw (exists two valid states) that enables such ambiguity to be exploited.

We highlight that these issues stem from the lack of atomicity in channel state transitions. Atomicity requires a direct transition between valid states, whereas existing protocols adopt a multi-step update process involving message exchanges. This introduces a window in which both old and new states remain valid, allowing either party to unilaterally finalize the channel using an outdated state. Attempts to patch this with grace periods, timeouts, or acknowledgments fail to resolve the root cause. Achieving true atomicity demands a fundamental redesign of the payment channel protocol.

Efficiency Limitations. Additionally, the current design of PC 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 messages to revoke the previous state, resulting in significant communication overhead. This overhead leads to high latency, especially in cross-regional settings. Current protocols cannot reduce communication complexity without compromising security or requiring unsupported Bitcoin features, indicating inherent limitations in existing designs.

Contributions. This paper addresses the fundamental limitations of existing PC protocols by proposing ULTRAVIOLET, a novel protocol that achieves stronger security and higher efficiency. We summarize our contributions as follows:

- **Common Paradigm.** We first provide a formal definition of *balance security*, a core security property for PCs. We also formalize mainstream Lightning-style protocols (e.g., LN and SC) into a unified *common paradigm*, and show that due to the lack of atomicity, this paradigm fails to satisfy balance security. We further reveal that designs adhering to common paradigm cannot fix this issue.
- **Atomic Paradigm and ULTRAVIOLET Protocol.**We propose a new *atomic paradigm* that provably enforces atomicity, eliminating ambiguity caused by the coexistence of multiple valid states and race conditions. Based on this paradigm, we design ULTRAVIOLET, the first payment channel protocol to achieve atomicity by introducing the novel Resolve Mechanism, while reducing the number of off-chain messages per payment.
- **Security Proof.** We formally prove the security of ULTRAVIOLET within the Universal Composability framework. To this end, we design an ideal functionality that captures balance security. The formal protocol description and security proof are provided in Appendix A.
- **Comprehensive Evaluation.** We implement and evaluate ULTRAVIOLET, measuring both its off-chain payment performance and on-chain cost. In cross-region tests, ULTRAVIOLET reduces latency by 37% compared to LN and 52% compared to SC, while achieving comparable TPS to LN and 2× that of SC.

2 Preliminaries and Related Works

This section introduces background on Bitcoin and related work on payment channels, and summarize the Universal Composability framework for formal security analysis.

2.1 Bitcoin

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: $\forall i \in [1,k], \pi_i \models \phi_i$.

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: *creation*, *update*, and *finalization*. In the *creation phase*, participants create a funding transaction that locks their shared funds in a multi-signature UTXO, typically using Bitcoin's native `OP_CHECKMULTISIG` or Schnorr-based schemes such as MuSig/MuSig2. During the *update phase*, participants exchange new commitment transactions representing the updated balance state S_i. To prevent outdated states from being broadcast, the Lightning Network employs a *punishment mechanism*. A common implementation is based on preimage and hash. Specifically, when transitioning from state S_{i-1} to S_i, both parties perform the following: (1) each party generates a new revocation hash-secret pair (rh_{A_i}, rs_{A_i}) and (rh_{B_i}, rs_{B_i}), and exchanges the hashes rh_{A_i}, rh_{B_i}; (2) using the received hashes, they create and exchange new commitment transactions reflecting S_i; (3) both parties reveal the previous secrets $rs_{A_{i-1}}$, $rs_{B_{i-1}}$ to revoke S_{i-1}. This ensures any broadcast of the outdated state S_{i-1} enables the counterparty to claim all funds as penalty using the revealed rs_{i-1} and signature. In the *finalization phase*, participants can either cooperatively broadcast a settlement transaction or forcibly close the channel by publishing the latest commitment transaction they hold.

Recent advances led to *Sleepy Channel* [7], a SOTA protocol that eliminates the need for watchtowers 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, SC leverages economic incentives and time-bound constraints to ensure security.

The concept of payment channels can be extend to payment channel networks (PCNs) where two users without a direct channel can connect with each other through a path of other people's PCs via a routing mechanism. One of the most popular implementation is the Basis of Lightning Technology (BOLTs) powering LN. Moreover, Payment Channel Hubs (PCHs) 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 [14, 20, 27], channel re-balancing [19], better payment routing [24], etc.

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 PCs, 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 PC protocols [4, 7, 17], including the SOTA one, Sleepy Channels.

3 Threat Model and Balance Security

In this section, we introduce the threat model applied in this work, and then formally define the notion of balance security for a single payment channel.

3.1 Threat Model

Blockchain. We assume a public and permissionless blockchain $\mathbb{B}$. The majority of the participants maintaining the blockchain (e.g., miners) are honest. The blockchain has a bounded confirmation time Δ [26]. meaning that once a valid transaction is broadcast by an honest party, it will be confirmed and become irreversible within Δ time with overwhelming probability. However, due to potential *race conditions*, a transaction may still be invalidated if a conflicting transaction is accepted earlier.

Cryptographic Primitives. We assume that the underlying blockchain provides basic cryptographic primitives. In particular, we require an EUF-CMA secure digital signature scheme to prevent forgery, and a collision-resistant hash scheme to ensure the security of the revocation mechanism.

Adversary. We assume that at most one participant in the payment channel is adversarial. A setting with two malicious parties is excluded, as meaningful security guarantees require at least one honest participant. We consider a computationally bounded Byzantine adversary $\mathcal{A}$, who may arbitrarily deviate from the protocol. For example, $\mathcal{A}$ may send arbitrary messages to the counterparty, or refuse to respond. However, $\mathcal{A}$ cannot forge digital signatures or find hash collisions, limited by their Probabilistic Polynomial-Time (PPT) computational capabilities as per standard cryptographic assumptions.

Rational Participant. All participants, including the potential adversary $\mathcal{A}$, are assumed to be financially rational. That is, they aim to maximize their economic benefit and will not take actions that knowingly result in their own financial loss.

Scope. Our threat model focuses on the security of the payment channel during protocol execution, specifically on whether a malicious participant can cause financial loss to an honest counterparty due to flaws in the protocol itself. We do not consider threats arising from network partitions or failures in the underlying blockchain. Furthermore, we do not consider real-world enforcement issues that fall outside protocol logic, such as a seller refusing to deliver goods after receiving payment. This is a practical concern that even cash cannot prevent and is typically addressed through legal protection.

3.2 Balance Security

Balance security is the central security property for payment channel protocols, ensuring that honest participants never lose funds they are entitled to. While several prior works [4–6, 19, 23, 29] have considered the notion of balance security, none has provided a formal definition in the context of a single payment channel. In this work, we provide a formal definition tailored to the single channel setting.

Definition 3.1 (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*.*

Consensus-approved state. A *consensus-approved state* is a channel state resulting from a completed update phase, where both parties have correctly performed all required actions. It represents a mutually agreed-upon balance distribution.

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.

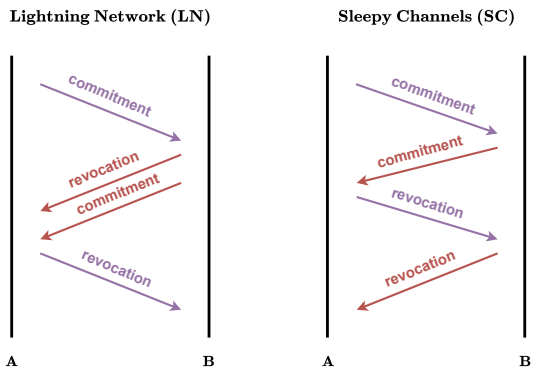


Figure 1: Message flow during the update process in LN and SC, both maintaining synchronized local states with explicit state revocation, despite differences in sequence and details.

- **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.** The channel can be closed either cooperatively by both parties or forcibly by either party at any time. In all cases, an honest participant must receive at least the balance distributed to them in the latest consensus-approved state.

It should be noted that our definition focuses solely on the safety of balances *within* the channel and does not consider the blockchain costs (often termed *Transaction Fees*) associated with submitting transactions for on-chain operations.

4 A Common Paradigm for PC Protocols

In this section, we first abstract a broad class of existing PC protocols into a *common paradigm*, then show that its inherently non-atomicity breaks balance security. We further argue that this is an inherent flaw, meaning any protocol adhering to this paradigm cannot fully rectify it. Finally, we remark on the issue within SC.

4.1 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: $S_i^A = S_i^B$.

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

$$\exists P, Q \in \{A, B\},\ P \neq Q,\ s.t.$$
$$(\vdash S_{n+1}^P \prec \vdash S_{n+1}^Q) \wedge (\vdash S_{n+1}^P \prec \nvdash S_n^P) \wedge (\vdash S_{n+1}^Q \prec \nvdash S_n^P)$$

Note that a new state becomes consensus-approved only upon the completion of State Replacement. Prior to this, only the previous state is considered consensus-approved.

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_{i+1}^A and sends it to B. Then B constructs S_{i+1}^B—consistent with S_{i+1}^A—and returns it to A. Importantly, each party revokes their old state (S_i^A, S_i^B) 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_i^A or S_i^B, 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.

Although a range of PCs, including SC and LN (which are specifically depicted in Figure 1), as well as Generalized Channels (GC) [4], Eltoo [13], and others, present distinct implementation details and message sequences, we can nevertheless formalize their shared adherence to a *common paradigm*:

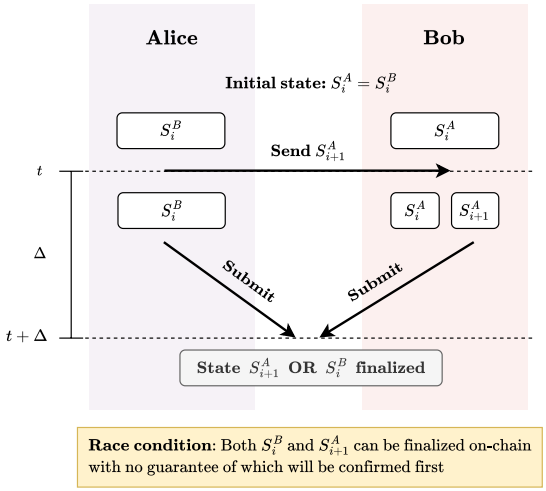


Figure 2: Race condition in the common paradigm: Without atomic updates, either S_{i+1}^A or S_i^B may be finalized, violating balance security.

- 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.

4.2 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. We now provide an in-depth explanation of the example presented in Section 1.

Let A be the honest buyer and B the malicious seller. Suppose they are currently operating on the valid state (S_i^A, S_i^B), and A initiates a payment to B to transition the channel to a new state (S_{i+1}^A, S_{i+1}^B). Initially, A sends S_{i+1}^A to B. Upon receiving the new state, B does not respond and instead submits S_{i+1}^A 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 LN or a commitment message representing the new state (S_{i+1}^B) in the SC protocol, A may submit the valid state S_i^B on-chain to forcibly close the channel to protect its funds.

At this point, the update process has not been completed, so (S_i^A, S_i^B) 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 4.2 (Atomicity)**.** *A state transition $S_i \to S_{i+1}$ is atomic if and only if:*

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

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 4.1 (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:

$$(S^A_i \neq \bot) \wedge (S^A_{i+1} \neq \bot) \implies (S^A_i = S^A_{i+1})$$

Here, S^A_{i+1} is the new state generated by A following S^A_i, satisfying $S^A_i \neq S^A_{i+1}$. According to the *State Replacement* principle, once B receives the new state ($\vdash S^A_{i+1}$), it gains the ability to finalize the channel using S^A_{i+1} without incurring any financial loss, meaning $S^A_{i+1} \neq \bot$. However, before B revokes the previous state ($\nvdash S^A_i$), it holds that $S^A_i \neq \bot$. There exists a time window prior to the reception of the new state and the revocation of the previous state, during which $S^A_i \neq \bot$, $S^A_{i+1} \neq \bot$, and $S^A_i \neq S^A_{i+1}$. That is, this 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 determined by the network latency between the them.

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*.

4.3 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 4.2. *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.*

5 Atomic Paradigm

In this section, we introduce the *atomic paradigm*. We formally prove its atomicity and demonstrate that this property guarantees balance security. Finally, through illustrative examples, we show how protocols adhering to this paradigm can mitigate the non-atomic issues.

5.1 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 5.1 (State Transition)**.** *When receiving a new state S^P_{n+1}, all existing states simultaneously ($\sim$) transform into this new state:*

$$\forall Q \in \{A,B\}, \exists P \in \{A,B\},\ s.t.\ \forall k, S^Q_k \neq \bot,$$
$$(\vdash S^P_{n+1}) \sim (S^Q_k \to S^P_{n+1})$$

where $S^Q_k \to S^P_{n+1}$ denotes the automatic transition of state S^Q_k sent by participant Q into the new state S^P_{n+1}.

The channel evolves through three main phases:

- **Creation.** Like common paradigm, both participants jointly establish the channel with initial state (S^A_0, S^B_0).
- **Update.** After creation, any party can perform updates through *State Transition*. To illustrate this, consider the following example. Suppose that current states are:
  - Several states generated by A, held by B: $S^A_0, \ldots, S^A_i$
  - Several states generated by B, held by A: $S^B_0, \ldots, S^B_j$

  When B initiates a payment by sending a new state S^B_{j+1} to A, all prior states are automatically transformed to the new state: $\forall k \leq i : S^A_k \to S^B_{j+1}$ and $\forall k \leq j : S^B_k \to S^B_{j+1}$.
- **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 *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.

Note that the atomic paradigm does not conflict with revocation mechanisms utilized for purposes such as pruning, provided that its core principle remains uncompromised.

5.2 Security Analysis of the Atomic Paradigm

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

Theorem 5.1 (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 at some point:

$$(S_i^A \neq \perp) \wedge (S_j^B \neq \perp) \wedge (S_i^A \neq S_j^B)$$

Here, S_j^B is the new state following S_i^A during an update. According to the principle *State Transition*, 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 point such that $S_i^A \neq S_j^B$, which directly contradicts the assumption. The paradigm is proven to satisfy atomicity. □

Theorem 5.2 (Balance Security Preservation)**.** *The atomic paradigm maintains balance security across creation, update, and finalization phases according to Definition 3.1.*

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 A and B agree on the new state S_{i+1}^A. The update phase adheres to atomicity, satisfying:

$$\forall Q \in \{A,B\},\ \forall k, S_k^Q \neq \perp, (\vdash S_{i+1}^A) \sim (S_k^Q \rightarrow S_{i+1}^A)$$

This ensures that during the update process, the previous state remains the only valid consensus-approved state. Once the update is completed, the new state S_{i+1}^A 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, ensuring no loss of funds for the participants. □

Corollary 5.3 (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.

To illustrate how the atomic paradigm mitigates the non-atomic issue highlighted in Section 1, consider the following scenario. Assume a buyer uses a payment channel protocol adhering to the atomic paradigm to purchase a product from a seller. Once the buyer transmits a signed transaction representing the post-purchase state to the seller, the buyer achieves certainty of payment completion, irrespective of receiving a response from the seller. This design shields the buyer from the uncertainty of payment status, preventing unwitting finance loss arising from payment ambiguity or fraudulent claims by a malicious seller (e.g., alleging the payment is incomplete due to network partitions). Furthermore, the seller cannot compel the buyer to bear channel settlement fees. This is because if either party unilaterally closes the channel at any time with any non-revoked state, this state will deterministically transition to the latest valid state, ensuring the channel always settles based on this most recent state.

6 ULTRAVIOLET Protocol

This section elaborates on our novel payment channel protocol, ULTRAVIOLET. We first introduce the protocol using an incremental construction approach to show the core ideas. Subsequently, we formalize its security guarantees within the UC framework, presenting ideal functionality and analyzing balance security. Finally, we provide a concrete specification of the protocol, including the message interactions and transaction structures.

6.1 Protocol Overview

To develop a protocol that adheres to atomic paradigm, we begin with an intuitive toy protocol as a starting point and iteratively refine its design. For clarity, we focus on the design of Update phase, which differentiates our protocol from the common paradigm followed by 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 a toy protocol. During each update, if the payee holds a valid state, they first revoke it. We recall that a "valid" state means it can be submitted to the blockchain and cannot occur any finance loss (punishment).

(a) Toy protocol

1. B sends the revocation secret for S^A_{i-1}, if not sent yet.
2. A sends the revocation secret for S^B_j, if not sent yet.
3. A sends the new state S^A_i to B.

(b) Remove Punishment Mechanism

1. A sends the new state S^A_i to B.

(c) Introduce Resolve Mechanism

1. A sends the new state S^A_i to B, along with the set of resolve states $\{S^{\curvearrowright S^B_n}_{A,i} \mid n \in [0, j]\}$.

(d) Final Protocol

1. B sends the revocation secret for S^A_{i-2}, if such a state exists.
2. A sends the new state S^A_i to B, along with the resolve state $S^{\curvearrowright S^B_j}_{A,i}$ and, if S^B_{j-1} exists, the resolve state $S^{\curvearrowright S^B_{j-1}}_{A,i}$.

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.

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, guaranteeing atomicity.

However, this protocol introduces critical issues. If A (the payer) is malicious while B (the payee) is honest, A can withhold any response after B revokes their valid state, leaving B unable to finalize the channel either cooperatively or forcibly since they no longer hold any valid state. Similarly, if A is honest and B is malicious, A completes the state update by revoking valid state and sending 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.

While this toy protocol addresses the race condition issue, it introduces severe fairness and liveness problems. A malicious or unresponsive counterparty can launch a *denial-of-service* (DoS) attack, preventing the honest party from finalizing the channel and potentially locking funds indefinitely.

Remove Punishment Mechanism. To address the issues above, one improvement is to remove the punishment mechanism entirely. As shown in Figure 3(b), the payer directly sends the new state to the payee. In this approach, if one party becomes unresponsive, the honest party can commit their latest valid state to finalize the channel forcibly, allowing them to withdraw funds from the channel.

While this straightforward modification restores liveness, it introduces a critical flaw: both parties may hold multiple valid states, violating the atomicity and reintroducing the race condition problem. A malicious party could exploit this by competing to submit an outdated but favorable state, undermining the balance security of the counterparty.

Introduce Resolve Mechanism. To overcome the issues of prior designs, we propose a novel Resolve Mechanism. As depicted in Figure 3(c), in each 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 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 $S^A_{\text{set}} = \{S^B_n \mid n \in [0, j]\}$. Let S^B_j denote the latest state where both A's balance V_A and B's balance V_B are equal to 10. Now, if A initiates its i-th payment to B, transferring 5 units, 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 $S^{\curvearrowright S^B_j}_{A,i}$ 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, B can use the corresponding resolve state to perform a state transition, ensuring the channel is finalized with the latest state S^A_i. Conceptually, this can be expressed as: $S^B_n \xrightarrow{S^{\curvearrowright S^B_n}_{A,i}} S^A_i$, for $n \in [0, j]$.

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.

The introduction of the resolve mechanism effectively addresses the security issues identified in the previous 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, A can commit the latest state S^B_j held. During the designated time window, if B commits the corresponding resolve state, the channel finalizes in the globally latest state S^A_i. Otherwise, it finalizes in S^B_j, causing financial loss to the malicious B and giving the honest A more than their fair share under 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 for every outdated state 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.

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.
- *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 all historical states and their corresponding resolve states, making the implementation more complex and error-prone.

Final Protocol. To address the previous limitations, we reintroduce the punishment mechanism for pruning, arriving at the final protocol. As depicted in Figure 3(d), during each update, the payee first revokes their second-to-last state. Then, 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 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 each party holds at most two valid states. Consequently, only two resolve states need to be exchanged and stored per update, reducing communication to a constant level and cutting resolve state storage complexity from $O(n^2)$ to $O(1)$, with only $O(n)$ revocation secrets retained.
- *Balance Security.* This protocol preserves atomicity by requiring the payer to send resolve states with the new state. If one party is unresponsive, the honest party can commit their latest valid state. To avoid financial loss, the other party may be forced to submit the corresponding resolve state, ensuring the channel finalizes in the globally latest state. Even if A withholds the new state after receiving a revocation secret, B retains a valid, unrevoked state for finalization, preventing DoS attacks.

6.2 UC Modeling

A. Security Model

To formally model and prove the security of our protocol, we adopt the synchronous Global Universal Composability (GUC) framework [11], following prior work [4–6, 15, 16, 18].

The security analysis considers two execution worlds. In the real world, the protocol Π is executed among a set of parties in the presence of an adversary $\mathcal{A}$. In the ideal world, the parties interact with a trusted ideal functionality $\mathcal{F}$, and a simulator $\mathcal{S}$ emulates the adversary's effect. In both worlds, an external environment $\mathcal{E}$ provides inputs to and receives outputs from the parties, and interacts with either the adversary or the simulator. Our goal is to prove that the real-world protocol UC-realizes an ideal functionality that captures the notion of balance security (defined in Section 3.2). For clarity and space, we briefly provide the key concepts of the model in this section, and defer the full description to Appendix A.1.

We consider a static corruption model, where the adversary $\mathcal{A}$ may corrupt any one party before the protocol execution begins. We assume a synchronous communication network where protocol execution proceeds in discrete rounds, and messages are transmitted over authenticated channels. Specifically, if a party P sends a message to party Q in round τ, Q receives it at the start of round $\tau+1$, with guaranteed authenticity. The adversary can observe message content, but cannot modify, delay or drop messages. Communications involving the adversary $\mathcal{A}$ or the environment $\mathcal{E}$ are assumed to take zero rounds. The blockchain is abstracted by a global ledger functionality $\mathbb{B}(\Delta, \Sigma, \mathcal{V})$, which models a UTXO-based ledger with bounded confirmation delay Δ, a secure signature scheme Σ, and a set $\mathcal{V}$ of spending conditions. On top of this ledger, we model the payment channel as $\mathcal{F}(T_p, k)$. The parameters T_p and k respectively denote an upper bound on consecutive off-chain communication rounds and the number of ways a channel state can be published.

A protocol Π is said to UC-realize an ideal functionality $\mathcal{F}$ if, for any adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that no environment $\mathcal{E}$ can distinguish the real-world execution from the ideal execution. In the next section, we describe the ideal functionality $\mathcal{F}$ that captures the notion of balance security. In Appendix A.3, we formally prove that our real-world protocol UC-realizes this ideal functionality, thereby achieving balance security within the UC framework.

B. Ideal Functionality

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 4. For brevity, we omit explicit calls to $\mathcal{F}_{clock}$ and $\mathcal{F}_{GDC}$, and adopt 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 natural checks that one would expect $\mathcal{F}$ to make, which could be formally handled by combining a functionality wrapper [4]. 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.

Create. In the Create phase, both participants A and B must first send a $(\texttt{CREATE}, \gamma, tid_p)$ message to $\mathcal{F}$. The tuple γ denotes the channel attributes: a unique identifier γ.id, participants γ.users, total funds γ.cash, and the latest consensus-approved state γ.st. tid_p refers the party P's input for the funding transaction. Upon receiving these messages from both parties, $\mathcal{F}$ checks $\mathbb{B}$ for a funding transaction tx_F spending tid_A and tid_B with output 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 party, 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 `CREATED` to

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 $\tau_1 \leq \tau + \Delta + T_p$, set $\omega := (\theta_A, \theta'_A, \theta_B, \theta'_B, \theta^*)$ with each initialized to $\gamma.\text{st}$, $\Gamma(\gamma.\text{id}) := (\gamma, tx_F, \omega)$, (CREATED, γ.id) $\overset{\tau_1}{\hookrightarrow} \gamma.\text{users}$. 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, $\overrightarrow{\theta}, t_{\text{stp}}$) $\overset{\tau_0}{\hookleftarrow} A$, (REVOKE-REQ, id, $\overrightarrow{\theta}, t_{\text{stp}}$) $\overset{\tau_1 \leq \tau_0 + T_p}{\longrightarrow} B$, 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, $\overrightarrow{tid}$) $\overset{\tau_3 \leq \tau_2 + T_p}{\longrightarrow} A$. 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 $(\text{PAID}, \text{id}, \overrightarrow{\theta})$ $\overset{\tau_5 \leq \tau_4 + T_p}{\longrightarrow} \gamma.\text{users}$ 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 $\tau_1 \leq \tau_0 + \Delta$, set $\Gamma(\text{id}) := \perp$, send (CLOSED, id) $\overset{\tau_1}{\hookrightarrow} \gamma.\text{users}$ and stop.
- Else, if at least one of the parties is not honest, run ForceClose(id). Else, output (ERROR) $\overset{\tau_0 + \Delta}{\longrightarrow} \gamma.\text{users}$ 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 $\text{id} \in \{0,1\}^*$ 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, $\overrightarrow{\theta}.\text{bal}(A) := v_A$, $\overrightarrow{\theta}.\text{bal}(B) := v_B$, distinguish:

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

- If $\overrightarrow{\theta} = \gamma.\text{st}$, set $\Gamma(\text{id}) := \perp$, (CLOSED, id) $\overset{\tau_1 \leq \tau_0 + \Delta}{\longrightarrow} \gamma.\text{users}$ 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 $\tau_0 + \Delta$, and $v'_A + v'_B \in \{v_A, v_B\}$, then set $\Gamma(\text{id}) := \perp$, (RESOLVED, id) $\overset{\tau_0 + \Delta}{\longrightarrow} \gamma.\text{users}$, (CLOSED, id) $\overset{\tau_0 + \Delta}{\longrightarrow} \gamma.\text{users}$ and stop.
- Otherwise, output (ERROR) $\overset{\tau_0 + \Delta}{\longrightarrow} \gamma.\text{users}$ 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 $\tau_0 + \Delta$, 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) $\overset{\tau_0 + \Delta}{\longrightarrow} \gamma.\text{users}$ 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 4: The Ideal Functionality.

both users and updates the map Γ to bind γ.id with (γ, tx_F, ω), indicating successful channel creation. Since the channel is created only after receiving CREATE from both parties, and the spending conditions ϕ of tx_F is jointly authorized by both, *balance security in creation* holds.

Pay. In the Pay phase where A pays B, 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 setup delay for off-chain applications. $\mathcal{F}$ then sends (REVOKE-REQ, id, $\overrightarrow{\theta}$, t_{stp}) to B, requesting B to revoke previous state. If B replies with (REVOKE, id), this indicates that B has agreed to the new state $\overrightarrow{\theta}$. Otherwise, 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, $\overrightarrow{tid}$) to A, requesting the finalization of the payment. Upon receiving (PAY-OK, id) from A, $\mathcal{F}$ updates Γ(id) with new γ and ω, and sends (PAID, id, $\overrightarrow{\theta}$) to both users. Otherwise, if no PAY-OK is received, $\mathcal{F}$ executes the subprocedure ForceClose, which expects the funding transaction to be spent within Δ rounds. The channel state is updated only after both parties have explicitly agreed on the new payment: the payee confirms the revocation of the previous state by sending REVOKE, and the payer finalizes the update by responding with PAY-OK. Only then is the new state recorded as the latest consensus-approved state, ensuring *balance security in update*.

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

Punish. Triggered at the end of each round, this phase enforces accountability and resolves disputes. For each active channel, $\mathcal{F}$ checks if tx' appears on the ledger $\mathbb{B}$ that spends the funding transaction tx_F. If tx' matches the latest state γ.st, $\mathcal{F}$ finalizes the channel by setting $\Gamma(\text{id}) := \perp$ and sending (CLOSED, id) to both users, indicating 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 γ.st but aligns with a permitted previous state in ω, $\mathcal{F}$ expects a follow-up transaction tx_{res} to appear that redistribute balances according to γ.st. Upon observing this resolve transaction, $\mathcal{F}$ updates $\Gamma(\text{id}) := \perp$, then sends (RESOLVED, id) and (CLOSED, id) to both users. This case indicates a forced closure 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. Across all finalization paths, for an honest participant, the channel will never finalize in a state that allocates less than the balance specified in the latest consensus-approved state γ.st. Thus, *balance security in finalization* satisfies.

6.3 Details of ULTRAVIOLET

We provide a detailed exposition of the ULTRAVIOLET protocol built on the atomic paradigm. The complete workflow is depicted in Figure 5. The formal protocol modelled in UC framework can be found in Appendix A.2.

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, reflecting the fund distribution following that payment.

Creation. To establish the channel, both parties contribute funds via individual transactions tx_{F_A} and tx_{F_B}, which are combined into a shared funding transaction tx_F with total capacity $v = v^A + v^B$. Before broadcasting tx_F, they sign and exchange refund transactions that allow either party to reclaim their initial balances v^A and v^B if needed. These refund transactions are effectively equivalent to zero-value payments executed between the parties prior to channel creation,following the same format as update-phase payments. For better readability, we assume each party executes two such refund transactions in advance, ensuring they 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 update begins with revocation. The payee B generates a revocation pair (rh_{B_i}, rs_{B_i}), sends rh_{B_i} along with the previous secret $rs_{B_{i-2}}$ to the payer A to revoke state A_{i-2}.

Upon receiving $rs_{B_{i-2}}$ and rh_{B_i}, A generates the i-th payment transaction $tx_{A_i} := \text{tx}(tx_F, [\langle sk_A, v^A_{A_i}\rangle, \langle (rs_{B_i} \wedge sk_A) \vee (sk_A \wedge sk_B) \vee (\mathsf{RelTime}(T) \wedge sk_B), v^B_{A_i}\rangle])$, reflecting the new 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: $v^A_{A_i}$, which A can claim immediately upon confirmation, and $v^B_{A_i}$, which can be spent by different conditional branches. By default, B must 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 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. 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 tx_{B_j} and $tx_{B_{j+1}}$ that originally belong to

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

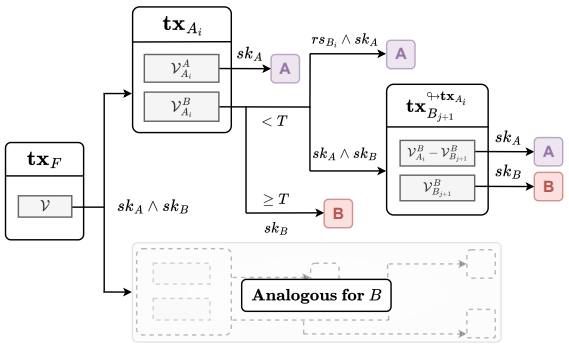
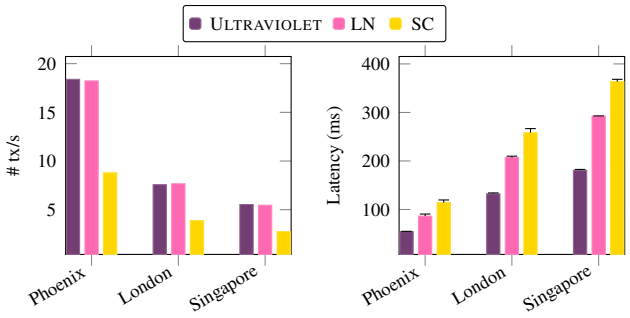


Figure 6: Flow for payment and resolve transaction.



(a) Throughput (b) Latency (95th percentile)

Figure 7: Comparison of average throughput and latency across regions, with latency including 95th percentile error.

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 $tx(tx_{B_{j-1}}, [\langle sk_A, v^A_{B_{j-1}}\rangle, \langle sk_B, 0\rangle])$.

Finalization. Similar to Lightning-style channels, the channel can be finalized collaboratively with a single transaction 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.

As the payee in the last payment, B can forcibly finalize the channel by posting tx_{A_i}, which reflects the globally latest state A_i. Upon confirmation, A immediately receives $v^A_{A_i}$,while B will receive $V^B_{A_i}$ after waiting for the relative time T.

As the payer in the last payment, party A can forcibly finalize the channel by posting the latest valid transaction tx_{B_j} they hold. Upon confirmation, B immediately receives $v^B_{B_j}$. 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 avoid loss, ensuring the channel ultimately finalizes in the latest consensus-approved 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, thereby claiming the entire funds of the channel.

7 Evaluation

In this section, we conduct a comprehensive evaluation of ULTRAVIOLET, comparing it against two SOTA PCs: LN and SC. The evaluation covers two key aspects: off-chain payment performance (Section 7.1) and on-chain cost (Section 7.2). To ensure a fair comparison, all three protocols are implemented using a consistent technology stack: Golang, gRPC for network communication, 256-bit ECDSA for digital signatures, SHA256 for hashing, `OP_CHECKMULTISIG` for multisignature support. The implementation consists of approximately 8,000 lines of Go code.

7.1 Off-chain Payment Performance

Experimental Setup. We evaluate performance by executing 10,000 sequential payments—where each payment begins only after the previous one completes—repeated five times per protocol. For each run, we measure payment latency, defined as the time from transaction initiation to completion, and report the average and 95th-percentile latency. We also compute the achieved transactions per second (TPS).

Experiments are conducted on Azure D8s_v4 virtual machines distributed across four regions: West US (Virginia), East US (Phoenix), South UK (London), and Southeast Asia (Singapore). Each VM is equipped with 8 vCPUs (Intel Xeon Platinum 8272CL @ 2.60GHz), 32 GB RAM, and 12.5 Gbps bandwidth, running Ubuntu 24.04 LTS. The node in Virginia acts as the payer, sending transactions to payee nodes in Phoenix, London, and Singapore.

Latency. Figure 7b shows the average transaction latency to different regions. ULTRAVIOLET consistently outperforms other protocols, achieving average latencies of 54.38ms (Phoenix), 132.11ms (London), and 180.84ms (Singapore)—representing 1.59× and 2.10× improvements over LN and SC, respectively. The P_{95} error bars indicate high stability, with ULTRAVIOLET exhibiting minimal variance ($\leq$1.96ms) compared to LN ($\leq$4.18ms) and SC ($\leq$ 8.37ms).

Table 1: Comparison of the number of transactions and on-chain bytes for unilateral closure on different PC protocols.

PC Protocol	Case	# tx	bytes
LN	close	1	397
	close(punished)	2	633
SC	close(slow)	2	931
	close(fast)	3	1501
	close(punished)	3	1239
ULTRAVIOLET	close	1	472
	close(resolved)	2	911
	close(punished)	2	794

Table 2: Comparison of different payment channel protocols.

PC	Atomicity	Flexibility	Compatibility
LN [1]	✗	✓	✓
SC [7]	✗	✗	✓
Eltoo [13]	✗	✓	✗
GC [4]	✗	✓	✗
ULTRAVIOLET	✓	✓	✓

The increased latency from Phoenix to Singapore reflects the dominant impact of network latency on transaction times.

Throughput. Figure 7a 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 (8.79 TPS, 3.87 TPS, 2.75 TPS) by approximately 2×.

7.2 On-chain Cost

As shown in Table 1, under unilateral closure where the initiator holds the latest state, both ULTRAVIOLET and LN require a single transaction (472 bytes and 397 bytes), while SC incurs higher overhead with two transactions totaling 931 bytes. If ULTRAVIOLET needs to perform a resolve transaction, it requires two transactions totaling 911 bytes — comparable to the slow closure in SC but still lower than its fast closure. In the case of punishment, ULTRAVIOLET incurs 794 bytes across two transactions, slightly more than LN (633 bytes) but significantly less than SC (1239 bytes). Overall, ULTRAVIOLET maintains competitive cost across all closure cases.

8 Analysis and Discussion

In this section, we analyze our protocol's characteristics, covering its communication and storage overheads. We also examine its compatibility, deployment considerations, and limitations, and conclude with a discussion of potential threats.

Efficiency in Communication and Computation. ULTRAVIOLET achieves low communication and computation overhead by requiring only two messages per payment (half of LN and SC) and relying solely on standard cryptographic primitives. Halving the number of messages lower the communication cost, while the avoidance of complex cryptography ensures lightweight processing. These advantages are reflected in the improved latency and throughput observed in our evaluation.

Storage Overhead. The resolve mechanism in ULTRAVIOLET requires each party to store at most two resolve transactions at any given time, each approximately 440KB. This added storage cost is minimal and practically negligible.

Flexibility and Deployment Compatibility. As summarized in Table 2, ULTRAVIOLET is the first and only payment channel protocol that achieves atomicity. In addition, it offers strong flexibility by supporting bidirectional payments, unrestricted channel lifetime, and unbounded transactions—allowing channels to remain open indefinitely and process an unlimited number of updates without reset. Among the compared protocols, only SC lacks such flexibility, as it imposes a fixed expiration on channel lifespan.

For deployment compatibility, ULTRAVIOLET relies solely on Bitcoin's native scripting and avoids non-standard cryptographic features. Unlike Eltoo, which depends on unimplemented opcodes (e.g., `SIGHASH_NOINPUT`), or GC, which tightly couple with specific signature schemes ECDSA or Schnorr, ULTRAVIOLET remains fully compatible with Bitcoin's existing functionality and can be readily deployed. However, it is not directly compatible with LN channels.

Limitations. ULTRAVIOLET requires participants to continuously monitor the blockchain or delegate this responsibility to a trusted third-party service such as watchtowers. This monitoring is necessary to either respond with a resolve transaction if the counterparty initiates a valid unilateral closure, or to issue a punishment transaction if an outdated state is broadcast. Such monitoring requirements are common across many existing protocols like LN. In contrast, protocols like SC adopt a watchtower-free design, avoiding this requirement through incentive mechanisms and time-bound constraints.

Potential Threats. We discuss three main potential threats:

- *DoS Attacks.* ULTRAVIOLET is resilient to protocol-layer DoS attacks caused by an unresponsive counterparty. In such cases, participants can unilaterally close the channel while preserving balance security. More severe network-layer DoS attacks that render parties unreachable to each other or the blockchain for extended periods, are analogous to network partitioning scenarios.
- *Network Partition.* If a network partition prevents communication between participants while both retain access to the blockchain, ULTRAVIOLET allows either party to securely close the channel unilaterally. However, if a participant is isolated from both their counterparty and the blockchain, they cannot monitor on-chain activity or respond with necessary transactions. In this case, a malicious counterparty could finalize the channel using an outdated state without being penalized. This threat

can be mitigated by employing watchtower services.

- *Privacy Concerns.* ULTRAVIOLET ensures that off-chain transactions remain hidden from the blockchain until the channel is closed. Under cooperative or standard unilateral closure, only the initial and final fund allocation states are revealed on-chain, providing privacy guarantees similar to LN. However, in closure cases requiring a resolve transaction, ULTRAVIOLET may additionally leak the intermediate state of a specific off-chain transaction, introducing a marginal trade-off compared to LN.

9 Conclusion

In this paper, we formalize existing payment channel protocols, such as the Lightning Network and Sleepy Channels, into a common paradigm and show that their non-atomic state transitions introduce fundamental vulnerabilities. To overcome this, we propose an atomic paradigm that ensures atomic updates while preserving core functionalities. Based on this paradigm, we design ULTRAVIOLET, a secure and efficient payment channel protocol that avoids reliance on unimplemented Bitcoin features. By halving the number of messages per transaction, ULTRAVIOLET reduces latency by up to 52% and achieves throughput comparable to the Lightning Network and 2x that of the Sleepy Channels. Our formal security analysis under the Universal Composability framework and experimental evaluation demonstrate that ULTRAVIOLET offers a practical and secure path for blockchain scalability.

10 Acknowledgements

This work is funded by National Key Research and Development Program of China (2023YFB2704000). The authors thank anonymous reviewers for their constructive comments and guidance, and thank Xinyu Zhang for the helpful discussion.

11 Ethics Considerations

Our research builds upon a previously disclosed design flaw in the Lightning Network that was formally identified and published at CCS'24 [28]. The authors of that work confirmed they had properly disclosed the vulnerability to the Lightning Network development team prior to publication. Our work proposes a protocol-level solution to address this design flaw.

We have carefully considered the ethical implications of our research through the lens of both consequentialist and deontological frameworks. From a beneficence perspective, our work aims to strengthen the Lightning Network's security by providing concrete improvements to its protocol design. The potential benefits to users and the broader cryptocurrency ecosystem outweigh any potential risks, particularly since the underlying vulnerability is already public knowledge.

Our research focuses on developing defensive measures rather than exploiting vulnerabilities, and builds upon properly disclosed findings. Therefore, we believe our work aligns with the principles outlined in The Menlo Report, particularly regarding respect for persons and promoting beneficence. Our protocol improvements protect users' financial interests and their right to secure transaction channels, supporting both stakeholder wellbeing and fundamental rights to security in financial transactions.

12 Open science

To ensure reproducibility and facilitate further research in this area, we have made our implementation of ULTRAVIOLET available as a comprehensive source code package. This package can be accessed via Zenodo [1] or Github [2]. It contains all necessary components to validate our protocol's effectiveness and reproduce our experimental results. We are committed to making the package publicly accessible upon publication.

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] 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.

[5] 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.

[6] 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.

[1] https://doi.org/10.5281/zenodo.15559635
[2] https://github.com/zju-abclab/ultraviolet

[7] Lukas Aumayr, Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, Pedro Moreno-Sanchez, and Matteo Maffei. Sleepy channels: Bi-directional payment channels without watchtowers. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 179–192, 2022.

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

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

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

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

[22] Bowen Liu, Pawel Szalachowski, and Siwei Sun. Fail-safe 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

[29] Jingjing Zhang, Yongjie Ye, Weigang Wu, and Xiapu Luo. Boros: Secure and efficient off-blockchain transactions via payment channel hub. *IEEE Transactions on Dependable and Secure Computing*, 20(1):407–421, 2021.

A Extended UC Modeling

In this section, we formalize the security of our protocol using the Universal Composability (UC) framework [10], specifically employing the Global UC (GUC) extension [11], 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 [4–6, 15, 16, 18], 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 $\mathcal{F}_{clock}$ [21], 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 $\mathcal{F}_{GDC}$ [15]. 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 [4, 9]. 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 Protocol

In this section, we formally present the protocol Π as outlined in Section 6.3. 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 ($\texttt{CREATE}, \gamma, tid_A$) $\overset{\tau_0}{\hookleftarrow} \mathcal{E}$:

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, $tid_A, auth^A_{set}$) $\overset{\tau_0}{\hookrightarrow} B$.
4. If (createInfo, id, $tid_B, auth^B_{set}$) $\overset{\tau_0+1}{\hookleftarrow} B$, $auth^A_{set} := auth^A_{set} \cup auth^B_{set}$, 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}) $\overset{\tau_0+1+\tau_{pre}}{\hookrightarrow} B$.
11. If (prepareInfo, id, tx^B_{set}, sig^B_{set}) $\overset{\tau_0+2+\tau_{pre}}{\hookleftarrow} B$, 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}) $\overset{\tau_0+2+\tau_{pre}}{\hookrightarrow} B$.
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 $\tau_1 \le \tau_0 + 3 + \tau_{pre} + \Delta$, set $\Gamma^A(\text{id}) := (tx_F, auth^A_{set}, rs^A_{set}, tx^A_{set}, sig^A_{set})$, ($\texttt{CREATED}$, id) $\overset{\tau_1}{\hookrightarrow} \mathcal{E}$.

Pay

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

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

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

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

1. Send ($\texttt{REVOKE-REQ}$, id, $\overrightarrow{\theta}, t_{\text{stp}}$) $\overset{t_0}{\hookrightarrow} \mathcal{E}$
2. If not ($\texttt{REVOKE}$, id) $\overset{t_0}{\hookleftarrow} \mathcal{E}$, 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, $rh_{B_i}, rs_{B_{i-2}}$) $\overset{t_0+t_g}{\hookrightarrow} A$.

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

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 $tx^{A_i}_{pay} \leftarrow \texttt{GenerateTxs}(tx_F, tx^A_{set}, auth^A_{set}, v^A_{A_i}, v^B_{A_i})$, $tx^A_{set} := tx^A_{set} \cup tx^{A_i}_{pay}$.

8. A uses sk_A to sign each transaction in $tx_{pay}^{A_i}$, generating a set of signatures $\sigma_{pay}^{A_i}$.
9. Let $\overrightarrow{tid}$ be the tuple of ids corresponding to each transaction in $tx_{pay}^{A_i}$, (PAY-REQ, id, $\overrightarrow{tid}$) $\xhookrightarrow{\tau_1 \le \tau_0+2+t_g+t_{\text{stp}}} \mathcal{F}$.
10. Send (payInfo, id, $tx_{pay}^{A_i}, \sigma_{pay}^{A_i}$) $\xhookrightarrow{\tau_1 \le \tau_0+2+t_g+t_{\text{stp}}} B$.

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

1. If (payInfo, id, $tx_{pay}^{A_i}$, $\sigma_{pay}^{A_i}$) $\xhookleftarrow{t_1} A$, go to next step. Else, execute ForceClose(id).
2. Let $tx_{set}^B := tx_{set}^B \cup tx_{pay}^{A_i}, sig_{set}^B := sig_{set}^B \cup \sigma_{pay}^{A_i}$.
3. Let $\Gamma^B(\text{id}) := (tx_F, auth_{set}^B, rs_{set}^B, tx_{set}^B, sig_{set}^B)$.
4. Send (payCom, id) $\xhookrightarrow{t_1} A$ and (PAID, id, $\overrightarrow{\theta}$) $\xhookrightarrow{t_1} \mathcal{F}$.

Party A in round $\tau_1 + 2$:

1. If (payCom, id) $\xhookleftarrow{\tau_1+2} B$, continue. Else, execute ForceClose(id).
2. If not (PAY-OK, id) $\xhookleftarrow{\tau_1+2} \mathcal{F}$, go idle.
3. Let $\Gamma^A(\text{id}) := (tx_F, auth_{set}^A, rs_{set}^A, tx_{set}^A, sig_{set}^A)$.
4. (PAID, id, $\overrightarrow{\theta}$) $\xhookrightarrow{\tau_1+2} \mathcal{F}$.

Close

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

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

1. Extract $(tx_F, auth_{set}^A, rs_{set}^A, tx_{set}^A, sig_{set}^A) = \Gamma^A(\text{id})$.
2. Retrieve tx_{A_n} from tx_{set}^A and extract $v_{A_n}^A$ and $v_{A_n}^B$ from tx_{A_n}.
3. Generate transaction $tx_C := tx(tx_F, [\langle sk_A, v_{A_n}^A \rangle, \langle sk_B, v_{A_n}^B \rangle])$.
4. A uses sk_A to generate signature σ_C^A for transaction tx_C.
5. Send (closeInfo, id, tx_C, σ_C^A) $\xhookrightarrow{\tau_0+\tau_g} B$.
6. If (closeInfo, id, tx_C, σ_C^B) $\xhookleftarrow{\tau_0+\tau_g+1} B$, post ($tx_C$, $\{\sigma_C^A, \sigma_C^B\}$) on $\mathbb{B}$. Else, go idle.
7. If tx_C is confirmed on $\mathbb{B}$ in round $\tau_1 \le \tau_0 + \tau_g + 1 + \Delta$, set $\Theta^A(\text{id}) := \bot$, $\Gamma^A(\text{id}) := \bot$ and (CLOSED, id) $\xhookrightarrow{\tau_1} \mathcal{F}$. Else, execute ForceClose(id).

Punish

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

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

1. Extract $(\mathcal{T}_{revoke}^A, \mathcal{T}_{pnsh}^A, \Sigma_{pnsh}^A) = \Theta^A(\text{id})$.
2. If any revoked payment $tx_{A_n} \in \mathcal{T}_{revoke}^A$ appears on $\mathbb{B}$, retrieve $tx_{pnsh}^{A_n}$, $\sigma_{pnsh}^{A_n}$ from $\mathcal{T}_{pnsh}^A$ and Σ_{pnsh}^A respectively. Post ($tx_{pnsh}^{A_n}$, $\sigma_{pnsh}^{A_n}$) on $\mathbb{B}$.
3. After $tx_{pnsh}^{A_n}$ is confirmed on $\mathbb{B}$ in round $\tau_1 \le \tau_0 + \Delta$, set $\Theta^A(\text{id}) := \bot$, $\Gamma^A(\text{id}) := \bot$ and (PUNISHED, id) $\xhookrightarrow{\tau_1} \mathcal{F}$.

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 $(tx_F, auth_{set}^P, rs_{set}^P, tx_{set}^P, sig_{set}^P) = \Gamma^P(\text{id})$.
2. Retrieve tx_{Q_j}, $\sigma_{Q_j}^Q$ from tx_{set}^P and sig_{set}^P respectively.
3. P uses sk_P to generate signature $\sigma_{Q_j}^P$ for transaction tx_{Q_j}. Post (tx_{Q_j}, $\{(\sigma_{Q_j}^P, \sigma_{Q_j}^Q)\}$) on $\mathbb{B}$.
4. After tx_{Q_j} is confirmed on $\mathbb{B}$ in round $\tau_1 \le \tau_0 + \Delta$, set $\Theta^P(\text{id}) := \bot$, $\Gamma^P(\text{id}) := \bot$. (forceClose, id, tx_{Q_j}) $\xhookrightarrow{\tau_1} Q$ and (CLOSED, id) $\xhookrightarrow{\tau_1} \mathcal{F}$.

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

1. After tx_{Q_j} appears on $\mathbb{B}$, extract (tx_F, $auth_{set}^Q$, rs_{set}^Q, tx_{set}^Q, sig_{set}^Q) $= \Gamma^Q(\text{id})$.
2. If there exists $tx_{P_i}^{\curvearrowright tx_{Q_j}} \in tx_{set}^Q$, continue. Else, (CLOSED, id) $\xhookrightarrow{t_0} \mathcal{F}$ and stop.
3. Retrieve $tx_{P_i}^{\curvearrowright tx_{Q_j}}$, $\sigma_{P_i(Q_j)}^P$ from tx_{set}^Q and sig_{set}^Q respectively.
4. Q uses sk_Q to generate signature $\sigma_{P_i(Q_j)}^Q$ for transaction $tx_{P_i}^{\curvearrowright tx_{Q_j}}$. Post ($tx_{P_i}^{\curvearrowright tx_{Q_j}}$, $\{\sigma_{P_i(Q_j)}^P, \sigma_{P_i(Q_j)}^Q\}$) 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) $\xhookrightarrow{t_0+\Delta} \mathcal{F}$ and (CLOSED, id) $\xhookrightarrow{t_0+\Delta} \mathcal{F}$.

GenerateTxs(tx_F, tx_{set}^P, $auth_{set}^P$, $v_{P_i}^P$, $v_{P_i}^Q$):

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 $auth_{set}^P$.
2. Generate the payer P's i-th payment transaction tx_{P_i} := tx(tx_F, $[\langle sk_P, v_{P_i}^P \rangle, \langle (rs_{Q_i} \wedge sk_P) \vee (sk_P \wedge sk_Q) \vee (\mathsf{RelTime}(T) \wedge sk_Q), v_{P_i}^Q \rangle]$).
3. Retrieve $\{tx_{Q_j}, tx_{Q_{j-1}}\}$ from tx_{set}^P, extract $v_{Q_j}^P$ and $v_{Q_{j-1}}^P$ respectively.
4. Let $v_{res_1}^P := v_{P_i}^P$, $v_{res_1}^Q := v_{Q_j}^P - v_{P_i}^P$.
5. Generate resolve transaction $tx_{P_i}^{\curvearrowright tx_{Q_j}} := tx(tx_{Q_j}, [\langle sk_P, v_{res_1}^P \rangle, \langle sk_Q, v_{res_1}^Q \rangle])$.
6. If $v_{P_i}^P \le v_{Q_{j-1}}^P$, set $v_{res_2}^P := v_{P_i}^P$, $v_{res_2}^Q := v_{Q_{j-1}}^P - v_{P_i}^P$. Else, set $v_{res_2}^P := v_{Q_{j-1}}^P$, $v_{res_2}^Q := 0$.

7. Generate resolve transaction $tx_{P_i}^{\not\hookrightarrow tx_{Q_{j-1}}} := tx(tx_{Q_{j-1}}, [\langle sk_P, v_{res_2}^P \rangle, \langle sk_Q, v_{res_2}^Q \rangle])$.
8. Return $tx_{pay}^{A_i} := \{tx_{P_i}, tx_{P_i}^{\not\hookrightarrow tx_{Q_j}}, tx_{P_i}^{\not\hookrightarrow tx_{Q_{j-1}}}\}$.

A.3 Proof

We present the simulator and the formal proof that the ULTRAVIOLET protocol Π, described in Appendix A.2, GUC-realizes the ideal functionality $\mathcal{F}$, defined in Section 6.2.

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

Due to space constraints, the complete proof is provided in the extended version [3] of this work.

[3] https://eprint.iacr.org/2025/180.pdf