SightCVC: An Efficient and Compatible Multi-Chain Transaction Protocol in Heterogeneous Blockchain Systems

Haonan Yang, *Student Member, IEEE*, Tianwei Zhang, *Member, IEEE*, Zuobin Ying, *Member, IEEE*, Runjie Yang, *Student Member, IEEE*, and Wanlei Zhou, *Senior Member, IEEE*

***Abstract*—With the popularity of cross-chain transactions in heterogeneous blockchain systems, scalability has become a critical challenge. To overcome this, researchers propose to establish *virtual channels*, which move cross-chain transactions off the blockchain, enabling instant transaction confirmation between users and improving the system throughput. However, existing off-chain cross-chain transaction schemes encounter the following issues: (i) they are incompatible with non-Turing complete blockchain systems; (ii) they are incapable of accessing authentic information from blockchain systems. These issues have a dual impact on the cross-chain transaction, affecting its compatibility and dispute resolutions among mutually distrustful users. To alleviate these issues, this paper introduces SightCVC, a novel cross-chain payment protocol. The core of SightCVC is a new smart contract, which facilitates unrestricted off-chain transactions among mutually distrustful users in heterogeneous blockchain systems. It only requires off-chain protocol of the blockchain system involved in the transactions to support a Turing complete scripting language, which resolves the compatibility issue. Meanwhile, it can securely retrieve the real information from the blockchain systems, significantly improving the effectiveness of dispute resolution and enforcing the privacy of cross-chain transactions. We conduct a thorough security analysis within the Universal Composability framework to validate that SightCVC can achieve consensus at each stage. We implement and deploy SightCVC on the experimental networks of Ripple and Ethereum. Comprehensive evaluations demonstrate that SightCVC is able to effectively handle the disputes and reduce the system costs by approximately 64% compared to existing solutions. Its superiority becomes more evidence when the number of transactions increases.**

***Index Terms*—Scalability, cross-chain, incompatible, dispute resolutions.**

Received 1 April 2025; revised 18 July 2025; accepted 21 August 2025. Date of publication 8 September 2025; date of current version 1 October 2025. This work was supported in part by Joint scientific research funding from the Macau Science and Technology Development Fund and the National Natural Science Foundation of China (FDCT-NSFC), China and Macau, under Grant 0051/2022/AFJ; and in part by Nanyang Technological University Centre in Computational Technologies for Finance (NTU-CCTF). The associate editor coordinating the review of this article and approving it for publication was Prof. Haibo Hu. *(Corresponding author: Zuobin Ying.)*

Haonan Yang, Zuobin Ying, Runjie Yang, and Wanlei Zhou are with the Faculty of Data Science, City University of Macau, Macau, China (e-mail: D23092100105@cityu.edu.mo; zbying@cityu.edu.mo; D23092100349@cityu.edu.mo; wlzhou@cityu.edu.mo).

Tianwei Zhang is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: tianwei.zhang@ntu.edu.sg).

Digital Object Identifier 10.1109/TIFS.2025.3607247

I. Introduction

OVER the years, the blockchain ecosystem has evolved into a diverse landscape encompassing a wide range of applications [1], [2], [3].One single blockchain gradually exhibits incapability of accommodating such diverse demands. Therefore, it becomes more popular to develop heterogeneous blockchain ecosystems, which incorporate blockchains with distinct technologies, security guarantees and performance [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]. Such multi-chain coexistence paradigm allows developers and users to flexibly select the most suitable chains based on their actual pReferences and budgets, paving the way for more robust and versatile blockchain applications.

In these multi-chain coexistence systems, it is important to coordinate different blockchains with diverse features. Currently, the mainstream solution is to introduce a Trusted Third Party (TTP). While such centralized method can facilitate real-time transactions, it introduces security risks associated with the single points of failure. For example, Mt. Gox, a famous Bitcoin exchange, ultimately declared bankruptcy after hackers stole a large number of bitcoins from its centralized system [14], which acted as a TTP.

To address this security concern, some multi-chain coexistence solutions have shifted the coordination responsibilities between blockchain systems from the centralized to decentralized manner. For instance, Herlihy [9] designed Atomic Cross-Chain Swaps (ACCS), which employs the Hashlock and Timelock techniques to facilitate cross-chain transactions without the need of a TTP. Malavolta et al. [10] designed Anonymous Multi-Hop Lock (AMHL) to maintain the security of cross-chain transactions by limiting the intermediate nodes to only the participation of necessary steps such as routing and communication. Although these decentralized systems can well reduce the failure of cross-chain coordination, they still bring new performance and privacy issues. (1) The complex coordination process can significantly decrease the overall system performance. (2) The introduced intermediate nodes need to be constantly online to participate in every transaction, greatly limiting the efficiency of the multi-chain system. (3) The intermediate nodes need to acquire the information about each chain in order to support multi-chain transactions, which may potentially compromise the transaction privacy.

Recently, Jia et al. [15] proposed a novel cross-chain transaction protocol, Cross-Chain Virtual Channel (CVC). CVC is built upon Virtual Channel (VC) [16], [17], which is an efficient mechanism to facilitate micropayments. In contrast to traditional blockchain transactions, VC allows an unlimited number of off-chain transactions between parties without the involvement of the blockchain or intermediary nodes. Then the core idea of CVC is to establish a VC between the transaction sender and receiver through an intermediary node across two blockchain systems. This enables two users to independently and efficiently execute any number of off-chain transactions, while the intermediary node only needs to participate during the establishment and closure of the VC. This design effectively migrates cross-chain transactions off-chain, significantly improving the overall throughput and performance of the system. Additionally, since the intermediary node does not participate in the specific transaction process, it also reduces the risk of transaction privacy leakage.

To ensure the security of user deposits, CVC requires that the two blockchain systems perform different operations based on the state of the other party. This functionality is not naturally available in blockchain systems and necessitates the adoption of an oracle [18], [19], [20] to achieve this. Specifically, the oracle is responsible for forwarding the state change information from the sender chain to the receiver chain. Correspondingly, it also manages the initiation and termination of transactions across the two chains. The design of CVC, unfortunately, does not prioritize oracle, but rather simply indicates that oracle is deployed on the Ethereum smart contract. After careful observation, we have identified the challenges in designing smart contracts with oracle functionality, for the following reasons: (1) If oracle utilizes the state of CVC to perform operations between two blockchain systems, it will be involved in every transaction of CVC, thereby compromising the transaction privacy. This contradicts the design concept of virtual channels [21]. (2) If oracle utilizes the state of the blockchain and its applications to execute operations between two blockchain systems, the presence of CVC becomes redundant, but also incurring additional costs for its establishment.

To address the deficiencies and challenges in the design of oracle smart contracts in CVC, we propose `SightCVC`, an efficient cross-chain protocol aimed at protecting the privacy of cross-chain transactions. Our protocol makes three key contributions. Firstly, within the context of CVC, we use *verifiable witness encryption* as the primary cryptographic building block to design a novel oracle smart contract. This contract relays the state of the blockchain through intermediate nodes without involving the transactions themselves. Specifically, the transaction sender provides the information and signature of a successful interaction with the smart contract to the receiver. The receiver then sends the transaction signature and the sender's successful interaction information to the oracle smart contract to generate the ciphertext. The condition for unlocking the ciphertext is that the intermediate node must provide the sender's successful interaction information with the oracle smart contract. Once the intermediate node unlocks the ciphertext, it obtains the transaction signature, completing the transaction. During this process, the blockchain state is transmitted by conveying the information of successful interactions to the smart contract. It is important to note that the interaction information within the smart contract does not include the transaction content itself.

Secondly, we redesign the CVC protocol based on the new oracle smart contract. Our revised protocol integrates an oracle smart contract with two virtual channels from different blockchain systems. Cross-chain transactions occur within the CVC, aiming to transmit the state of virtual channels to the smart contract through an intermediary node. The new protocol ensures transaction privacy within the CVC and is compatible with any blockchain that supports smart contracts [16], [22]. Additionally, users do not need to engage directly with the blockchain, intermediary nodes, or oracle during transactions, preserving the original cross-chain throughput capacity of the CVC [15]. Furthermore, deploying oracle within smart contracts allows the integration of external data and the triggering of automated execution. This enhancement increases the functionality, credibility, and transparency of the contract, meeting the complex requirements of the CVC.

Thirdly, we implement and deploy `SightCVC` on Ripple and Ethereum. We simulate transactions across these platforms to evaluate key metrics such as transaction delay, fee costs, oracle overhead, and success ratio. The results show that `SightCVC` achieves lower latency by leveraging virtual channels, effectively avoiding the complexities found in other protocols. Fee costs, primarily in terms of Gas usage during different phases of `SightCVC` operations, exhibit minimal overhead, particularly during update operations. Oracle overhead remains acceptable, even as exchange rates increase. However, the success ratio of `SightCVC` decreases as the proportion of cross-chain virtual channels grows, largely due to the increased locking of funds within the network.

The remainder of this paper is organized as follows: we describe the background of payment channel, payment channel network, virtual channel and cross-chain systems in Section II. Section III presents the overview of `SightCVC` and the design goals. In Section IV, we show the detailed protocol design. Following this, Section V gives the security analysis. We evaluate the performance of `SightCVC` in Section VI. Section VII reviews related work. Limitations are discussed in Section VIII, and Section IX concludes the paper.

II. Background

We describe the background related to cross-chain blockchain systems. Without loss of generality, we consider a transaction between two users: Alice and Bob. We also introduce Ingrid as an intermediary user.

A. Payment Channel Network

A Payment Channel Network (PCN) is established with multiple interconnected payment channels [22], [23] to enable transactions between a sender and receiver who are not directly connected [24]. For instance, Alice and Bob, who do not share a direct payment channel, can transact through an intermediary, Ingrid. Alice and Ingrid establish a channel α and deposit

$\alpha.x_A$ and $\alpha.x_I$ coins into it, respectively. The initial state of α is represented by the function $[\text{Alice} \mapsto \alpha.x_A, \text{Ingrid} \mapsto \alpha.x_I]$. Similarly, Ingrid and Bob establish another channel β and deposit $\beta.x_I$ and $\beta.x_B$ coins into it, respectively. The initial state of channel β is represented by the function $[\text{Ingrid} \mapsto \beta.x_I, \text{Bob} \mapsto \beta.x_B]$. These functions denote the coin balances of each party within the respective channels, with the total balances of $\alpha.x_A + \alpha.x_I$ for α and $\beta.x_I + \beta.x_B$ for β.

Alice can pay Bob q coins (where $q \leq \alpha.x_A$) by sending her coins to Ingrid via channel α, who then forwards them to Bob via channel β. The protocol must ensure atomicity, i.e., the transfers from Alice to Ingrid and from Ingrid to Bob either both occur or neither occur. The transfer can be represented with the following functions simultaneously:

$$[\text{Alice} \mapsto \alpha.x'_A, \text{Ingrid} \mapsto \alpha.x'_I]$$

$$[\text{Ingrid} \mapsto \beta.x'_I, \text{Bob} \mapsto \beta.x'_B]$$

where $\alpha.x'_A = \alpha.x_A - q, \alpha.x'_I = \alpha.x_I + q, \beta.x'_I = \beta.x_I - q, \beta.x'_B = \beta.x_B + q$.

However, PCNs suffer from several drawbacks: (1) low reliability: the success of payments relies on Ingrid's availability; (2) high latency: each payment must be routed through Ingrid; (3) high costs: Ingrid may charge fees for each payment between Alice and Bob; (4) low privacy: Ingrid can observe every transaction between Alice and Bob.

B. Virtual Channels

To address the limitations of PCNs, researchers introduced the concept of VCs [16]. A VC significantly reduces the number of interactions with the intermediary Ingrid, particularly by eliminating the need to confirm individual payments routed through her. The core idea is the recursive applications of payment channel techniques, establishing new VCs on top of existing payment channels. To better elucidate this concept, we assume that the initial state of a VC is a PCN.

Alice and Bob want to *open* a VC, denoted as γ, with an initial balance of $[\text{Alice} \mapsto \gamma.x_A, \text{Bob} \mapsto \gamma.x_B]$. This process is completed through channels α and β without interacting with the underlying blockchain. During the opening of γ, a portion of the coins in the account of each party in the payment channels α and β are temporarily locked. Specifically, after opening γ, the balance changes in α and β are as follows: in channel α, Alice locks $\gamma.x_A$ coins from her account, and Ingrid locks $\gamma.x_B$ coins from her account; similarly in channel β, Ingrid locks $\gamma.x_A$ coins, and Bob locks $\gamma.x_B$ coins.

The VC γ can be successfully opened only when the following conditions are satisfied: $\gamma.x_A \leq min(\alpha.x_A, \beta.x_I)$ and $\gamma.x_B \leq min(\alpha.x_I, \beta.x_B)$, ensuring that all relevant values are non-negative. This implies that Alice, Bob, and Ingrid must possess sufficient coins in their respective payment channels to facilitate the opening of γ. Upon the opening of VC γ, the amounts $\gamma.x_A$ and $\gamma.x_B$ are deducted from each party's respective account in channels α and β.

Once the VC γ is opened, it can be updated multiple times, similar to a payment channel. Each update represents a transfer between Alice and Bob. In a scenario where all participants are honest, Alice and Bob only need to interact with Ingrid during the *open* and *close* procedures of the virtual channel, but not in each *update* of γ.

When closing a VC γ, the final transaction result is reflected on the payment channels α and β, but it does not directly affect the account balances of the parties on the blockchain. Let $[\text{Alice} \mapsto \gamma.x'_A, \text{Bob} \mapsto \gamma.x'_B]$ be the final transaction states of γ, assuming the states of α and β remain unchanged. Upon closing γ, the state of α becomes $[\text{Alice} \mapsto \alpha.x_A - \gamma.x_A + \gamma.x'_A, \text{Ingrid} \mapsto \alpha.x_I - \gamma.x_B + \gamma.x'_B]$, while the state of β becomes $[\text{Ingrid} \mapsto \beta.x_I - \gamma.x_A + \gamma.x'_A, \text{Bob} \mapsto \beta.x_B - \gamma.x_B + \gamma.x'_B]$.

For Alice, the net transaction result is that she gains $\gamma.x_A - \gamma.x'_A$ coins in her account in α.[1] Bob has an analogous guarantee. Conversely, Ingrid's result is neutral, implying that if she gains a coins in α, she will lose a coins in β. For instance, assume that the final state of γ is more advantageous to Alice than the initial state ($\gamma.x'_A > \gamma.x_A$). By agreeing to open a virtual channel, Ingrid essentially agrees to pay, in α, the amount that Bob transfers to Alice in γ.

III. Preliminaries and Overview

A. Preliminaries and Key Building Blocks

Definition 1: (Verifiable Witness Encryption Based on Threshold Signatures). A verifiable witness encryption scheme based on threshold signatures is a cryptographic primitive parameterized by $(\rho, N, M \in \mathbb{N})$, defined for two signature schemes $(\mathsf{DS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf}))$ and $(\overline{\mathsf{DS}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf}))$. It comprises three probabilistic polynomial-time (PPT) algorithms $((\mathsf{EncSig}, \mathsf{VfEnc}, \mathsf{DecSig}))$, defined as follows.

- $(c, \pi_c) \leftarrow \mathsf{EncSig}(((\overline{vk})_{i\in[N]}, (\overline{m}_j)_{j\in[M]}), sk, (m_j)_{j\in[M]})$: the signature-encryption algorithm takes as input tuples of verification key instances $(\overline{vk})_{i\in[N]}$, corresponding instance messages $(\overline{m}_j)_{j\in[M]}$, messages $(m_j)_{j\in[M]}$, and a signing key sk. It outputs a ciphertext c and a proof π_c.
- $0/1 \leftarrow \mathsf{VfEnc}(c, \pi_c, ((\overline{vk})_{i\in[N]}, (\overline{m}_j, m_j)_{j\in[M]}, vk))$: the encryption verification algorithm takes as input a ciphertext c, a proof π_c, tuples of instance verification keys $(\overline{vk})_{i\in[N]}$, instance messages $(\overline{m}_j)_{j\in[M]}$, and messages $(m_j)_{j\in[M]}$, and a verification key vk. It outputs 1 (for valid) if its a valid ciphertext and 0 (for invalid) otherwise.
- $\sigma \leftarrow \mathsf{DecSig}(j, \{\overline{\sigma}_i\}_{i\in K}, c, \pi_c)$: the signature-decryption algorithm takes as input an index $j \in [M]$, corresponding witness signatures $\{\overline{\sigma}_i\}_{i\in K}$ where $|K| = \rho$ and $K \subset [N]$, a ciphertext c, and a proof π_c. It outputs a signature σ.

Definition 2: (Oracle Contracts). Oracle Contracts is a protocol parameterized by $\rho, N, M \in \mathbb{N}$ (where $\lceil \frac{N}{2} \rceil \leq \rho \leq N$) and executed among the following entities: N oracles $\{\mathcal{O}_1, \ldots, \mathcal{O}_N\}$, and two users, Alice $\mathcal{A}$ (signing party) and Bob $\mathcal{B}$ (verifying party). The protocol is defined relative to the signature scheme $\Pi_{\mathsf{bds}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ of the transaction scheme on blockchain C, and comprises five probabilistic polynomial-time (PPT) algorithms: OKGen, Attest, $\mathsf{AttestVf}$, $\mathsf{Anticipate}$, AntiVf, Redeem, as detailed below.

- $(pk^{\mathcal{O}}, sk^{\mathcal{O}}) \leftarrow \mathsf{OKGen}(1^{\lambda})$: the oracle key generation algorithm takes as input the security parameter λ and

[1] Gaining x coins is equivalent to losing $-x$ coins if x is negative.

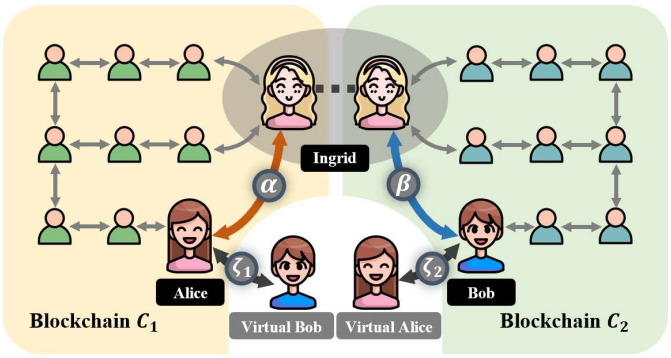

Fig. 1. Overview of SightCVC.

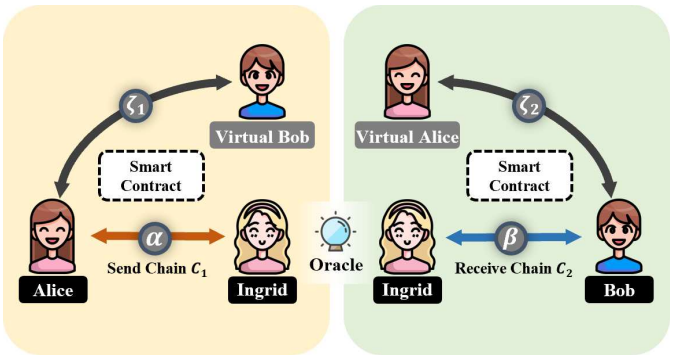

Fig. 2. SightCVC Protocol.

outputs the oracle public key $pk^{\mathcal{O}}$ and the corresponding oracle secret key $sk^{\mathcal{O}}$.

- $att \leftarrow \mathsf{Attest}(sk^{\mathcal{O}}, o)$: the event attestation algorithm takes as input oracle's secret key $sk^{\mathcal{O}}$, and the event outcome o, and outputs the outcome attestation att.
- $\{0,1\} \leftarrow \mathsf{AttestVf}(pk^{\mathcal{O}}, att, o)$: the attestation verification algorithm takes as input oracle's public key $pk^{\mathcal{O}}$, the outcome attestation att and the outcome o, and returns 1 if att attests to o being the outcome the event and 0 otherwise.
- $ant \leftarrow \mathsf{Anticipate}(sk_A, (pk_i^{\mathcal{O}})_{i\in[N]}, (o_j, Tx_j)_{j\in[M]})$: the attestation anticipation algorithm takes as input the signing party's secret key sk_A, oracles' public keys $(pk_i^{\mathcal{O}})_{i\in[N]}$, and tuples of outcomes and transactions $(o_j, Tx_j)_{j\in[M]}$, and outputs the anticipation ant.
- $\{0,1\} \leftarrow \mathsf{AntiVf}(pk_A, ant, (pk_i^{\mathcal{O}})_{i\in[N]}, (o_j, Tx_j)_{j\in[M]})$: the anticipation verification algorithm takes as inputs the signing party's public key pk_A, the anticipation ant, oracles' public keys $(pk_i^{\mathcal{O}})_{i\in[N]}$, and tuples of outcomes and transactions $(o_j, Tx_j)_{j\in[M]}$, and outputs 1 if ant is well formed and 0 otherwise.
- $\sigma \leftarrow \mathsf{Redeem}(j, (att_i)_{i\in[K]}, ant)$: the redeem algorithm takes as input an index $j \in [M]$, attestations $(att_i)_{i\in[K]}$ for $|K| = \rho$ and $K \subset [N]$, and the anticipation ant. It returns as output a signature σ on the transaction Tx_j.

B. SightCVC Overview

As shown in Fig. 1, there are four mutually distrustful parties in SightCVC. For convenience, we refer to them as Alice, Ingrid, Bob, and oracle. Alice and Bob hold accounts on the sender chain (C_1) and receiver chain (C_2), respectively, while Ingrid maintains accounts on both C_1 and C_2. Oracle facilitates the reading of state changes between C_1 and C_2. SightCVC enables Alice and Bob to execute an unlimited number of off-chain transactions, provided that they agree on the transaction amounts. The funds for Virtual Alice and Virtual Bob are provided by Ingrid but are independently controlled by Alice and Bob. We designate the sender of the transaction as P, the receiver as Q, and the intermediary as I. Since our channel is bidirectional, Alice and Bob can serve as either P or Q, while Ingrid is always designated as I. In contrast to the previous CVC solution [15], SightCVC offers a comprehensive oracle design. Its detailed description with the complete oracle design will be given in Section IV. SightCVC comprises three main procedures: *Open*, *Update*, *Close*.

1) Open Procedure: During the process of opening a cross-chain virtual channel, users P and Q, acting as *EndUsers*, first submit requests to a smart contract to establish a payment channel, locking their respective funds. Once the funds are successfully locked, both parties exchange signatures and construct transaction details, which are then sent to the intermediary, Ingrid. Subsequently, P and Q submit oracle function invocation requests to the smart contract, awaiting signature confirmation. After verifying the message validity, they send channel opening requests to the smart contract, finalizing the cross-chain payment channel deployment. The oracle function ensures atomicity, meaning both channels either open simultaneously or neither opens. As the intermediary, Ingrid confirms the transactions and sends the channel opening confirmation to the smart contract, ensuring the VC is successfully opened.

2) Update Procedure: In the process of updating the cross-chain virtual channel ζ, Party P receives an update request from the environment $\mathcal{E}$, defines the current state of channel ζ_1, and signs it. P then updates the channel state and balance, sending the updated information to Party Q. Upon receiving the message, Q verifies the signature, updates its own channel state and balance, and adjusts the state of cross-chain channel ζ_2 based on the exchange rate. Both parties confirm the update and submit a request to the environment to finalize the channel state update. If either party fails, the smart contract returns NOTUPDATE and terminates the process.

3) Close Procedure: For closing the cross-chain virtual channel ζ, Party P receives a closure request from the environment $\mathcal{E}$, calculates the current channel state, and signs it. P then sends the closure request to the intermediary I and waits for confirmation from the smart contract. P and Q exchange signatures and transaction details before submitting the final closure request to the smart contract to confirm fund distribution. I verifies the signatures, submits the closure request, and computes the final channel state, ensuring the process is completed and funds are distributed to both parties. If an error occurs, the smart contract returns NOTCLOSED.

IV. Detailed Method

In this section, we introduce the detailed method of the SightCVC protocol and illustrate its design motivation using

Below we abbreviate $I := \mathsf{Ingrid}$. For $P \in \zeta.\mathsf{endUsers}$, we denote $Q := \zeta.\mathsf{otherParty}(P)$.

Party P

Upon receiving $(\mathtt{CVCOPEN}, \zeta) \xleftarrow{r_0^P} \mathcal{E}$ proceed as follows:

1) Send $(\mathtt{LOPEN}, \beta, \beta.x_Q) \xrightarrow{r_0^P} \mathcal{SC}$.
2) Wait if $(\mathtt{LOPENED}_\beta, id_\beta) \xleftarrow{r_1^P \le r_0^P + t} \mathcal{SC}$.
3) Compute $\sigma_{\beta_{open}} = \mathsf{Sign}_{sk_Q}(\mathtt{LOPENED}_\beta)$, then send $(\mathtt{LOPENED}_\beta, \sigma_{\beta_{open}}) \xrightarrow{r_1^P} P$

Upon receiving $(\mathtt{LOPENED}_\alpha, \sigma_{\alpha_{open}}) \xleftarrow{r_2^P \le r_1^P + t} P$ proceed as follows:

1) Let $\theta_{\beta.Q} := (id_\beta, \beta.x_Q, Q)$, then compute $\sigma_{\theta_{\beta.Q}} = \mathsf{Sign}_{sk_Q}(\theta_{\beta.Q})$.
2) Let $\mathtt{TX}_\beta := (\theta_{\beta.Q}, \sigma_{\theta_{\beta.Q}}, \alpha.x_Q, t_{stp})$, then send $(\mathtt{TX}_\beta) \xrightarrow{r_2^P} I$.
3) Compute $\sigma_{\mathtt{TX}_\beta} = \mathsf{Sign}_{sk_Q}(\mathtt{TX}_\beta)$, then send $(\mathtt{SIGHTSIGNREQ}, \mathtt{LOPENED}_\alpha, \sigma_{\mathtt{TX}_\beta}) \xrightarrow{r_2^P+1} \mathcal{SC}$.
4) Wait for $(sight_\beta, \mathtt{SIGHTSIGNOK}) \xleftarrow{r_3^P \le r_2^P + t} \mathcal{SC}$, then send $(\mathtt{MESSAGEFORESEEREQ}, vk, \mathtt{LOPENED}_\alpha, \sigma_{\mathtt{TX}_\beta}) \xrightarrow{r_3^P} \mathcal{SC}$.
5) Wait for $(foresee_\beta, \mathtt{MESSAGEFORESEEOK}) \xleftarrow{r_4^P \le r_3^P + t} \mathcal{SC}$, then send $(\mathtt{CVCOPEN}, \zeta_2, \zeta_2.x_Q) \xrightarrow{r_4^P} \mathcal{SC}$ and $(foresee_\beta) \xrightarrow{r_4^P+1} I$.
6) Let $\theta_{\zeta_2.Q} := (id_{\zeta_2}, \zeta_2.x_Q, Q)$, then compute $\sigma_{\theta_{\zeta_2.Q}} = \mathsf{Sign}_{sk_Q}(\theta_{\zeta_2.Q})$.
7) Let $\mathtt{TX}_{\zeta_2} := (\theta_{\zeta_2.Q}, \sigma_{\theta_{\zeta_2.Q}}, \zeta_2.x_Q, t_{stp})$, then send $(\sigma_{\mathtt{TX}_{\zeta_2}}, \mathtt{TX}_{\zeta_2}) \xrightarrow{r_4^P+2} I$.
8) Wait for $(\mathtt{CVCOPENED}_{\zeta_2}) \xleftarrow{r_5^P \le r_4^P + t} \mathcal{SC}$, then procedure stop.

Party I

Upon receiving $(\mathtt{CVCOPEN}, \zeta) \xleftarrow{r_0^I} \mathcal{E}$ proceed as follows:

1) Wait for $(\mathtt{LOPENING}, \alpha) \xleftarrow{r_0^I+1} \mathcal{SC}$, then send $(\mathtt{LOPEN}, \mathtt{TRANSFERINIT}, \alpha.x_I) \xrightarrow{r_0^I+1} \mathcal{SC}$.
2) Wait for $(\mathtt{LOPENED}_\alpha) \xleftarrow{r_1^I \le r_0^I + t} \mathcal{SC}$, then send $(\mathtt{LOPENED}_\alpha) \xrightarrow{r_1^I} Q$.
3) Send $(\mathtt{SIGHTSIGNREQ}, \mathtt{LOPENED}_\alpha) \xrightarrow{r_1^I+1} \mathcal{SC}$ and $(\mathtt{SIGHTSIGNREQ}, \mathtt{LOPENED}_\beta) \xrightarrow{r_1^I+1} \mathcal{SC}$.
4) Wait for $(sight, \mathtt{SIGHTSIGNOK}) \xleftarrow{r_2^I \le r_1^I + t} \mathcal{SC}$ and $(foresee) \xleftarrow{r_2^I+1} P$, then send $(\mathtt{SIGHTREDEEMREQ}, sight, foresee) \xrightarrow{r_2^I+1} \mathcal{SC}$.
5) Wait for $(\sigma_{\mathtt{TX}}, \mathtt{SIGHTREDEEMOK}, \mathtt{CVCOPENING}, \zeta) \xleftarrow{r_2^I \le r_2^I + t} \mathcal{SC}$ and $(\sigma_{\mathtt{TX}_\zeta}, \mathtt{TX}_\zeta) \xleftarrow{r_2^I+1} P$, then Let $\zeta_1.x_{VQ} := \zeta.x_I$ and $\zeta_2.x_{VP} := \zeta_2.x_I$.
6) Let $\theta_1 := (id_{\zeta_1}, \zeta_1.x_P, \zeta_1.x_{VQ}, P, I)$ and $\theta_{\zeta_2} := (id_{\zeta_2}, \zeta_2.x_Q, \zeta_2.x_{VP}, Q, I)$.
7) Send $(\mathtt{CVCOPEN}, \mathtt{TRANSFERINIT}, \theta_1, \theta_{\zeta_2}, \zeta_1.x_{VQ}, \zeta_2.x_{VP}) \xrightarrow{r_2^I+1} \mathcal{SC}$.

Fig. 3. An instantiation of open procedure.

the transfer protocol as an example. We adopt a progressive approach, beginning with a naive construction, systematically analyzing potential vulnerabilities and challenges, proposing targeted solutions, and ultimately developing a comprehensive secure protocol system. The protocol overview is illustrated in Fig. 2. Fig. 3 details the design and implementation of the *Open* procedure, Fig. 4 presents the architecture of the *Update* procedure, Fig. 5 outlines the design specifics of the *Close* procedure, and Fig. 6 describes the core functional logic of the smart contract.

A. Naive Construction

We assume that mutually distrusted parties Alice and intermediary Ingrid have established a payment channel on blockchain C_1 (e.g., the Bitcoin network), while on the target blockchain C_2 (e.g., the Ethereum network), mutually distrusted parties Bob and intermediary Ingrid have established a corresponding payment channel. To prevent any nodes other than the endusers from observing and recording users' payment information, such as amounts and transaction

Party P

Upon receiving $(\texttt{UPDATE}, id_{\zeta_1}, \Delta, \zeta_1) \xleftarrow{r_0^P} \mathcal{E}$, proceed as follows:

1) Let $\theta_1 := (id_{\zeta_1}, \zeta_1.x, P) = (\theta_{1_P}, \theta_{1_Q})$.
2) Compute $\sigma_{\theta_1} := \mathsf{Sign}_{sk_P}(\theta_1)$ and $\sigma_{\theta_{\zeta_1.P}} := \mathsf{Sign}_{sk_P}(\theta_{\zeta_1.P})$.
3) Let $\tilde{\theta}_{1_P} := \theta_{1_P}$ and $\zeta_1.\tilde{x}_P := \zeta_1.x_P - \Delta_{\zeta_1}$.
4) Compute $\tilde{\sigma}_1 := \mathsf{Sign}_{sk_P}(\tilde{\theta}_{1_P})$.
5) Send $(\texttt{UPDATING}, (\tilde{\theta}_1, c_1 + 1, \zeta_1, \tilde{\sigma}_1)) \xhookrightarrow{r_0^P} Q$.

Party Q

Upon receiving $(\texttt{UPDATING}, (\tilde{\theta}_1, c_1 + 1, \zeta_1, \tilde{\sigma}_1)) \xleftarrow{r_0^Q} P$, proceed as follows:

1) Verify $\tilde{\sigma}_1 = \mathsf{Sign}_{sk_P}(\tilde{\theta}_{1_P})$.
2) Let $\tilde{\theta}_{1_Q} := \theta_{1_Q}$ and $\zeta_1.\tilde{x}_Q := \zeta_1.x_Q + \Delta_{\zeta_1}$, then compute $\sigma_{\theta_{\zeta_2.Q}} := \mathsf{Sign}_{sk_Q}(\theta_{\zeta_2.Q})$ and $\Delta_{\zeta_2} := Rate \times \Delta_{\zeta_1}$.
3) Let $\tilde{\theta}_{2_Q} := \theta_{2_Q}$ and $\zeta_2.\tilde{x}_Q := \zeta_2.x_Q + \Delta_{\zeta_2}$, then send $(\texttt{UPDATEREQ}, \zeta_2, \tilde{\theta}_2, id_{\zeta_2}) \xhookrightarrow{r_0^Q} \mathcal{E}$.
4) Upon receive $(\texttt{UPDATEOK}_{\zeta_2}) \xleftarrow{r_1^Q \le r_0^Q + t} \mathcal{E}$, compute $\tilde{\sigma}_2 := \mathsf{Sign}_{sk_Q}(\tilde{\theta}_{2_Q})$.
5) Send $(\texttt{UPDATING}, (\tilde{\theta}_2, c_2 + 1, \zeta_2, \tilde{\sigma}_2)) \xhookrightarrow{r_1^Q} P$.

Party P

Upon receiving $\texttt{UPDATEOK}_{\zeta_2} \xleftarrow{r_1^P \le r_0^P + t} \mathcal{E}$, proceed as follows:

1) Wait for $(\texttt{UPDATING}, (\tilde{\theta}_2, c_2 + 1, \zeta_2, \tilde{\sigma}_2)) \xleftarrow{r_1^P + 1} Q$.
2) Verify $\tilde{\sigma}_2 := \mathsf{Sign}_{sk_Q}(\tilde{\theta}_{2_Q})$.
3) Let $\tilde{\theta}_{2_P} := \theta_{2_P}$ and $\zeta_2.\tilde{x}_P := \zeta_2.x_P - \Delta_{\zeta_2}$, then compute $\sigma_{\theta_{\zeta_2.P}} := \mathsf{Sign}_{sk_P}(\theta_{\zeta_2.P})$.
4) Send $(\texttt{UPDATEREQ}, \zeta, \tilde{\theta}, id_\zeta) \xhookrightarrow{r_1^P + 1} \mathcal{E}$ and wait for $\texttt{UPDATEOK} \xleftarrow{r_2^P \le r_2^P + t} \mathcal{E}$.
5) Compute $\tilde{\sigma}_P := \mathsf{Sign}_{sk_P}(\tilde{\theta})$, then send $(\texttt{UPDATED}_P, (\tilde{\theta}, \zeta, \tilde{\sigma}_P)) \xhookrightarrow{r_2^P} Q$. Else send $(\texttt{NOTUPDATE}) \xhookrightarrow{r_2^P} \mathcal{E}$ and go to stop.

Party Q

Upon receiving $(\texttt{UPDATED}_P, (\tilde{\theta}, \zeta, \tilde{\sigma}_P)) \xleftarrow{r_2^Q \le r_1^Q + t} Q$, proceed as follows:

1) Verify $\tilde{\sigma}_P := \mathsf{Sign}_{sk_P}(\tilde{\theta})$.
2) Compute $\tilde{\sigma}_Q := \mathsf{Sign}_{sk_Q}(\tilde{\theta})$.
3) Send $(\texttt{UPDATED}_P, (\tilde{\theta}, \zeta, \tilde{\sigma}_Q)) \xhookrightarrow{r_2^Q} P$. Else $(\texttt{NOTUPDATE}) \xhookrightarrow{r_2^Q} \mathcal{E}$ and go to stop.

Fig. 4. An instantiation of update procedure.

frequencies, the transaction privacy discussed in this paper specifically refers to the complete invisibility of off-chain transactions to external observers, with only the final settlement results recorded on the blockchain and made public. During protocol execution, both parties must continuously monitor transaction states on the C_2 chain (by running a full node or querying blockchain explorer interfaces). For simplicity, this work assumes that 1 native token of C_1 equals 1 token of C_2 in value; in practice, cross-chain value anchoring can be achieved through fixed exchange rates or asset encapsulation technologies. If Alice wishes to transfer 5 tokens to Bob on ζ, Bob and intermediary Ingrid must lock at least 5 tokens as collateral in the smart contract on C_2 to activate ζ. The successful execution of ζ's *Open* procedure requires strict timing constraints: ζ becomes effective only when both parties submit valid proofs of payment channel state updates to the contract within a preset time window. In the *Open* procedure, participants must submit to the smart contract: (i) the channel funding transaction $\texttt{TX}$ and (ii) the channel opening signature σ. This process creates a security risk: if the commitment transaction $\texttt{TX}$ and signature σ are submitted to the smart contract and broadcast to the chain,

Party P

Upon receiving $(\mathsf{CVCCLOSE}, \zeta) \xleftarrow{r_0^P} \mathcal{E}$, proceed as follows:

1) Compute $\theta := (id_\zeta, \zeta.x, EndUsers) = (\theta_P, \theta_Q)$.
2) Let $\theta_{\zeta_P} := (id_\zeta, \zeta.x_P, P)$.
3) Check if $\sigma_{\zeta_Q} = \mathsf{Sign}_{sk_Q}(\theta_{\zeta_Q})$.
4) Compute $\sigma_{\theta_{\zeta_P}} = \mathsf{Sign}_{sk_P}(\theta_{\zeta_P})$.
5) Let $\mathtt{TX}_\zeta := (\theta_{\zeta_P}, \sigma_{\theta_{\zeta_P}}, \zeta.x_P, t_{stp})$, then $(\mathtt{CVCLOSE}, \theta_{\zeta_Q}, \sigma_{\zeta_Q}) \xhookrightarrow{r_0^P} I$.
6) Wait for $(\mathtt{CVCCLOSING}, id_\zeta) \xleftarrow{r_1^P \le r_0^P + t} \mathcal{SC}$.
7) Send $(\mathtt{CVCCLOSING}, id_\zeta, \theta_\zeta, \sigma_\zeta) \xhookrightarrow{r_1^P} \mathcal{SC}$.
8) Wait for $(\mathtt{CVCCLOSEFINAL}_{\zeta_P}) \xleftarrow{r_2^P \le r_1^P + t} \mathcal{SC}$, then send $(\mathtt{CVCCLOSEFINAL}_{\zeta_P}) \xhookrightarrow{r_2^P} Q$.
9) Compute $\sigma_{\mathtt{TX}_{\zeta_P}} = \mathsf{Sign}_{sk_P}(\mathtt{TX}_{\zeta_P})$, then send $(\mathtt{SIGHTSIGNREQ}, \mathtt{CVCLOSEFINAL}_{\zeta_P}, \sigma_{\mathtt{TX}_\zeta}) \xhookrightarrow{r_2^P + 1} \mathcal{SC}$.
10) Wait for $(sight_{\zeta_P}, \mathtt{SIGHTSIGNOK}) \xleftarrow{r_2^P \le r_1^P + t} \mathcal{SC}$, then send $(\mathtt{MESSAGEFORESEEREQ}, \tilde{vk}, \mathtt{CVCLOSEFINAL}_{\zeta_P}, \sigma_{\mathtt{TX}_\zeta}) \xhookrightarrow{r_2^P} \mathcal{SC}$.
11) Wait for $(foresee_{\zeta_P}, \mathtt{MESSAGEFORESEEOK}) \xleftarrow{r_3^P \le r_2^P + t} \mathcal{SC}$.
12) Send $(\mathtt{CVCCLOSEDREQ}_{\zeta_P}, \zeta, \zeta.x_P) \xhookrightarrow{r_3^P} \mathcal{SC}$ and $(foresee_\alpha) \xhookrightarrow{r_3^P + 1} I$.

Upon receiving transfer $\zeta_2.x \xleftarrow{r_4^P \le r_3^P + t} I$, proceed as follows:

1) Confirm $\zeta.x_P$, then send $(\mathtt{VALREADYCLOSED}) \xhookrightarrow{r_4^P} \mathcal{SC}$ and go to stop.

Party I

Upon receiving $(\mathtt{CVCCLOSE}, \theta_{\zeta_Q}, \sigma_{\zeta_Q}) \xleftarrow{r_0^I} P$, proceed as follows:

1) Verify $\sigma_{\zeta_Q} := \mathsf{Sign}_{sk_P}(\zeta_Q)$.
2) Send $(\mathtt{CVCCLOSEINIT}, \sigma_{\zeta_Q}) \xhookrightarrow{r_0^I} \mathcal{SC}$.
3) Compute $\theta := \mathsf{Win}(\theta_{\zeta_P}, \theta_{\zeta_Q})$.
4) Wait for $(\mathtt{CVCLOSEFINAL}_{\zeta_P}) \xleftarrow{r_1^I \le r_0^I + t} \mathcal{SC}$, then send $(\mathtt{CVCLOSEFINAL}_{\zeta_P}) \xhookrightarrow{r_1^I} Q$.
5) Send $(\mathtt{SIGHTSIGNREQ}, \mathtt{CVCCLOSED}_\zeta) \xhookrightarrow{r_1^I + 1} \mathcal{SC}$.
6) Wait for $(sight, \mathtt{SIGHTSIGNOK}) \xleftarrow{r_2^I \le r_1^I + t} \mathcal{SC}$ and $(foresee) \xleftarrow{r_2^I + 1 \le r_1^I + t} P$, send $(\mathtt{SIGHTREDEEMREQ}, sight, foresee) \xhookrightarrow{r_2^I} \mathcal{SC}$.
7) Wait for $(\sigma_{\mathtt{TX}_\zeta}, \mathtt{SIGHTREDEEMOK}, \mathtt{CVCOPENING}, \zeta) \xleftarrow{r_3^I \le r_2^I + t} \mathcal{SC}$ and $(\sigma_{\mathtt{TX}_\zeta}, \mathtt{TX}_\zeta) \xleftarrow{r_3^I + 1 \le r_2^I + t} P$.
8) Let $\zeta.x_P := \zeta.\tilde{x}_P$ and $\zeta.x_Q := \zeta.\tilde{x}_Q$.
9) Wait for $(\mathtt{CVCCLOSED}) \xleftarrow{r_3^I + 2} \mathcal{SC}$, Else let $\theta := \perp$.
10) Adds $\zeta.x_P$ to Q's account and $-\zeta.x_P$ to P's account.
11) Send $(\mathtt{VALREADYCLOSEDREQ}) \xhookrightarrow{r_3^I + 3} \mathcal{SC}$.

Upon receiving $\mathtt{VALREADYCLOSED} \xleftarrow{r_4^I \le r_3^I + t} \mathcal{SC}$, proceed as follows:

1) Send $\mathtt{VALREADYCLOSED} \xhookrightarrow{r_4^I} \mathcal{E}$ and go to stop.

Fig. 5. An instantiation of close procedure.

non-participants may intercept this data. Any entity obtaining this transaction data and both signatures can trigger the forced settlement mechanism of the payment channel. To mitigate this vulnerability, we recommend a transaction submission mechanism—simultaneously submitting two payment channel commitment transactions $\mathtt{TX}_\alpha$ and $\mathtt{TX}_\beta$ to the contract, with time-bound opening signatures $\sigma_{\alpha_{open}}$ and $\sigma_{\beta_{open}}$. However, this basic scheme presents two critical vulnerabilities: (i) Amount

(A) The contract for CVC open:

Upon receiving $(\texttt{LOPEN}, \gamma, \gamma.x_P) \xleftarrow{t_0} P$, proceed as follows:

1) Let $id_\gamma := \mathsf{subchaincontract}(P)$, then send $(\texttt{LOPENING}, \gamma) \xhookrightarrow{t_0} I$.
2) If $(\texttt{LOPEN}, \texttt{TRANSFERINIT}, \gamma.x_I) \xleftarrow{t_1 \le t_0 + t} I$, then send $(\texttt{LOPENED}_\gamma) \xhookrightarrow{t_1} \gamma.\mathsf{endUsers}$ and go to procedure (B).

Upon receiving $(\texttt{CVCOPEN}, \zeta, \zeta.x_P) \xleftarrow{t_2 \le t_1 + t} P$ proceed as follows:

1) Let $id_{\zeta_1} := \mathsf{subchaincontract}(P)$ and $id_{\zeta_2} := \mathsf{subchaincontract}(Q)$ and $\zeta := (\zeta_1, \zeta_2)$.
2) Send $(\texttt{CVCOPENING}, \zeta) \xhookrightarrow{t_2} I$ and $(id_\zeta) \xhookrightarrow{t_2+1} P$.
3) If $(\texttt{CVCOPEN}, \texttt{TRANSFERINIT}, \theta_1, \theta_{\zeta_2}, \zeta_1.x_{VQ}, \zeta_2.x_{VP}) \xleftarrow{t_3 \le t_2 + t} I$. Then $(\texttt{CVCOPENED}_{\zeta_1}) \xhookrightarrow{t_3} P$ and $(\texttt{CVCOPENED}_{\zeta_2}) \xhookrightarrow{t_3+1} Q$. Else $(\texttt{CVCNOTOPENED}, \texttt{TIMEOUT}, \zeta.x_P) \xhookrightarrow{t_3} P$.

(B) The contract for CVC close:

Upon receiving $(\mathsf{CVCCLOSEINIT}, (\zeta, \sigma_\zeta), \tau_{clo}) \xleftarrow{t_0} I$, proceed as follows:

1) Check if $\zeta \notin \tau_{clo}$, then send $(\mathsf{CVCLOSEINIT}, id_\zeta) \xhookrightarrow{t_0} EndUsers$.
2) Wait for $(\mathsf{CVCCLOSE}, \theta_\zeta, \sigma_\zeta) \xleftarrow{t_1 \le t_0 + t} EndUsers$, then send $(\mathsf{CVCCLOSE}, \theta_\zeta, \sigma_\zeta) \xhookrightarrow{t_1} I$. Else let $\tau_{clo} := \tau_{clo} \cup \{\zeta\}$.

Upon receiving $(\texttt{CVCLOSEFINAL}, (\theta_{\zeta_P}, \sigma_{\zeta_P}), (\theta_{\zeta_Q}, \sigma_{\zeta_Q})) \xleftarrow{t_2 \le t_1 + t} I$, proceed as follows:

1) Send $(\texttt{CVCLOSEFINAL}, (\theta_{\zeta_Q}, \sigma_{\zeta_P}), (\theta_{\zeta_P}, \sigma_{\zeta_Q})) \xhookrightarrow{t_2} EndUsers$.
2) Wait for $(\texttt{VALREADYCLOSED}, \sigma_{clo}) \xleftarrow{t_3 \le t_2 + t} EndUsers$, then let $\tau_{clo} := \tau_{clo} \cup \{\zeta\}$. Else do nothing.

(C) The contract for channel Punishing:

Remove x coins from corrupted party P' account in $\mathsf{transfer}$ and add x coins to Q's account in $\mathsf{transfer}$.
Let $\tau_{clo} = \tau_{clo} \cup \{id\}$, then send $(\texttt{PUNISHED}) \xhookrightarrow{t_0} EndUsers$.

(D) The contract for oracle procedures:

Upon receiving $(\texttt{LOPEN}, \beta, \beta.x_P) \xleftarrow{t_0} P$

1) Generate $(pk^{\mathcal{O}}, sk^{\mathcal{O}}) := \mathsf{OGenR}$ and set $(vk, sk) := (pk^{\mathcal{O}}, sk^{\mathcal{O}})$.
2) Send $(\texttt{OGENOK}, vk) \xhookrightarrow{t_0} P$.

Upon receiving $(\texttt{SIGHTSIGNREQ}, m) \xleftarrow{t_1 \le t_0 + t} P$. Where M represents the message generated by the protocol Π_{pol} instruction.

1) Compute $\sigma_m^{\mathcal{O}} := \mathsf{Sign}_{sk}(m)$. Let $sight := \sigma_m^{\mathcal{O}}$.
2) Check if $\mathsf{Vrfy}_{sk}(m, sight) = 1$.
3) If the above check is successful, then $(sight, \texttt{SIGHTSIGNOK}) \xhookrightarrow{t_1} P$. Else return 0 and stop.

Upon receiving $(\texttt{MESSAGEFORESEEREQ}, vk, m, \hat{m}) \xleftarrow{t_2 \le t_1 + t} P$ proceed as follows:

1) Compute $(c, \pi_c) := \mathsf{VweTS.EncSig}((vk, m), (sk_B, \hat{m}))$, then let $foresee := (c, \pi_c)$.
2) Check if $\mathsf{VweS.VfEnc}(c, \pi_c, vk, m, \hat{m}) = 1$.
3) If the above check is successful, then $(foresee, \texttt{MESSAGEFORESEEOK}) \xhookrightarrow{t_2} P$. Else return 0 and stop.

Upon receiving $(\texttt{SIGHTREDEEMREQ}, sight, foresee) \xleftarrow{t_3 \le t_2 + t} P$ proceed as follows:

1) Compute $\sigma_{\hat{m}}^P := \mathsf{VweS.DecSig}(\sigma_m^{\mathcal{O}}, (c, \pi_c))$.
2) Send $(\sigma_{\hat{m}}^P, \texttt{SIGHTREDEEMOK}) \xhookrightarrow{t_3} P$.

Fig. 6. Smart contract functionality $\mathcal{SC}$.

verification defect: The channel state commitment transaction records only balance snapshots without explicitly encoding transaction amount information, preventing the smart contract from verifying whether Alice has actually transferred 5 tokens to Bob in state channel ζ. (ii) State version inconsistency risk: Attackers may maliciously submit different versions of channel state commitment transactions. For instance, Alice might submit commitment transaction $\texttt{TX}_n$ corresponding to the n-th

update of ζ, while Bob submits commitment transaction $\mathtt{TX}_m$ for the m-th update (where $m \neq n$). This version misalignment prevents the smart contract from determining the channel's true state.

B. Transfer Cross-Chain Virtual Channel Balance to C_2

To verify that the channel update request submitted by Alice corresponds to the $A \xrightarrow{5} B$ operation, the smart contract must know the balance distribution state of both parties in channel ζ prior to the update. For instance, in the *Open* procedure, Alice and Bob can declare their initial balance distribution to the smart contract through the funding lock transactions of α and β, and subsequently execute the $A \xrightarrow{5} B$ update operation. However, before the completion of the SightCVC protocol, neither party can perform any intermediate update operations and must strictly execute $A \xrightarrow{5} B$ according to the declared balances. To address this limitation, the smart contract can be granted access to the channel state before the update. Specifically, if the contract can simultaneously access the commitment transaction before the update (Alice holds 10 tokens, Bob holds 5 tokens) and after the update (Alice holds 5 tokens, Bob holds 10 tokens), it can verify that Alice has indeed transferred 5 tokens. Although this approach resolves the defect in amount verification, two significant inadequacies persist: (i) voluminous proof data requiring multiple commitment transactions to be submitted, and (ii) the unresolved state version inconsistency risk.

C. Embed Oracle Contract Functionality in SightCVC

To prevent participant corruption and ensure smart contracts can verify the validity of the $A \xrightarrow{5} B$ update, this work proposes a novel protocol: enabling an arbitrary number of update operations during channel ζ's state *Update* procedure while requiring only streamlined verification materials consisting of state proofs from α and β. In the implementation, participants must embed the oracle contract function module into the SightCVC protocol within the commitment transaction for $A \xrightarrow{5} B$, following the process detailed below:

- **Key Initialization:** The successful state updates of α and β are designated as events $\mathtt{LOPENED}_\alpha$ and $\mathtt{LOPENED}_\beta$, respectively.
- **Event Definition:** The anticipatory proof *foresee* for the LOPENED event is generated using the private key $sk^{\mathcal{O}}$.
- **Proof Generation:** The expected verification parameters *sight* for the $A \xrightarrow{5} B$ operation are generated using the oracle public key $pk^{\mathcal{O}}$.
- **Anticipatory Verification:** Bob can obtain the signature σ_β for transaction $\mathtt{TX}_\beta$ only when he provides the $\mathtt{LOPENED}_\alpha$ event for α (a similar constraint applies to Alice).
- **Signature Constraint:** The signature pair $(\sigma_\alpha, \sigma_\beta)$ functions as the core credential for state transition, representing the initial state (opening of α and β) and the terminal state (closing of α and β), respectively.
- **State Evolution:** Signature pairs $(\sigma_\alpha, \sigma_\beta)$ authenticate state evolution, encoding initial (α/β activation) and terminal (α/β termination) phases.

This protocol enables multiple executions of the $A \xrightarrow{5} B$ operation within channel ζ without blockchain interaction, effectively resolving both proof data inflation and state version inconsistency challenges.

D. Exception Handling Strategies

To ensure liveness and safety in the presence of faults or malicious behavior, SightCVC relies on the *Offload* procedure to handle abnormal states. We begin by analyzing the conditions under which SightCVC enters such a state: when the environment $\mathcal{E}$ fails to receive identical event messages from both P and Q within the valid period of the current round, the system is considered to be in an abnormal state. These adverse scenarios typically arise from provision of outdated states and refusal to update states.

As shown in Fig. 7, to resolve potential disputes among P, I, and Q, we design the *Offload* procedure, which comprises three subroutines. From the perspective of a corrupted intermediary Ingrid, we explain how the offloading process addresses disputes between endusers P and Q and the intermediary Ingrid, while also illustrating the fault tolerance of the intermediary node:

- **Provision of outdated states.** A corrupted Ingrid may attempt to exploit an expired and revoked state ζ to update the underlying payment channel γ, thereby profiting from discrepancies in cross-chain transactions. In this case, the oracle in environment $\mathcal{E}$ receives different update events from α and β, which triggers the *Offload* procedure. At this point, endusers may execute the ChannelInitialize operation to close channel γ, thereby converting state ζ into a payment channel state. According to the payment channel protocol, the outdated state provided by Ingrid is deemed malicious, and all funds in the channel are refunded to the endusers.
- **Refusal to update states.** When Ingrid goes offline or deliberately refuses to update state ζ, the oracle fails to receive synchronized update messages from P and Q in environment $\mathcal{E}$, thereby triggering the *Offload* procedure. The system then sends a punishment confirmation message to enduser P and forcibly executes the ChannelInitialize operation to close channel α. The dispute is resolved based on the latest state. Subsequently, P is required to execute PunishSettle to complete the transfer to Q. Finally, Q receives a PUNISHED message from the environment, upon which Bob executes PunishTransfer to retrieve from Ingrid's account an amount equivalent to the transfer in ζ_1, as recorded in state ζ_2.

V. Modeling SightCVC in the UC-Framework

A. Security Model

To analyze the security of SightCVC, we employ the Global Universal Composability (GUC) framework [25],

which extends the original Universal Composability (UC) framework [16], [22], [23], [26], [27]. Our analysis closely follows the methodologies presented in [15], [16], [17], [22], [23], and [26]. The protocol Π of SightCVC is executed among a set of parties P, modeled as interactive Turing machines. During execution, parties exchange messages under the presence of an adversary $\mathcal{A}$. We assume a static corruption model, in which $\mathcal{A}$ selects the set of parties to corrupt before the protocol begins. Corruption grants $\mathcal{A}$ full control over the internal state of the corrupted parties, enabling it to send messages and execute arbitrary code on their behalf. Additionally, a special entity called the environment $\mathcal{E}$ provides inputs to each party and the adversary $\mathcal{A}$, and observes all outputs. The environment $\mathcal{E}$ simulates all external influences outside the protocol execution. The environment $\mathcal{E}$, and by extension the adversary $\mathcal{A}$, are given a security parameter $\lambda \in \mathbb{N}$ and an auxiliary input $e \in \{0,1\}^*$.

B. Communication Model

To model SightCVC in the synchronous communication model, we assume the existence of a global clock (as defined in [28]) that divides protocol execution into discrete rounds. This facilitates a more intuitive treatment of time. The global clock advances once all honest parties have completed their operations for the current round. In this setting, protocol execution proceeds in a round-based fashion. For instance, t_0^P denotes the round in which party P begins executing the protocol. With the aid of the global clock, all honest parties remain synchronized and aware of the current round.

Message delivery adheres to the following rule: any message sent by party P to party Q in round t_0^P is received by round $t_1^Q \leq t_0^P + t$, where t denotes the maximum network delay. The adversary has the following capabilities and limitations: it can observe messages exchanged between parties and control the delivery order of messages within a round, but it cannot drop, delay, or modify them, nor can it alter the relative order of messages exchanged between honest parties.

For communications involving third-party entities (e.g., the environment $\mathcal{E}$), message exchanges may occur directly without requiring round synchronization. We assume that all computations are completed within a single round. In protocol descriptions, when certain operations are required to complete within time, the number of rounds needed is determined by the adversary but remains bounded by a predefined upper limit.

C. Security Analysis

We first present the Universal Composability (UC) security definition. Let Π denote a hybrid protocol that has access to a set of auxiliary ideal functionalities $\mathcal{F}_a$. Consider an environment $\mathcal{E}$ interacting with an adversary $\mathcal{A}$. Given inputs λ and e, the execution set $EXEC^{\mathcal{F}_a}_{\Pi,\mathcal{A},\mathcal{E}}(\lambda, e)$ is defined as the set of all outputs and side effects produced through interaction with protocol Π. These outputs and side effects are observable to the environment $\mathcal{E}$. Furthermore, let $\phi\mathcal{F}$ denote an idealized protocol corresponding to an ideal functionality $\mathcal{F}$, where messages between $\mathcal{F}$ and $\mathcal{E}$ are routed through a dummy party. The idealized protocol $\phi\mathcal{F}$ also has access to the ideal functionalities $\mathcal{F}_a$. In the ideal world, the execution set observed by environment $\mathcal{E}$ when interacting with $\phi\mathcal{F}$ and simulator $\mathcal{S}$ is defined as $EXEC^{\mathcal{F}_a}_{\phi\mathcal{F},\mathcal{S},\mathcal{E}}(\lambda, e)$. If protocol Π GUC-realizes functionality $\mathcal{F}$, then any attack against the real-world protocol Π can be effectively simulated in the idealized protocol $\phi\mathcal{F}$. This equivalence establishes that protocol Π provides the same security guarantees as the idealized protocol $\phi\mathcal{F}$. The formal security definition as follows.

Theorem 1: A protocol Π GUC-realizes the ideal functionality $\mathcal{F}$ with respect to $\mathcal{F}_a$ if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that

$$EXEC^{\mathcal{F}_a}_{\Pi,\mathcal{A},\mathcal{E}}(\lambda, e) \overset{c}{\approx} EXEC^{\mathcal{F}_a}_{\phi\mathcal{F},\mathcal{S},\mathcal{E}}(\lambda, e)$$

where $\lambda \in \mathbb{N}, e \in \{0,1\}^*$ and $\approx^c$ denotes computational indistinguishability.

Our objective is to achieve atomicity as a security property within the SightCVC protocol. Atomicity ensures that, given the latest valid state on SightCVC, honest users can guarantee two fundamental properties: (i) the corresponding outcome is enforceable by honest users on SightCVC; (ii) no alternative outcome can be enforced on SightCVC while the state remains the latest valid state—failure to satisfy this condition results in users obtaining all funds in the corresponding SightCVC channel. To formalize this concept, we define an ideal function $\mathcal{F}_{\text{SightCVC}}$ that explicitly specifies the input/output behavior, ledger side effects, and requisite properties. This formalization enables us to prove that SightCVC satisfies Theorem 1.

VI. Performance Evaluation

In this section we evaluate SightCVC cross-chain transactions using two real payment channel network topologies on Ripple and Ethereum. Since Ripple does not inherently support smart contracts, we conducted the simulation using the payment channel smart contract provided by Ripple's official sources. To thoroughly test SightCVC's performance, we employed a single intermediate node with accounts on both Ripple and Ethereum.

A. Transaction Delay

We simulated a cross-chain scenario to assess the latency of each transaction. Specifically, we first constructed a blockchain network with N nodes, where N represents the number of nodes in the blockchain. Each node was configured to open m channels, consisting of two layers—payment channels and virtual channels—within which user transactions occur. Based on this experimental setup, we measured transaction latency by varying the values of N and m, with N ranging from 3 to 30, and m set to 4 and 10. Transaction latency, defined as the time interval between the sending and confirmation of a cross-chain transaction, is presented in Fig. 8. The result indicates that as the number of nodes increases from 10 to 100, transaction latency also increases. This trend occurs because a larger number of nodes requires more time for transaction broadcasting and consensus. Additionally, the number of channels in the network influences transaction latency; as the number of channels increases, the number of queued transactions rises, resulting in higher latency.

(A) **Subprocedure** ChannelInitialize :

Upon receiving $(\texttt{OFFLOADCONFIRM}, \alpha) \xleftrightarrow{r_0} \mathcal{E}$ proceed as follows:

1) Send $(\texttt{OFFLOADREQ}, \alpha) \xhookrightarrow{r_0} \mathcal{SC}$.

Upon receiving $(\mathsf{PCCLOSE}, \alpha) \xleftrightarrow{r_1 \le r_0 + t} \mathcal{E}$, proceed as follows:

1) Compute $\theta := (id_\alpha, \alpha.x, EndUsers) = (\theta_P, \theta_I)$ and $\sigma_{\theta_P} = \mathsf{Sign}_{sk_P}(\theta_P)$.
2) Let $\texttt{TX}_\alpha := (\theta_P, \sigma_{\theta_P}, \alpha.x_P, r)$, then $(\texttt{PCCLOSE}, \theta_Q, \sigma_Q) \xhookrightarrow{r_1} I$.

Upon receiving $(\texttt{PCCLOSING}, id_\zeta) \xleftrightarrow{r_2 \le r_1 + t} I$ proceed as follows:

1) Send $(\texttt{PCCLOSING}, id_\alpha, \theta_P, \sigma_\alpha) \xhookrightarrow{r_2} \mathcal{E}$.

Upon receiving $(\texttt{PCCLOSEFINAL}_{\alpha_P}) \xleftrightarrow{r_3 \le r_2 + t} \mathcal{E}$ proceed as follows:

1) Compute $\sigma_{\texttt{TX}_\alpha} := \mathsf{Sign}_{sk_P}(\texttt{TX}_\alpha)$.
2) Send $(\texttt{PCCLOSEFINAL}_{\alpha_P}) \xhookrightarrow{r_3} I$ and $(\texttt{SIGHTSIGNREQ}, \texttt{PCCLOSEFINAL}_{\alpha_P}, \sigma_{\texttt{TX}_\alpha}) \xhookrightarrow{r_3^P} \mathcal{E}$.

Upon receiving $(sight_{\zeta_P}, \texttt{SIGHTSIGNOK}) \xleftrightarrow{r_4 \le r_3 + t} \mathcal{SC}$ proceed as follows:

1) Send $(\texttt{MESSAGEFORESEEREQ}, \texttt{PUNISHREQ}, \tilde{vk}, \texttt{CVCLOSEFINAL}_{\alpha_P}, \sigma_{\texttt{TX}_\alpha}) \xhookrightarrow{r_4} \mathcal{SC}$.
2) Upon receiving penalty, then procedure stop.

(B) **Subprocedure** PunishSettle :

Upon Ingrid receiving $(\texttt{PUNISHCONFIRM}, \zeta) \xleftrightarrow{r_0} \mathcal{E}$ proceed as follows:

1) Check if ζ expired, send $(\texttt{PUNISHREQ}, \zeta) \xhookrightarrow{r_0} \mathcal{SC}$. Else stop.
2) Wait for $(\texttt{VALREADYCLOSED}, \texttt{TX}_\gamma) \xleftrightarrow{r_1 \le r_0 + t} \mathcal{SC}$.
3) Verify $\sigma_{\texttt{TX}_\alpha} = \mathsf{Sign}_{sk_P}(\texttt{TX}_\alpha)$. If yes waits for $2r$ rounds and then get penalty $\alpha.x_A$ from $\mathcal{SC}$ and go to stop. Elsh Ingrid transfers $\alpha.x_I$ to P.

(C) **Subprocedure** PunishTransfer :

Upon Q receiving $(\texttt{PUNISHED}, \alpha) \xleftrightarrow{r_0} \mathcal{E}$ proceed as follows:

1) Send $(\texttt{PUNISHED}, \sigma_{\texttt{TX}_\alpha}, \beta.x_I, \beta) \xhookrightarrow{r_0} \mathcal{SC}$.
2) Wait for $2r$ rounds and then get penalty $\beta.x_I$ from $\mathcal{SC}$ and go to stop.

Fig. 7. An instantiation of offload procedure.

TABLE I
THE FEE COST FOR SIGHTCVC

	#on-chain transactions	cost			#off/on chain msg		#ciphertext
		Gas($\times 10^3$)	ETH($\times 10^{-3}$)	USD	endusers	intermediary	
Open PC	1	129	0.85	2.1	1	1	0
Open CVC	2	206.4	1.36	3.4	2	2	2
Update PC	0	0	0	0	0	0	2
Update CVC	0	0	0	0	0	0	0
PC Closing: optimistic	2	126.8n+131.5	0.84n+0.87	2.1n+2.2	2	2	2
CVC Closing: optimistic	4	228.2n+223.5	1.51n+1.48	3.8n+3.7	4	4	0

We also compared SightCVC with AMHL [10], AMHL+ [10], EHMHL+ [29], Cross-channel [30], and CVC [15] in terms of cross-chain transaction latency. The results demonstrate that SightCVC achieves the lowest transaction latency. This is because, unlike AMHL, AMHL+, EHMHL+ and Cross-channel, SightCVC primarily conducts cross-chain transactions in virtual channels, which offer higher throughput. SightCVC's lower transaction latency compared to CVC is attributed to its avoidance of complex voting consensus for completing oracle functions, though this advantage assumes

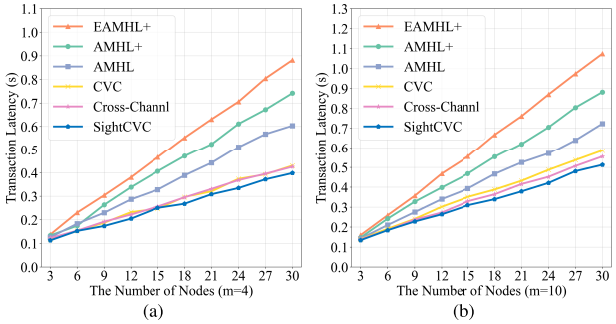

Fig. 8. Transaction delay comparisons.

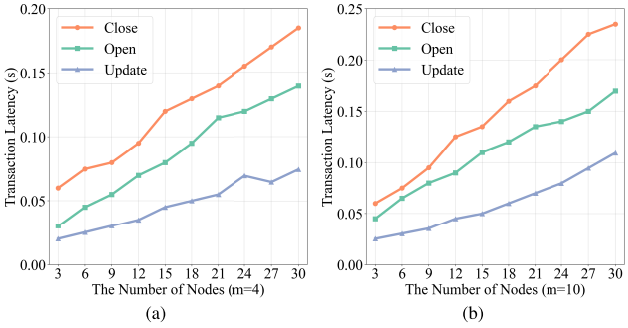

Fig. 9. SightCVC transaction delay details.

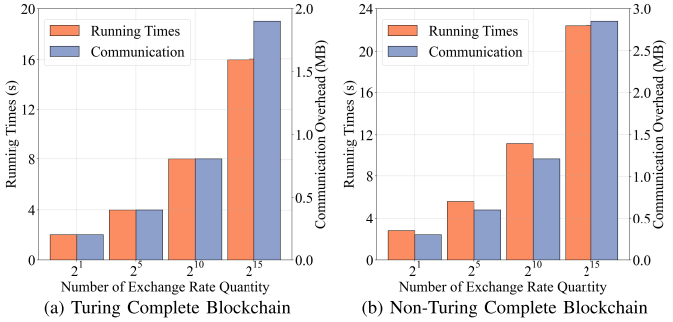

Fig. 10. Oracle overhead on turing complete blockchain and non-Turing complete blockchain.

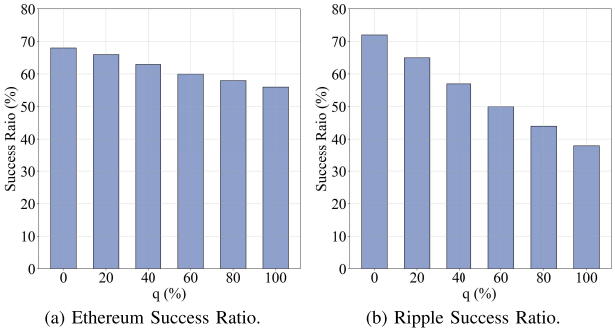

Fig. 11. Success ratio comparison.

that intermediate nodes remain online. Furthermore, we analyzed the transaction latency at each stage of SightCVC under the same N and m values, with the update stage being the primary stage for cross-chain transactions. The Fig. 9 shows that transaction latency at this stage is nearly zero.

B. Fee Costs

We evaluated the fee costs of SightCVC on the Ethereum network. In Ethereum, fee costs are calculated using an internal currency called Gas, which is paid by users to miners. In our setup, the amount of Gas required depends on the data size of the smart contract and its computational complexity. Additionally, the cost of SightCVC is influenced by the exchange rate between Gas and ETH. For our calculations, we used an updated exchange rate of 1 Gas = 6.61 Gwei = 6.61×10^{-9} ETH. Deploying the oracle smart contract requires approximately 3,300,000 Gas, which is equivalent to 0.022 ETH. As of 8:00 AM on October 28, 2024, the exchange rate of ETH to USD was approximately 1 ETH = 2,501 USD, making the cost of deploying the oracle smart contract around 54.55 USD. It is important to note that optimizing the Gas cost during smart contract deployment was not the focus of our implementation of SightCVC. Instead, we concentrated on the Gas costs of PC and CVC during the open, update, and close stages, as well as the required number of signatures. The specific results are presented in the Table I. As observed, following the design concept of virtual channels, the Gas costs for CVC in the open and close stages are nearly zero, although not exactly zero due to the additional Gas overhead required by the oracle. It is also noteworthy that the Gas cost for CVC in the update stage is zero because the oracle does not participate in this stage.

C. Oracle Overhead

We now evaluate the overhead of the oracle. The time and communication overhead of the oracle are primarily influenced by the number of exchange rates, as different exchange rates generate distinct ciphertexts. Fig. 10 presents the results. The experimental findings indicate that as the number of exchange rates increases, both running time and communication overhead rise significantly. However, for Turing complete blockchain, even with exchange rates reaching up to 2^{15}, the computational overhead remains below 20 seconds, while the communication overhead stays under 2 MB, which is entirely acceptable. In contrast, non-Turing complete blockchains, lacking complex control structures, require more cryptographic operations and external data calls, resulting in increased operational and communication overhead.

D. Success Ratio

We evaluated the success ratio of SightCVC in cross-chain transactions through simulations. In setting up cross-chain virtual channels, we prioritized establishing virtual channels for users with more than five transactions. We varied the ratio of cross-chain virtual channels to payment channels, denoted as q, from 0% to 100% to assess performance. When q is 0%, the system has no cross-chain transactions, while at q equal to 100%, only cross-chain virtual channels exist (where cross-chain virtual channels are treated as virtual channels), and all

TABLE II
COMPARISON OF SIGHTCVC WITH EXISTING WORKS

Approaches	Oracle Embed[1]	Off-Chain Transaction	High Throughput	Compatibility[2]	Security Availability
ACCS [9]	○	✗	✓	NTC	✗
zkCross [31]	○	✓	✓	TC	✓
BridgeGuard [32]	●	✓	✗	NTC	✓
AMHL [10]	◑	✓	✗	TC	✓
EAMHL [29]	◑	✓	✗	TC	✓
CVC [15]	◑	✓	✓	TC	✓
SightCVC	●	✓	✓	TC	✓

[1] ○: No oracle embed. ◑: Partially oracle embed. ●: Oracle embed.
[2] TC: Turing complete. NTC: Non-Turing complete.

high-frequency transactions are cross-chain. In each simulation run, 1,000 transactions were processed. Fig. 11 illustrates the transaction success ratio on the Ripple and Ethereum platforms across different values of q. At $q = 0\%$, SightCVC operates without cross-chain transactions, resembling the Perun scheme, where all users follow the platform-specified routing protocol. As the proportion of cross-chain virtual channels increases, the overall success ratio decreases significantly. This decline is attributed to the increased locking of funds within the network, reducing their availability for other virtual or payment channels. Furthermore, as the number of cross-chain virtual channels rises, the burden on intermediary nodes to respond grows, lowering the success ratio.

VII. RELATED WORK

In this section, we discuss the related work of the proposed approach, including cross-chain and off-chain protocol. Table II provides a detailed comparison of SightCVC with other protocols.

A. Cross-Chain Protocol

The prevailing cross-chain protocol in use is centralized, where traders need to deposit funds into a server. However, this approach poses a risk of fund loss in case the server is compromised by attackers.

- **ACCS.** ACCS [9] ensures payment confirmation by requiring transaction initiators to disclose a secret, reverting transactions if undisclosed before timeout. Each swap involves four on-chain transactions across two blockchains, incurring high costs and delays from constrained throughput.
- **Collateral-based cryptocurrency exchange.** XClaim [33] employs a third-party Vault locking collateral in blockchain A's smart contract. To transfer funds from blockchain B to A, initiators send funds to Vault on B for collateralized assets and transfer equivalent amounts to recipients on A. Initiators can exchange with others or lock A assets to obtain B assets. Vault retrieves collateral by proving B asset redemption, while A's smart contract eliminates locked assets. XClaim reduces on-chain transactions versus ACCS but remains throughput-limited. Moreover, Vault is obligated to provide collateral equivalent to the exchanged funds. Lastly, it is imperative for blockchain system A to possess Turing completeness, thus preventing XClaim from supporting exchanges between cryptocurrencies like Bitcoin.
- **Trusted Execution Environment (TEE).** Tesseract [34] executes cross-chain protocols in Intel SGX-based TEE, acting as a trusted third party. It generates blockchain-specific key pairs. Public key shared for deposits, and private key secured in TEE. Users reclaim deposits post-timeout or during TEE failures. This arrangement ensures the secure storage of the deposit as the TEE safeguards the undisclosed private key.

B. Off-Chain Protocol

- **Two-party payment channel.** Payment channels [35] enhance blockchain scalability, originally designed for Bitcoin-compatible systems. Moreno-Sanchez et al. [36] implemented payment channels on Monero, while we developed payment channels on Bitcoin and Ethereum. In response to privacy concerns within payment channels, Green et al. [37] introduced Bolt. In Bolt, transactions within the payment channel are not linkable. State channels [22], [38] generalize functionality by supporting arbitrary blockchain state transitions. Lind et al. [39] introduced asynchronous channels as a means to mitigate the synchronous assumption inherent in their construction. Asynchronous channels leveraging TEEs to eliminate blockchain syncing requirements while preventing misconduct, or employing third-party watchers [40], [41] to monitor counterparty on-chain actions.
- **PCNs.** PCNs expand a single payment channel into a network composed of two-party channels. Trustworthiness is not required among the parties participating in PCNs. Off-chain payments are conducted through payment channels established between these parties. To improve the success rate of payments, multiple path strategies [42], [43], [44] are simultaneously employed to discover paths with sufficient funds between users. Lastly, PCNs have also made efforts to address additional attributes, including

security [10], [32], efficiency [27], [38], [45], and privacy [16], [17], [31], [46].

VIII. Discussion

A. SightCVC in Non-Turing Complete Blockchains

We chose to deploy SightCVC on both the Turing complete Ethereum and the non-Turing complete Ripple blockchains because SightCVC cannot operate directly on non-Turing complete blockchains that offer only basic scripting capabilities. This limitation stems from the fact that SightCVC requires smart contract functionalities involving complex cryptographic operations, which exceed the capabilities of basic scripting systems. Furthermore, to implement oracle functionality, the system must maintain cross-chain state synchronization, necessitating that the underlying blockchain supports dynamic data structures and persistent storage.

For instance, Bitcoin's scripting system is stack-based with a simplified operational model that supports only fundamental cryptographic operations [47]. It lacks control structures such as loops and recursion and is restricted by a strict size limit of 520 bytes [48]. Consequently, the cryptographic computations required by SightCVC cannot be implemented on this platform. More importantly, Bitcoin's use of the UTXO model and absence of a global state concept prevent it from supporting oracle's cross-chain state synchronization requirements [49].

To enable SightCVC deployment on non-Turing complete blockchains, we can migrate the smart contract logic and cryptographic computations into a TEE. The TEE executes computations based on the contract state and encapsulates the results in transactions that include state updates. Each state update transaction references the preceding valid state to ensure the linear progression of states and prevent double-spending. The oracle's cross-chain states are persisted via consecutive UTXO outputs; critical state changes require multisignature approval by participants meeting a threshold. This approach allows SightCVC to achieve full smart contract and oracle functionality on non-Turing complete blockchains without modifying the underlying protocol.

Specifically, Alice and Bob must first collect payment channel states from the blockchain, then transmit these states and previous transactions to Ingrid. Ingrid subsequently executes the smart contract invocation and sends the event results along with her signature to the TEE, which completes the transaction and broadcasts it. The entire process consists of the following six steps:

- Step 1. Alice and Bob reconstruct the current smart contract state by traversing SightCVC's transaction history and extracting state changes from payment channels α and β.
- Step 2. Alice and Bob transmit the current payment channel states θ_α and θ_β, together with the smart contract code, to Ingrid.
- Step 3. Ingrid performs dual verification to validate both the contract code and payment channel states. Upon successful verification, she transmits the current state and input parameters to the TEE and executes the smart contract, generating new state changes for payment channels α and β and the corresponding fund transfer operations.
- Step 4. Ingrid constructs transactions $\mathtt{TX}_\alpha$ and $\mathtt{TX}_\beta$ based on the current state and establishes dependencies on previous transactions. She encodes the new state signature, contract code hash, and state change list as fund transfer information m, ultimately generating signature $\sigma_I = \mathsf{Sign}_{sk_I}(m)$ and returning it to the TEE.
- Step 5. The TEE collects the fund transfer information m and its signature σ_I. Due to the deterministic nature of smart contract execution, Alice and Bob generate identical fund transfer information m. The TEE subsequently generates multi-party signatures $\Sigma = \{\sigma_A, \sigma_B\}$, where $\sigma_A = \mathsf{Sign}_{sk_A}(m)$ and $\sigma_B = \mathsf{Sign}_{sk_B}(m)$.
- Step 6. The TEE broadcasts the signed event message (m, Σ, σ_I). This completes the SightCVC deployment on the non-Turing complete blockchain, with transactions finalized through the TEE's updating and broadcasting of signed event messages.

B. Multi-Hop Cases

Thus far, we have only considered user-to-user communication. We now extend SightCVC to longer paths, analogous to how multiple users connect through payment channel networks in virtual channels [16], [17]. Users on either C_1 or C_2 can recursively construct virtual channels on top of two underlying channels (payment channels, virtual channels, or a combination thereof). Consequently, any user can directly establish a virtual channel with the intermediary Ingrid. Unlike the user-to-user scenario, when constructing cross-chain virtual channels in this case, the underlying channels are no longer payment channels but rather the topmost virtual channels. Notably, this construction requires careful timing coordination for channel updates. For instance, if the topmost layer is the n-th layer, users of the n-th layer virtual channel must have sufficient time to update all (virtual/payment) channels from layer 1 to layer $n-1$. This requirement stems from the fact that transactions must be validated on the bottommost payment channel. We therefore propose that virtual channels without expiration constraints can serve as building blocks for any recursive layer in this scenario. Conversely, cross-chain virtual channels with expiration periods may be more suitable for the top layer, given their predefined expiration time, after which all underlying channels must be closed.

C. Serialization Mechanism

SightCVC does not support concurrent updates. To prevent race conditions through serialization mechanisms, we examine three key aspects:

- State Synchronization. SightCVC maintains a version number for each channel's state, initialized to θ_0 and incremented after each update. When updating a channel, participants must provide signed messages containing the current version number. The receiver verifies whether the version number matches the expected value (current version + 1); if not, disputes are resolved by invoking the *Offload* program.
- Time Separation. SightCVC employs a synchronous communication model. To avoid race conditions from simultaneous update requests, SightCVC partitions time

into update time slots organized in rounds. Communication between users can only be initiated during different rounds.

- Mandatory Response. SightCVC requires the sender to wait for time t after sending a message. If no reply is received, the sender transmits a PUNISHREQ message to trigger dispute resolution.

IX. Conclusion

In this paper, we introduce SightCVC, an innovative cross-chain payment protocol designed to tackle the interoperability and scalability challenges of multi-chain transaction processing within heterogeneous blockchain systems. SightCVC leverages virtual channels alongside a sophisticated oracle-based smart contract to ensure compatibility with Turing complete blockchains, enhance privacy in transactions, and significantly improve system throughput. The protocol design prioritizes efficient fund transfers, minimized latency, and reduced transaction costs, while providing robust mechanisms for dispute resolution and ensuring balance safety. Evaluations on Ripple and Ethereum networks demonstrate that SightCVC achieves notable reductions in transaction delay and costs when compared to existing protocols, underlining its efficacy in real-world, multi-chain scenarios. This work advances the field of cross-chain transactions by offering a scalable, secure solution that maintains an ideal balance of privacy, cost-effectiveness, and interoperability.

X. Acknowledgment

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NTU-CCTF.

References

[1] K. Qin, L. Zhou, B. Livshits, and A. Gervais, "Attacking the DeFi ecosystem with flash loans for fun and profit," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2021, pp. 3–32.

[2] J. Hamlin. (Apr. 2022). *Big Investors are Finally Serious About Crypto. But Experienced Talent is Still Scarce*. [Online]. Available: https://www.institutionalinvestor.com/article/2bstmmp9mfkb5eesiwtmo/corner-office/big-investors-are-finally-serious-about-crypto-but-experienced-talent-is-still-scarce

[3] L. Wintermeyer. (2021). *Institutional Money is Pouring Into the Crypto Market and its Only Going to Grow*. [Online]. Available: https://www.forbes.com/sites/lawrencewintermeyer/2021/08/12/institutional-money-is-pouring-into-the-crypto-market-and-its-only-going-to-grow/

[4] M. T. Akçura and K. Altınkemer, "Diffusion models for B2B, B2C, and P2P exchanges and E-speak," *J. Organizational Comput. Electron. Commerce*, vol. 12, no. 3, pp. 243–261, Sep. 2002.

[5] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," *White paper*, vol. 21, no. 2327, p. 4662, 2016.

[6] J. Kwon and E. Buchman, "Cosmos whitepaper," *A Netw. Distrib. Ledgers*, vol. 27, pp. 1–32, Apr. 2019.

[7] H. Tian et al., "Enabling cross-chain transactions: A decentralized cryptocurrency exchange protocol," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 3928–3941, 2021.

[8] J. Davies. (2017). *Web-Based XCAT Tool for Easy ZEC/BTC Atomic Trading*. [Online]. Available: https://github.com/ZcashFoundation/GrantProposals-2017Q4/files/1363993/29.pdf

[9] M. Herlihy, "Atomic cross-chain swaps," in *Proc. ACM Symp. Princ. Distrib. Comput.*, Jul. 2018, pp. 245–254.

[10] G. Malavolta, P. Moreno-Sánchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," Cryptol. ePrint Arch., IACR, Bellevue, WA, USA, Rep. 2018/472, 2018.

[11] S. Thomas and E. Schwartz. (May 2015). *A Protocol for Interledger Payments*. [Online]. Available: https://interledger.org/developers/documents/interledger.pdf

[12] Bloomberg Intelligence. (Aug. 2022). *Multi-Chain Future Likely as Ethereum's Defi Dominance Declines Bloomberg Professional Services*. [Online]. Available: https://www.bloomberg.com/professional/insights/data/multi-chain-future-likely-as-ethereums-defi-dominance-declines/

[13] A. Al-Balaghi. (2022). *A Multichain Approach is the Future of the Blockchain Industry*. [Online]. Available: https://cointelegraph.com/news/a-multichain-approach-is-the-future-of-the-blockchain-industry

[14] C. Decker and R. Wattenhofer, "Bitcoin transaction malleability and MtGox," in *Proc. 19th Eur. Symp. Res. Comput. Secur. Comput. Security (ESORICS)*, Wroclaw, Poland, 2014, pp. 313–326.

[15] X. Jia, Z. Yu, J. Shao, R. Lu, G. Wei, and Z. Liu, "Cross-chain virtual payment channels," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 3401–3413, 2023.

[16] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 106–123.

[17] L. Aumayr et al., "Bitcoin-compatible virtual channels," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 901–918.

[18] B. Liu, P. Szalachowski, and J. Zhou, "A first look into DeFi oracles," in *Proc. IEEE Int. Conf. Decentralized Appl. Infrastructures (DAPPS)*, Aug. 2021, pp. 39–48.

[19] L. Zhou et al., "SoK: Decentralized finance (DeFi) attacks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2023, pp. 2444–2461.

[20] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 270–282.

[21] V. Madathil, S. A. K. Thyagarajan, D. Vasilopoulos, L. Fournier, G. Malavolta, and P. Moreno-Sánchez, "Cryptographic oracle-based conditional payments," Cryptol. ePrint Arch., IACR, Bellevue, WA, USA, Rep. 2022/499, 2022.

[22] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 949–966.

[23] L. Aumayr et al., "Generalized channels from limited blockchain scripts and adaptor signatures," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2021, pp. 635–664.

[24] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 455–471.

[25] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *Proc. 4th Theory Cryptogr. Conf. Theory Cryptogr.*, Netherlands. Cham, Switzerland: Springer, Feb. 2007, pp. 61–85.

[26] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, "Multi-party virtual state channels," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, Darmstadt, Germany. Cham, Switzerland: Springer, 2019, pp. 625–656.

[27] L. Aumayr, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Blitz: Secure multi-hop payments without two-phase commits," in *Proc. 30th USENIX Secur. Symp. (USENIX Secur.)*, Mar. 2021, pp. 4043–4060.

[28] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," in *Proc. Theory Cryptogr. Conf.* Cham, Switzerland: Springer, 2013, pp. 477–498.

[29] Y. Zhang et al., "Anonymous multi-hop payment for payment channel networks," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 1, pp. 476–485, Jan. 2024.

[30] Y. Guo, M. Xu, D. Yu, Y. Yu, R. Ranjan, and X. Cheng, "Cross-channel: Scalable off-chain channels supporting fair and atomic cross-chain operations," *IEEE Trans. Comput.*, vol. 72, no. 11, pp. 3231–3244, Nov. 2023.

[31] Y. Guo et al., "ZKCross: A novel architecture for cross-chain privacy-preserving auditing," in *Proc. 33rd USENIX Secur. Symp. (USENIX Secur.)*, 2024, pp. 6219–6235.

[32] Z. Zhou et al., "BridgeGuard: Checking external interaction vulnerabilities in cross-chain bridge router contracts based on symbolic dataflow analysis," *IEEE Trans. Dependable Secure Comput.*, vol. 22, no. 5, pp. 5798–5812, May 2025.

[33] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. Knottenbelt, "XCLAIM: Trustless, interoperable, cryptocurrency-backed assets," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 193–210.

[34] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1521–1538.
[35] C. Decker and R. Wattenhofer, "A fast and scalable payment network with Bitcoin duplex micropayment channels," in *Proc. 17th Int. Symp. Stabilization, Saf., Secur. Distrib. Syst.*, Edmonton, AB, Canada. Cham, Switzerland: Springer, Aug. 2015, pp. 3–18.
[36] P. Moreno-Sanchez, A. Blue, D. V. Le, S. Noether, B. Goodell, and A. Kate, "DLSAG: Non-interactive refund transactions for interoperable payment channels in Monero," in *Proc. Int. Conf. Financ. Cryptogr. Data Security*, Jun. 2020, pp. 325–345.
[37] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 473–489.
[38] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *Proc. Int. Conf. Financ. Cryptogr. Data Secur.*, Cham, Switzerland, 2019, pp. 508–526.
[39] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. Pietzuch, "Teechain: A secure payment network with asynchronous blockchain access," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, Oct. 2019, pp. 63–79.
[40] P. McCorry, S. Bakshi, I. Bentov, S. Meiklejohn, and A. Miller, "Pisa: Arbitration outsourcing for state channels," in *Proc. Conf. Adv. Financ. Technol.*, 2019, pp. 16–30.
[41] Z. Avarikioti, O. S. T. Litos, and R. Wattenhofer, "Cerberus channels: Incentivizing watchtowers for bitcoin," in *Proc. 24th Int. Conf. Financial Cryptogr. Data Secur.*, Kota Kinabalu, Malaysia. Cham, Switzerland: Springer, Feb. 2020, pp. 346–366.
[42] G. Malavolta, P. Moreno-Sánchez, A. Kate, and M. Maffei, "SilentWhispers: Enforcing security and privacy in decentralized credit networks," IACR, Bellevue, WA, USA, Rep. 2016/1054, 2016.
[43] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," 2017, *arXiv:1709.05748*.
[44] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: Efficient dynamic routing for offchain networks," in *Proc. 15th Int. Conf. Emerg. Netw. Exp. Technol.*, Dec. 2019, pp. 370–381.
[45] C. Egger, P. Moreno-Sanchez, and M. Maffei, "Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 801–815.
[46] E. Heilman, L. AlShenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "TumbleBit: An untrusted bitcoin-compatible anonymous payment hub," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017, pp. 1–15.
[47] F. Gritti et al., "Confusum contractum: Confused deputy vulnerabilities in Ethereum smart contracts," in *Proc. 32nd USENIX Secur. Symp. (USENIX Secur.)*, 2023, pp. 1793–1810.
[48] C. Sendner et al., "Smarter contracts: Detecting vulnerabilities in smart contracts with deep transfer learning," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, May 2023, pp. 1–18.
[49] A. Augusto, R. Belchior, M. Correia, A. Vasconcelos, L. Zhang, and T. Hardjono, "SoK: Security and privacy of blockchain interoperability," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2024, pp. 3840–3865.

Haonan Yang (Student Member, IEEE) is currently pursuing the Ph.D. degree with the Faculty of Data Science, City University of Macau, Macau, China. His research interests include intelligent transportation systems, applied cryptology, and blockchain.

Tianwei Zhang (Member, IEEE) received the bachelor's degree from Peking University in 2011 and the Ph.D. degree from Princeton University in 2017. He is an Assistant Professor with the School of Computer Science and Engineering, Nanyang Technological University. His research focuses on computer system security. He is particularly interested in security threats and defenses in machine learning systems, autonomous systems, computer architecture, and distributed systems.

Zuobin Ying (Member, IEEE) received the Ph.D. degree in computer architecture from Xidian University, Xi'an, China, in 2016. From 2019 to 2021, he was a Research Fellow with Nanyang Technological University, Singapore. Currently, he is an Associate Professor with the Faculty of Data Science, City University of Macau. He is hosting two scientific research projects and works as a CO-PI of an NSFC-FDCT Joint Project. He has published more than 70 research articles, including IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON CLOUD COMPUTING, and IEEE INTERNET OF THINGS JOURNAL. His research interests lie in privacy computing, applied cryptography, and blockchain. He serves as the IEEE Macau Membership Development Chair and the Doctoral Program Coordinator.

Runjie Yang (Student Member, IEEE) received the B.S. degree from Shenzhen University, China, in 2021. She is currently pursuing the Ph.D. degree with the Faculty of Data Science, City University of Macau, China. Her research interests include distributed systems, target recognition, and image dehazing.

Wanlei Zhou (Senior Member, IEEE) received the B.Eng. and M.Eng. degrees in computer science and engineering from Harbin Institute of Technology, Harbin, China, in 1982 and 1984, respectively, the Ph.D. degree in computer science and engineering from The Australian National University, Canberra, Australia, in 1991, and the D.Sc. degree from Deakin University, Australia, in 2002. He is currently the Vice Rector (Academic Affairs) and the Dean of the Faculty of Data Science, City University of Macau, Macau, China. Before joining the City University of Macau, he held various positions, including the Head of the School of Computer Science, University of Technology Sydney, Australia; the Alfred Deakin Professor; the Chair of information technology; the Associate Dean; and the Head of the School of Information Technology, Deakin University. He was a Lecturer with the University of Electronic Science and Technology of China; a System Programmer at HP, MA, USA; a Lecturer with Monash University, Melbourne, Australia; and a Lecturer with the National University of Singapore, Singapore. He has published more than 500 papers in refereed international journals and refereed international conferences proceedings, including many articles in IEEE TRANSACTIONS and journals. His main research interests include security, privacy, and distributed computing.