AMBiPay: Trustless and Scriptless Bidirectional Payment Channels for Account Model-based Cryptocurrencies

Abstract—**Payment Channels (PC), both bidirectional (bi) and unidirectional (uni), offer a promising solution to blockchain scalability issues. Bi-PC is more flexible and versatile than uni-PC, enabling an unbounded number of payments without any Trusted Third Party (TTP) or script. Current scriptless bi-PCs have been shown to work with the Unspent Transaction Output (UTXO) model, but their applicability to the account model remains unknown. This is due to two primary challenges inherent in the account model: (i) *uncertainty* caused by the single-input-single-output and nonce mechanisms that may disrupt transaction flow during payments, and (ii) *weak constraint* introduced by transactions spending directly from account balances, which allows one party to unilaterally perform a fast-finish payment of old states with a negligible amount, rendering the punishment invalid.**

To address the above challenges, we present AMBiPay, the first bi-PC protocol without any TTP or script for the account model. The key idea is to redesign *fork-then-sleepy* channels and propose a *strong constraint* mechanism based on adaptor signatures and time-lock puzzles. While Erwig et al. proved in PKC'21 that constructing adaptor signatures for unique signatures is impossible due to their respective extractability and uniqueness, we overcome this limitation in 2-party scenarios by restricting signature generation and witness selection to a single party. We provide the formal definition of *2-party adaptor signatures with one-party chosen witness* and a security model covering both unique and common signatures. Coupled with the generic construction, this approach ensures the versatility of AMBiPay. We further extend an *optimized verifiable timed signatures* construction, a key component for scriptless, to enhance the efficiency of AMBiPay. These contributions are also of independent interests in the research of adaptor signatures and verifiable timed signatures. Finally, we evaluate AMBiPay and find its time and communication costs comparable to state-of-the-art PC protocols, demonstrating its practicality.

1. Introduction

Blockchain has transformative potential in industries such as finance, healthcare, and supply chain. However, scalability remains a fundamental technical challenge for blockchains like Bitcoin and Ethereum, particularly in terms of cumulative transactions. The limited capacity of blockchain can lead to high transaction fees and slow confirmation times, which hinders the usability of decentralized payments. For example, the current size of Ethereum currently ranges from 500 GB to 1 TB and continues to grow, making it impractical for nodes to sustain operations if it expands to 5 TB [1].

Payment Channels (PC) [2] provide a scalable solution by enabling off-chain transactions, which permits parties to transact with each other off-chain and settle the final state on-chain. This considerably optimizes the blockchain storage, increases the transaction throughput and reduces transaction fees. There exist two types of PC: unidirectional (uni) and bidirectional (bi). Uni-PC supports transactions to only occur in one direction, while bi-PC enables transactions in both directions. This work specifically focuses on bi-PC, which owns distinct advantages (e.g., flexibility and versatility) over uni-PC.

The critical technical aspect of bi-PC is to prevent the theft of coins between the two untrusted parties. One intuitive approach is utilizing Trusted Execution Environment (TEE) [3] at both parties or depending on a third-party arbitration [4], [5] to reach an agreement on the mutually acceptable last balance. However, both methods add trust assumptions that go against the decentralized nature of cryptocurrencies. As a mitigation, Lightning Network [6] and Generalized Channels [7] introduce a punishment mechanism to recover coins in a channel if either party tries to establish an outdated balance. However, it requires a continuous online presence due to relative timelock scripts, and alternative watchtowers reintroduce trust assumptions. Moreover, existing privacy-enhancing currencies like Monero and Mimblewimble may not support such scripts.

Sleepy Channels [8] present a Trusted Third Party (TTP)-free approach for bi-PC. They utilize absolute timelocks instead of relative timelocks to eliminate the need for watchtowers and ensure punishment within a predefined time. Verifiable Timed Signatures (VTS) [15], [16] facilitate absolute timelocks, making Sleepy Channels compatible with scriptless currencies. The collateral and negligible amount mechanisms incentivize fast-finish payments, where the initiator can promptly complete payments if the other party is responsive and posts the required transaction to transfer a negligible amount of coins. Otherwise, lazy-finish payments occur, and the initiator must wait until the predefined time if the other party is unresponsive.

Despite Sleepy Channels are compatible with UTXO-based currencies, their compatibility with account-based currencies is uncertain due to different transaction formats. Unlike UTXO, where transactions are multi-input-multi-output and spend coins from previous transaction outputs, the account model spends from account balances and employs the single-input-single-output format with a nonce mechanism to prevent double-spending [9]. These differences pose considerable challenges for implementing Sleepy Channels on account-based currencies. As such, Aumayr et al. [8] raised the open problem of applying Sleepy Channels into account-based currencies, which is rephrased as:

An interesting open question is the applicability of Sleepy Channels to account-based currencies, as opposed to UTXO-based currencies.

In this work, we answer the above open problem affirmatively. We present AMBiPay, a new bi-PC protocol for account-based currencies that addresses two challenges, denoted as *uncertainty* and *weak constraint*, when applying Sleepy Channels to the account model. Challenge *uncertainty* is caused by the single-input-single-output and nonce mechanisms in the account model, which can result in transaction flow disruption and unexpected fund assignment. Challenge *weak constraint* pertains to the negligible amount mechanism in fast-finish payments. In the UTXO model, transactions spend from previous outputs, however in the account model, transactions spend from account balances. This facilitates a party to promptly transfer a negligible amount to the desired address, launching fast-finish payments of old states even if the other party are unresponsive. Consequently, the punishment mechanism becomes invalid.

1.1. Our Contribution

The main contributions are as follows.

- In AMBiPay, we redesign *fork-then-sleepy* channels for the account model. Based on this redesign, we combine the nonce mechanism, adaptor signatures and Timelock Puzzles (TLP) to establish a *strong constraint* mechanism and effectively resolve the uncertainty challenge. As a result, we circumvent the need for the negligible amount mechanism and preserve the advantages of TTP-free and script independence, collateral as incentive, and timely punishment (see Table 1). We formalize AMBiPay in the Universal Composability (UC) framework and rigorously prove its security.
- The impossibility of constructing adaptor signatures for unique signatures [10] restricts the adaptability and versatility of AMBiPay. This limitation arises from by the extractability of adaptor signatures and uniqueness of unique signatures. Given a hard relation in a unique signature scheme, an adversary can create a secret / public key pair, calculate the pre-signature upon a statement and the signature of a message, and directly extract the witness. This violates the hardness of the relation. To address this issue, we provide a formal definition of *2-party adaptor signatures with one-party chosen witness*, and present a generic construction that accommodates both unique and common signatures.
- While AMBiPay can achieve the compatibility with scriptless currencies by utilizing VTS, the computation for generating the VTS signatures increases linearly. Therefore, we propose an *Optimized Verifiable Timed Signatures* (OVTS) construction with constant computation efficiency. OVTS improves upon VTS by utilizing verifiably encrypted signatures and a one-time key pair during encryption. Unlike VTS, which distributes signatures into multiple shares, OVTS operates directly on the entire signature, while applying the time-lock puzzle only to the one-time key.
- We showcase the viability of our proposals through concrete instantiations of 2-party adaptor signatures and OVTS. These implementations are deployed on Sepolia, an Ethereum testnet, and rigorous simulations are conducted using the MetaMask-Chrome plugin to connect to Sepolia. During these simulations, transactions are deployed, and the overhead is precisely measured. The evaluation shows that the efficiency of AMBiPay aligns closely with that of Sleepy Channels.

1.2. Related Work

Bi-PC. Related bi-PC proposals fall into two categories.

UTXO-based PC. Duplex Channels [11] enable bi-directional channels with absolute timelock script, achieving script independence via VTS. However, the number of payments in Duplex Channels is limited as the channel lifetime decreases with each successive payment. Additionally, closing the channel requires $\log d$ transactions, where d denotes the total number of performed payments. Eltoo [12] sidesteps the punishment mechanism, but it involves special scripts like SIGHASH_NOINPUT and relative timelocks. Teechan [3] is a simple-yet-efficient proposal but it requires both parties to be equipped with TEE. This trusted assumption may not hold in practice and goes against the decentralized nature of currencies.

Watchtower-based proposals, such as Outpost [17], Cerberus Channels [18], FPPW [19], and BlindHub [13], aim to assist parties in channel monitoring and security. However, some of these proposals lack penalties for offline watchtowers, and others require special scripts or substantial deposits. Lightning Network [6] and Generalized Channels [7] are popular UTXO-based bi-PC proposals without watchtowers. Yet, they rely on relative timelock scripts, necessitating timely blockchain monitoring or alternative watchtowers. To address these issues, Sleepy Channels [8] were proposed, which eliminate script dependency and TTP. However, it is unclear how applicable Sleepy Channels are to account-based currencies. As previously discussed, there may be uncertainties and weak constraints when attempting to apply Sleepy Channels to account-based currencies.

Account-based PC. Ethereum is an account-based currency with Turing-complete script language, enabling complex smart contracts. State Channels [14], [20], [21] have been developed to execute these contracts off-chain, which allow for faster and cheaper transactions compared to on-chain executions. However, the dependence on smart contracts raises security concerns, as evidenced by various smart contract vulnerabilities in Ethereum, including the infamous DAO attacks [22]. Furthermore, other account-based currencies like Ripple [23] and Stellar [24] lack pro-

TABLE 1: Comparison among bi-PC protocols. Trustless means no need for a TTP during payments, scriptless implies no involvement of scripts, fast-finish permits fast channel closure without waiting for a specific time, and unrestricted lifetime means there is no pre-specified channel lifetime. #Tx. for closing is the required amount of transactions for the channel closure, with d representing the amount of payments executed in Duplex [11]. We let ✓ denote complete satisfaction, ✓$^{-}$ denote partial satisfaction (e.g., involving a trusted watchtower in trustless, or offering a weak constraint in fast-finish payments), and ✗ denote unsatisfied properties.

Protocol	Model	Trustless	Scriptless	Fast-finish payments	Unrestricted lifetime	#Tx. for closing
Duplex [11]	UTXO	✓	✓	✓	✗	$\log d$
Eltoo [12]	UTXO	✓$^{-}$	✗	✗	✓	2
Teechan [3]	UTXO	✗	✓	✓	✓	1
BlindHub [13]	UTXO	✗	✓	✓	✗	2
Lightning Network [6]	UTXO	✓$^{-}$	✗	✗	✓	1
Generalized Channels [7]	UTXO	✓$^{-}$	✗	✗	✓	2
Sleepy Channels [8]	UTXO	✓	✓	✓*	✗	1
State Channels [14]	Account	✓	✗	✗	✓	1
AMBiPay	Account	✓	✓	✓	✗	1

* Sleepy Channels effectively constrain transactions in UTXO but are weak when applied directly to the account model.

grammable smart contract functionality. This has prompted an exploration of scriptless bi-PC for the account model, facilitating diverse computations without exposing potential security risks associated with smart contracts. To the best of our knowledge, no work currently exists in this field.

Adaptor Signatures. Adaptor signatures [10], have been widely adopted in blockchain applications such as PC networks [25], PC hubs [13], and atomic swaps [26]. Aumayr et al. [7] formalized adaptor signatures as a standalone primitive, and Fournier [27] proposed a weaker definition. Previous works only designed adaptor signatures from Schnorr and ECDSA without generic transformations. Erwig et al. [10] gave a generic construction for specified signatures. However, these approaches lack the ability to construct adaptor signatures from unique signatures, and their security models were confined to the unforgeability of two-party adaptor signatures for common signatures, making them unsuitable for scenarios involving unique signatures. In contrast, our proposed adaptor signatures align with 2-party weak adaptor signatures [28], which restrict one party's witness selection. Despite this alignment, the latter did not explore construction from unique signatures or provide formal security definitions and proofs.

VTS. VTS [15] enables time-locking a signature for a predetermined time interval $\mathbf{T}$. Its verifiability ensures that anyone can check that the signature can be recovered after a sequential computation of time $\mathbf{T}$. Verifiable Timed Discrete Logarithm (VTDL) [26] and Verifiable Timed Linkable Ring Signatures (VTLRS) [16], were proposed to enhance the efficiency and functionality of VTS. These primitives, applied in PC and atomic swaps to eliminate script dependency, suffer from linearly increasing computation, potentially becoming a bottleneck in real-world applications.

2. Solution Overview

This section provides an overview of AMBiPay, initially applying Sleepy Channels [8] to account-based currencies, but noting its limitations. We then present our solution.

Applying Sleepy Channels in Account Model. The intuitive application involves transitioning from the UTXO model to the account model with the nonce mechanism, as depicted in Figure 1. Similar to the original Sleepy Channels in UTXO, parties A and B first lock coins in a shared address Ch_{AB}, referred to as a channel. However, due to the single-input-single-output transaction format of the account model, the parties exchange three payment transactions (tx_R, tx_{Pay}, and tx_{RPay}) to spend the funds from Ch_{AB} and update the balance for both parties. Each party (e.g., A) possesses its own transactions (i.e., tx_R^A, tx_{Pay}^A, and tx_{RPay}^A) to initiate payments unilaterally. Fast-finish and lazy-finish payments are enabled with two extra transactions (tx_{FPay}^A and tx_D^A). In fast-finish payments, if B is responsive, a negligible amount ε is transferred from Exit_A to A's address, enabling the former transaction to take effect immediately. In contrast, if B is unresponsive, the lazy-finish process occurs, requiring A to wait until the timelock $\mathbf{T}_d$ expires before posting tx_D^A.

To ensure security, parties A and B revoke the old payment state during each update via a punishment transaction tx_{Pnsh} that transfers the balance of the malicious party to the honest party. The absolute timelock $\mathbf{T}_d$ set in tx_D^A prevents party A from publishing an old state until $\mathbf{T}_d$. Thus, party B can punish party A by posting tx_{Pnsh}^B before $\mathbf{T}_d$. The approach inherits the advantages of Sleepy Channels, although two limitations need to be addressed.

(i) Uncertainty. The single-input-single-output and nonce mechanisms of account model ensure sequential transaction execution. However, this can result in uncertainty as parties may compete to post their transactions. For instance, after party A posts tx_R^A with nonce "0" to redeem her collateral c, party B can post tx_{Pay}^B and tx_{RPay}^B to obtain $2v_B + c$ coins. If party A posts tx_R^A and tx_{Pay}^A simultaneously, party B can still pre-post tx_{Pay}^B in the pending transaction pool. Chaining tx_{Pay}^B remains highly probable, which can disrupt the bidirectional payment process.

(ii) Weak Constraint. In the original Sleepy Channels, a negligible amount ε is utilized in the fast-finish payment. The negligible amount is spent from exactly B's transaction (identified by transaction hash and output index) and cannot be funded through any other means. This guarantees that

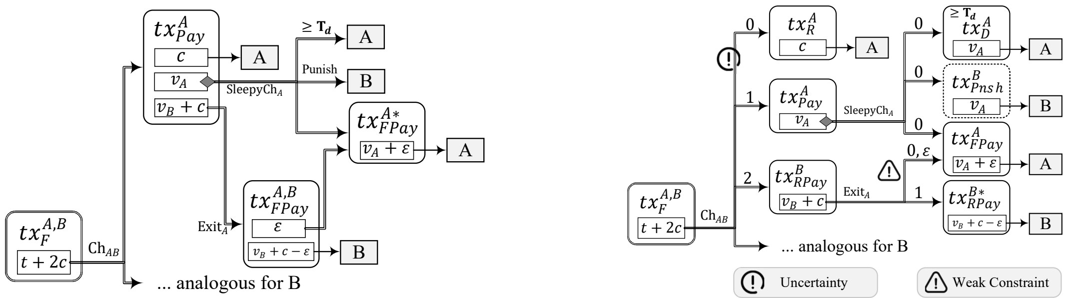

Figure 1: *Left sub-figure (cited from [8])*: Transaction flow of original Sleepy Channels in the UTXO model between parties A and B. *Right sub-figure*: Transaction flow for applying Sleepy Channels in the account model. Rounded boxes represent transactions with input and output arrows. Double lines are shared spending addresses, and single lines are single-party addresses. The diamond shape signifies that only one transaction will be chained. The nonce i on the line denotes the i-th transaction of the address (incremented after each channel closure). For clarity, we display only the first several transactions of each channel in the figure. v_A and v_B are the locked coins of parties A and B, respectively, $t = v_A + v_B$, and c is the prepaid collateral.

one party (e.g., A) can only perform the fast-finish payment after the other party (i.e., B) has redeemed their balance and collateral. In the account model, this mechanism will be weaker, as party A can independently transfer ε coins to the specified address and perform the fast-finish payment, rendering the punishment invalid.

Achieving certainty and strong constraint. We first redesign *fork-then-sleepy* channels (depicted in Figure 4), which create a forking channel FCh_A that combines tx_R^A and tx_{Pay}^A into a single tx_{Pay}^A. From this, two transactions tx_{Pay}^{A*} and tx_R^A are generated and spent from FCh_A to $\mathsf{SleepyCh}_A$ and party A's address, respectively. This mechanism partially solves the uncertainties, as there are still two transactions with nonce "1" after the chained tx_{Pay}^A. The weak constraint between tx_{RPay}^{B*} and tx_{FPay}^A also remains.

We replace the negligible amount mechanism with adaptor signatures and TLPs to achieve a strong constraint. Applying adaptor signatures to tx_{RPay}^{B*} and tx_{FPay}^A with the same witness, initially, party A possesses only the pre-signatures of tx_{FPay}^A and tx_{RPay}^{B*}, and party B has the pre-signature of tx_{RPay}^{B*} and the witness. The adaptability of adaptor signatures allows party B to adapt the signature of tx_{RPay}^{B*} using the witness. Once party B posts the signature of tx_{RPay}^{B*} on-chain, party A can extract the witness (ensured by the extractability) and further adapt the signature of tx_{FPay}^A. This ensures strict sequential execution between tx_{RPay}^{B*} and tx_{FPay}^A, analogous to tx_{RPay}^{A*} and tx_{FPay}^B.

Similarly, to achieve sequential execution between tx_{Pay}^A and tx_{RPay}^B, a time-lock puzzle must be embedded in the witness. Upon obtaining the signature of tx_{Pay}^A from party A, party B can only access the puzzle using the pre-signature and signature of tx_{Pay}^A. Party B must spend the predefined time, typically set as a block generation time, to force-open the puzzle and utilize the recovered witness to obtain the signature of tx_{RPay}^B. Party B can only post tx_{RPay}^B because tx_{Pay}^A with the same nonce as tx_{Pay}^B has already been posted in the blockchain after the block generation time. This prevents party B from posting the signatures of tx_{Pay}^B and tx_{RPay}^B together, as tx_{Pay}^B, sharing the same nonce as tx_{Pay}^A, will be discarded.

Ensuring versatility. Adopter signatures can enforce a strong constraint between transactions, but existing schemes have been proven unfeasible for unique signatures [10]. This limitation hinders the application of AMBiPay in blockchain systems equipped with unique signatures, such as Beacon Chain that employs BLS signatures for aggregatable transactions and practical consensus [29].

We below review the impossibility to construct an adaptor signature scheme for unique signatures shown in [10]. Given a hard relation $(Y; y) \in \mathsf{R}$, an adversary $\mathcal{A}$ creates a secret / public key pair of the unique signature scheme. Then, it calculates the pre-signature upon statement Y and the signature of a message. Due to the extractability of adaptor signatures and uniqueness of unique signatures, $\mathcal{A}$ can directly extract the witness y, which violates the hardness of R. Intuitively, $\mathcal{A}$ will be unable to extract the witness, if we restrict $\mathcal{A}$'s ability to generate signatures. This makes it possible to permit only one party (called as signature holder hereafter) to obtain the final signature in 2-party scenarios [30], [31], [32].

To ensure the versatility of AMBiPay, we propose a generic construction of 2-party adaptor signatures for both unique and common signatures. Our approach is inspired by blinding techniques used in blind or adaptor signature schemes. In our approach, the signature holder (e.g., party A) randomly chooses a blinding key to blind the signature, resulting in a ciphertext that serves as the pre-signature, and the blinding key acts as the witness. Given the pre-signature and witness, the signature can be recovered via a de-blinding operation, satisfying the adaptability requirement.

For extractability, we ensure that the blinding key can be extracted from the ciphertext and message. This can be achieved using trivial blinding, where the blinding key can be extracted from the ciphertext and message via $K = C \oplus M$, given the message M and ciphertext C. For correctness and security, we introduce Non-Interactive Zero-Knowledge (NIZK) proofs for a statement Y along with necessary properties among defined functions, ensuring correct usage of the witness in the pre-signing process.

Considering the witness is chosen by one of the two parties, we term this scheme as *2-party adaptor signature with one-party chosen witness*. Its security model is adapted to cover both unique and common signatures and will be introduced in Section 3.

Optimizing efficiency. Applying VTS [15] in AMBiPay can eliminate the absolute timelock, but the VTS will cause high latency during payments. In VTS, the prover needs to generate $O(n)$ shares of public keys, signatures, and puzzles, which results in a linear increasing computation cost to the number of shares (i.e., n) [1]. Inspired by verifiably encrypted signatures [33] and VTS [15], we propose an OVTS construction, where the entire signature is encrypted using a one-time key pair, and the time-lock puzzle is applied only to the one-time key. NIZK proofs are employed to ensure verifiability. By force-opening the puzzle, one can obtain the secret key and further recover the signature. OVTS satisfies both efficiency and compatibility. Efficiency refers to the reduced computation cost, which is a constant $O(1)$. Compatibility indicates that OVTS can be used with any signature schemes such as ECDSA, BLS, and Schnorr.

3. Preliminaries

The notation $a \in \mathbb{S}$ is denoted to uniformly sample an element a from the set $\mathbb{S}$. We denote $\lambda \in \mathbb{N}$ as a security parameter, and $b \leftarrow \mathsf{f}(a)$ as a Probabilistic Polynomial Time ($\mathcal{PPT}$) algorithm f that takes as input a, and outputs b. We also denote $b := \mathsf{f}(a)$ if f is with Deterministic Polynomial Time ($\mathcal{DPT}$). The notation $\mathsf{negl} : \mathbb{N} \rightarrow \mathbb{R}$ is called a negligible function in λ if $\forall k \in \mathbb{N}, \exists \lambda_0 \in \mathbb{N}$ such that $\forall \lambda \geq \lambda_0, |\mathsf{negl}(\lambda)| \leq 1/\lambda^k$. EUF-CMA and IND-CPA are desirable security properties for digital signature and public key encryption schemes, respectively. They ensure Existential UnForgeability under Chosen Message Attacks and INDistinguishability under Chosen Plaintext Attacks.

Account Model. Account model and UTXO model are two main transaction types of blockchain. In the account model, the ledger $\mathbb{L}$ maintains a world state for each account $ws_i = (pk_i, v_i, n_i)$, where pk_i is the unique address, v_i is the account balance, and n_i is the number of executed transactions, which prevents double-spending and maintains sequential execution of transactions. If user i wishes to transfer v coins to user k, user i needs to post a transaction $tid_{i,n_i+1} = tx(pk_i, pk_k, v, n_i + 1)$ and associated signature σ_{i,n_i+1}. Then, the user i's world state is updated as $ws_i = (pk_i, v_i - v, n_i + 1)$, and accordingly $ws_k = (pk_k, v_k + v, n_k)$ updated for user k. This implies that a transaction in the account model is single-input-single-output. The UTXO supports multi-input-multi-output transactions, ensuring that when a transaction is posted on-chain, multiple transaction outputs take effect simultaneously.

1. VTDL [26] also face this issue, despite being more efficient than VTS by committing the signing key instead of the signature itself. VTDL only supports the one-time use of signing keys, making it unsuitable for the account model with reused requirements.

Universal Composability (UC). To model security in concurrent execution scenarios, we employ the UC framework with global setup [8]. This involves a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ executing the protocol, with a static adversary $\mathcal{A}$ declaring the upfront corrupted parties. The environment $\mathcal{E}$ captures any external events outside the protocol. Synchronous communication is modeled by a global clock $\mathcal{F}_{clock}$, and communication between users is authenticated and ensured with delivery by $\mathcal{F}_{GDC}$. We denote real protocol execution (Π and $\mathcal{A}$) as $EXEC_{\Pi,\mathcal{A},\mathcal{E}}$, and ideal functionality execution ($\mathcal{F}$ and $\mathcal{S}$) as $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$.

Definition 1 ***(Universal Composability)***. *A protocol Π is the UC-realization of an ideal functionality $\mathcal{F}$ if, $\forall \mathcal{PPT}$ adversary $\mathcal{A}$, there is a simulator $\mathcal{S}$ satisfying that the ensembles $EXEC_{\Pi,\mathcal{A},\mathcal{E}}$ and $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable for any $\mathcal{E}$.*

NIZK. A NIZK proof system [34] enables a prover to prove the validity of a relation R to a verifier with a single message, without disclosing the witness. A public common reference string is initialized by a setup algorithm $crs \leftarrow \mathsf{NIZK.Setup}(\lambda, \mathsf{R})$ on input security parameter λ and relation R. The proof is generated by a proving algorithm $\pi \leftarrow \mathsf{NIZK.Prove}(crs, x, w)$ using an instance / witness pair (x, w). The verifier checks the validity of π via the verification algorithm $b := \mathsf{NIZK.Verf}(crs, x, \pi)$, where $b = 1$ indicates valid and $b = 0$ invalid. A NIZK proof system should fulfill completeness and soundness, and maintain zero-knowledge property by not disclosing any extra information beyond the validity of R.

TLP. A time-lock puzzle scheme TLP = (Setup, PGen, PSolve) [35] empowers concealing a value for a specific period, strictly greater than $\mathbf{T} \in \mathbb{N}$. PGen is a probabilistic algorithm that takes the hardness parameter $\mathbf{T}$, a solution $s \in \{0, 1\}^*$, and random coins r as input and outputs a puzzle z. PSolve, the solving algorithm, takes a puzzle z as input and recovers a solution s. The security requirement is that no Parallel Random Access Machines ($\mathcal{PRAM}$, which is a model considered for most of the parallel algorithms) adversary $\mathcal{A}$ with running time $\leq \mathbf{T}^\epsilon(\lambda)$ can distinguish between two puzzles generated with solutions $(s_0, s_1) \in \{0, 1\}^2$, both with timing hardness $\mathbf{T}$, except with negligible probability ϵ.

VTS. A VTS scheme VTS = (Setup, Commit-and-Prove, Verf, Open, and ForceOpen) [15] enables time-locking a signature on a specific message for a predetermined time interval $\mathbf{T}$. The Setup algorithm generates a public parameter pp. The Commit-and-Prove algorithm commits a message/signature pair (m, σ) under a public key pk for a hiding time $\mathbf{T}$ and produces a commitment c_{vts} and a proof π. The Verf algorithm verifies the validity of a commitment c_{vts} and its embedded signature for a given message m and public key pk. The Open algorithm reveals the committed signature σ and the randomness r used in the commitment. The ForceOpen algorithm directly outputs a signature σ from a commitment c_{vts}.

A secure VTS scheme must satisfy two security properties: (i) *soundness*, ensuring that the VTS.ForceOpen algorithm will correctly output the committed σ upon commitment c_{vts}, and (ii) *privacy*, guaranteeing that the probability of any $\mathcal{PRAM}$ algorithm extracting σ from the commitment c_{vts} within time t (where $t < \mathbf{T}$) is negligible.

4. Generic Construction of 2-Party Adaptor Signatures with One-party Chosen Witness

This primitive aligns with 2-party weak adaptor signatures [28], limiting the witness selection to one party. We also provide formal definitions of correctness and security, covering unforgeability, adaptability, and extractability for both unique and common signatures.

4.1. Definition

Definition 2 ***(2-Party Adaptor Signature with One-party Chosen Witness)****. This primitive is defined via a hard relation* R *and 2-party digital signatures with aggregatable public keys* $\Pi_{\mathsf{DS}} = (\mathsf{Setup}, \mathsf{KG}, \Gamma_{\mathsf{AKG}}, \Gamma_{\mathsf{Sign}}, \mathsf{Verf})$. *It involves two interactive parties and a tuple* aSIG_2 = (Γ_{pSign}, pVerf, Adapt, Ext) *of the following algorithms and protocols.*

- $(\widehat{\sigma}, Y_i, \pi_i) \leftarrow \Gamma_{\mathsf{pSign}_{\langle sk_i, y_i; sk_{1-i}\rangle}}(pk_0, pk_1, m)$. *This $\mathcal{PPT}$ presigning protocol is executed by two parties via inputting their secret / public key pairs (sk_i, pk_i) with $i \in \{0,1\}$, a message $m \in \{0,1\}^*$ and a witness y_i chosen by one party of them. It finally outputs a pre-signature $\widehat{\sigma}$, a statement Y_i, and a proof π_i of $(Y_i; y_i) \in \mathsf{R}$.*
- $\{0,1\} := \mathsf{pVerf}_{pk}(Y_i, m, \widehat{\sigma}, \pi_i)$. *This $\mathcal{DPT}$ verification algorithm inputs a statement Y_i, a message $m \in \{0,1\}^*$, a pre-signature $\widehat{\sigma}$, and a proof π_i. It outputs 1 if $\widehat{\sigma}$ and π_i are valid, and 0 otherwise.*
- $\sigma := \mathsf{Adapt}_{pk}(\widehat{\sigma}, y_i)$. *This $\mathcal{DPT}$ adaptor algorithm inputs a pre-signature $\widehat{\sigma}$ and a witness y_i, and outputs the signature σ.*
- $y_i := \mathsf{Ext}_{pk}(\sigma, \widehat{\sigma})$. *This $\mathcal{DPT}$ extraction algorithm inputs a signature σ and a pre-signature $\widehat{\sigma}$, and outputs the witness y_i.*

We define a secure 2-party adaptor signature scheme with one-party chosen witness, denoted as aSIG_2, by presenting the following properties. The adversary can query the honest relation oracle $\mathcal{O}_H$, the corrupted relation oracle $\mathcal{O}_C$, the signing oracle $\mathcal{O}_S$, and the pre-signing oracle $\mathcal{O}_{pS}$ (see Figure 2) in the formal presentation of these properties.

Definition 3 ***(2-Party Pre-signature Correctness)****. The property holds for all $\lambda \in \mathbb{N}$ and messages $m \in \{0,1\}^*$, if the following equation is satisfied.*

$$\Pr\left[\begin{array}{l} pp \leftarrow \mathsf{Setup}(\lambda), (sk_0, pk_0) \leftarrow \mathsf{KG}(pp), \\ (sk_1, pk_1) \leftarrow \mathsf{KG}(pp), pk := \Gamma_{\mathsf{AKG}}(pk_0, pk_1), \\ (\widehat{\sigma}, Y_0, \pi_0) \leftarrow \Gamma_{\mathsf{pSign}_{\langle sk_0, y_0; sk_1\rangle}}(pk_0, pk_1, m), \\ \sigma := \mathsf{Adapt}_{pk}(\widehat{\sigma}, y_0), y_0' := \mathsf{Ext}_{pk}(\sigma, \widehat{\sigma}) : \\ \mathsf{pVerf}_{pk}(Y_0, m, \widehat{\sigma}, \pi_0) = 1 \wedge \\ \mathsf{Verf}_{pk}(m, \sigma) = 1 \wedge (Y_0; y_0') \in \mathsf{R} \end{array}\right] = 1.$$

Definition 4 ***(2-aEUF-CMA Security)****. This property holds when* $\Pr[\mathsf{aSigForge}^b_{\mathcal{A},\mathsf{aSIG}_2}(\lambda) = 1] \le \mathsf{negl}(\lambda)$ *for any $\mathcal{PPT}$ adversary $\mathcal{A}$, where the experiment* $\mathsf{aSigForge}$ *is defined as follows.*

$\underline{\mathsf{aSigForge}^b_{\mathcal{A},\mathsf{aSIG}_2}(\lambda):}$
$\mathcal{Q}_H, \mathcal{Q}_C, \mathcal{Q}_S, \mathcal{Q}_{pS} := \emptyset, pp \leftarrow \mathsf{Setup}(\lambda)$
$(sk_{1-b}, pk_{1-b}) \leftarrow \mathsf{KG}(pp)$
$(sk_b, pk_b) \leftarrow \mathcal{A}(pp, pk_{1-b})$
$(m^*, Y^* \in \mathcal{Q}_H) \leftarrow \mathcal{A}^{\mathcal{O}_H, \mathcal{O}_C, \mathcal{O}_{\Gamma_S}, \mathcal{O}_{\Gamma_{pS}}}(pk_{1-b}, sk_b, pk_b)$
retrieve $(Y^*; y^*) \leftarrow \mathcal{Q}_H$
$(\widehat{\sigma}, Y^*, \pi^*) \leftarrow \Gamma_{\mathsf{pSign}_{\langle sk_{1-b}, y^*; \cdot\rangle}}(pk_0, pk_1, m^*)$
$\sigma^* \leftarrow \mathcal{A}^{\mathcal{O}_H, \mathcal{O}_C, \mathcal{O}_{\Gamma_S}, \mathcal{O}_{\Gamma_{pS}}}(\widehat{\sigma}, Y^*, \pi^*), pk := \Gamma_{\mathsf{AKG}}(pk_0, pk_1)$
return $(m^* \notin \mathcal{Q}_S \wedge Y^* \in \mathcal{Q}_H \wedge \mathsf{Verf}_{pk}(m^*, \sigma^*))$

Definition 5 ***(2-Party Pre-Signature Adaptability)****. This property means that if for all $\lambda \in \mathbb{N}$, messages $m \in \{0,1\}^*$, public keys pk_0 and pk_1, statement Y, and pre-signature tuples $(\widehat{\sigma}, \pi)$, the equation* $\mathsf{pVerf}_{pk}(Y, m, \widehat{\sigma}, \pi) = 1$ *holds where* $pk := \Gamma_{\mathsf{AKG}}(pk_0, pk_1)$. *Then, it follows that* $\Pr[\mathsf{Verf}_{pk}(m, \mathsf{Adapt}_{pk}(\widehat{\sigma}, y)) = 1] = 1$.

Definition 6 ***(2-Party Witness Extractability)****. This property holds when* $\Pr[\mathsf{aWitExt}^b_{\mathcal{A},\mathsf{aSIG}_2}(\lambda) = 1] \le \mathsf{negl}(\lambda)$ *for any $\mathcal{PPT}$ adversary $\mathcal{A}$, where the experiment* $\mathsf{aWitExt}$ *is defined as follows.*

$\underline{\mathsf{aWitExt}^b_{\mathcal{A},\mathsf{aSIG}_2}(\lambda):}$
$\mathcal{Q}_H, \mathcal{Q}_C, \mathcal{Q}_S, \mathcal{Q}_{pS} := \emptyset, pp \leftarrow \mathsf{Setup}(\lambda)$
$(sk_{1-b}, pk_{1-b}) \leftarrow \mathsf{KG}(pp)$
$(sk_b, pk_b) \leftarrow \mathcal{A}(pp, pk_{1-b})$
$(m^*, Y^* \in \mathcal{Q}_H \cup \mathcal{Q}_C) \leftarrow \mathcal{A}^{\mathcal{O}_H, \mathcal{O}_C, \mathcal{O}_{\Gamma_S}, \mathcal{O}_{\Gamma_{pS}}}(pk_{1-b}, sk_b, pk_b)$
retrieve $(Y^*; y^*) \leftarrow \mathcal{Q}_H \cup \mathcal{Q}_C$
$(\widehat{\sigma}, Y^*, \pi^*) \leftarrow \Gamma_{\mathsf{pSign}_{\langle sk_{1-b}, y^*; \cdot\rangle}}(pk_0, pk_1, m^*)$
$\sigma^* \leftarrow \mathcal{A}^{\mathcal{O}_H, \mathcal{O}_C, \mathcal{O}_{\Gamma_S}, \mathcal{O}_{\Gamma_{pS}}}(\widehat{\sigma}, Y^*, \pi^*)$
$y' := \mathsf{Ext}_{pk}(\sigma^*, \widehat{\sigma}), pk := \Gamma_{\mathsf{AKG}}(pk_0, pk_1)$
return $(m^* \notin \mathcal{Q}_S \wedge (Y^*; y') \notin \mathsf{R} \wedge \mathsf{Verf}_{pk}(m^*, \sigma^*))$

Remark. Our security model differs from [10] due to the one-party chosen witness, specifically in the $\mathsf{aSigForge}$ game. In [10], the adversary attempts to forge a valid signature σ^* for a challenged statement Y^* and a pre-signature $\widehat{\sigma}$. The adversary can query the signing oracle $\mathcal{O}_{\Gamma \mathsf{S}}$ and the pre-signing oracle $\mathcal{O}_{\Gamma \mathsf{pS}}$ but is restricted from querying the challenged message m^*. While this notion is valid for common signatures, it becomes invalid for unique signatures. The adversary can query a different message m' along with Y^* to the oracles, extract the witness y^* based on uniqueness, and adapt it to forge the signature σ^* for the original challenged message m^*. Our chosen-witness approach introduces adjustments, allowing the adversary to query honest and corrupted relation oracles. However, the challenged statement Y^* must remain uncorrupted. This ensures the security of our approach, even in the context of unique signatures.

$$
\begin{array}{lll}
\underline{\mathcal{O}_H(pp)} & \underline{\mathcal{O}^b_{\Gamma_S}(m)} & \underline{\mathcal{O}^b_{\Gamma_{pS}}(m, Y_i)} \\
(Y_i; y_i) \in \mathsf{RG}(pp) & \textbf{retrieve } (m, Y_i) \leftarrow \mathcal{Q}_{pS} & \textbf{if } Y_i \in \mathcal{Q}_H \cup \mathcal{Q}_C \textbf{ then} \\
\mathcal{Q}_H := \mathcal{Q}_H \cup \{(Y_i; y_i)\} & \forall i, \mathcal{Q}_H := \mathcal{Q}_H \backslash \{(Y_i; y_i)\} & \textbf{retrieve } (Y_i; y_i) \leftarrow \mathcal{Q}_H \cup \mathcal{Q}_C \\
\textbf{return } Y_i & \mathcal{Q}_C := \mathcal{Q}_C \cup \{(Y_i; y_i)\} & \mathcal{Q}_{pS} := \mathcal{Q}_{pS} \cup \{(m, Y_i)\} \\
 & \mathcal{Q}_S := \mathcal{Q}_S \cup \{m\} & (\widehat{\sigma}, Y_i, \pi_{1-b}) \leftarrow \Gamma_{\mathsf{pSign}_{\langle sk_{1-b}, y_i; \cdot \rangle}}(pk_0, pk_1, m) \\
\underline{\mathcal{O}_C(Y_i)} & \sigma \leftarrow \Gamma_{\mathsf{Sign}_{\langle sk_{1-b}; \cdot \rangle}}(pk_0, pk_1, m) & \textbf{endif} \\
\textbf{if } Y_i \in \mathcal{Q}_H \textbf{ then} & \textbf{return } \sigma & \textbf{if } m \in \mathcal{Q}_S \textbf{ then} \\
\mathcal{Q}_H := \mathcal{Q}_H \backslash \{(Y_i; y_i)\} & & \mathcal{Q}_H := \mathcal{Q}_H \backslash \{(Y_i; y_i)\} \\
\mathcal{Q}_C := \mathcal{Q}_C \cup \{(Y_i; y_i)\} & & \mathcal{Q}_C := \mathcal{Q}_C \cup \{(Y_i; y_i)\} \\
\textbf{endif} & & \textbf{endif} \\
\textbf{return } y_i & &
\end{array}
$$

Figure 2: Definition of involved oracles.

4.2. Construction

Our construction is based on a 2-party signature protocol $\Pi_{\mathsf{DS}} = (\mathsf{Setup}, \mathsf{KG}, \Gamma_{\mathsf{AKG}}, \Gamma_{\mathsf{Sign}}, \mathsf{Verf})$. Here, Setup, KG, and Verf algorithms are inherited from traditional digital signatures, and $\Gamma_{\mathsf{AKG}}, \Gamma_{\mathsf{Sign}}$ are 2-party interactive protocols for jointly generating the aggregated public key and signature, respectively. Γ_{Sign} inputs the message m and two shared public / secret key pairs (pk_0, sk_0) and (pk_1, sk_1), and it outputs a valid signature σ to one of the two parties, namely $\mathsf{Verf}(\Gamma_{\mathsf{AKG}}(pk_0, pk_1), m, \sigma) = 1$. We below formalize other functions used in our construction.

- For pre-signing, we define a blinding function $f_{\mathsf{bnd}} : \mathbb{D}_{\mathsf{m}} \times \mathbb{D}_{\mathsf{w}} \to \mathbb{D}_{\mathsf{c}}$ and a stating function $f_{\mathsf{state}} : \mathbb{D}_{\mathsf{pp}} \times \mathbb{D}_{\mathsf{w}} \to \mathbb{D}_{\mathsf{state}}$. The blinding function inputs a message $m \in \mathbb{D}_{\mathsf{m}}$ and a witness $y \in \mathbb{D}_{\mathsf{w}}$, and it outputs a ciphertext $c \in \mathbb{D}_{\mathsf{c}}$. The stating function inputs a public parameter $pp \in \mathbb{D}_{\mathsf{pp}}$ and a witness $y \in \mathbb{D}_{\mathsf{w}}$, and it outputs a statement $Y \in \mathbb{D}_{\mathsf{state}}$ satisfying that $(Y; y) \in \mathsf{R}$.
- For verification, we define a shifting function $f_{\mathsf{shift}} : \mathbb{D}_{\mathsf{pk}} \times \mathbb{D}_{\mathsf{m}} \times \mathbb{D}_{\mathsf{ps}} \to \mathbb{D}_{\mathsf{state}}$ that inputs a public key $pk \in \mathbb{D}_{\mathsf{pk}}$ and a message / pre-signature pair $(m, \widehat{\sigma}) \in (\mathbb{D}_{\mathsf{m}}, \mathbb{D}_{\mathsf{ps}})$, and outputs a statement $Y \in \mathbb{D}_{\mathsf{state}}$.
- For adaptation, we define a de-blinding function $f_{\mathsf{debnd}} : \mathbb{D}_{\mathsf{c}} \times \mathbb{D}_{\mathsf{w}} \to \mathbb{D}_{\mathsf{m}}$ that inputs a ciphertext $c \in \mathbb{D}_{\mathsf{c}}$ as well as a witness $y \in \mathbb{D}_{\mathsf{w}}$, and it outputs the message $m \in \mathbb{D}_{\mathsf{m}}$.
- For extracting witness, we define an extraction function $f_{\mathsf{ext}} : \mathbb{D}_{\mathsf{c}} \times \mathbb{D}_{\mathsf{m}} \to \mathbb{D}_{\mathsf{w}}$ that inputs a ciphertext $c \in \mathbb{D}_{\mathsf{c}}$ together with a message $m \in \mathbb{D}_{\mathsf{m}}$, and it outputs the witness $y \in \mathbb{D}_{\mathsf{w}}$.

Our generic construction Π_{AS} = (Setup, KG, Γ_{AKG}, Γ_{Sign}, Verf , Γ_{pSign}, pVerf, Adapt, Ext) is presented in Figure 3. The algorithms KG, Γ_{AKG}, Γ_{Sign}, and Verf are inherited from Π_{DS}, and Setup also generates a crs for the NIZK proof system via invoking NIZK.Setup.

For Π_{AS} to be a 2-party adaptor signature scheme, the properties of *hiding*, *consistency* and *extractability* must be satisfied by f_{bnd}, f_{debnd}, f_{ext}, f_{state}, and f_{shift}. The *hiding* property implies that f_{bnd} can hide the message and the witness. The *consistency* property is twofold: 1) the pre-signature can be recovered to the signature with the same witness y, implying the adaptability of Π_{AS}; 2) the recovered statements from f_{shift} and f_{state} must be consistent, indicating that the witness is indeed used as the blinding key in the pre-signature. Formally, $\forall pk, \forall \sigma \in \mathbb{D}_{\mathsf{m}}, \forall y \in \mathbb{D}_{\mathsf{w}}$, the following two equations must hold.

$$
\begin{array}{ll}
\underline{\Gamma_{\mathsf{pSign}\langle sk_0, y_0; sk_1 \rangle}(pk_0, pk_1, m)} & \underline{\mathsf{pVerf}_{pk}(Y_0, m, \widehat{\sigma}, \pi_0)} \\
\sigma \leftarrow \Gamma_{\mathsf{sign}\langle sk_0, sk_1 \rangle}(pk_0, pk_1, m) & b := \mathsf{NIZK.Verf}(crs, Y_0, \pi_0) \\
\widehat{\sigma} \leftarrow f_{\mathsf{bnd}}(\sigma, y_0) & Y_0' := f_{\mathsf{shift}}(pk, m, \widehat{\sigma}) \\
Y_0 := f_{\mathsf{state}}(pp, y_0) & \textbf{return } b \wedge (Y_0' == Y_0) \\
\pi_0 \leftarrow \mathsf{NIZK.Prove}(crs, Y_0, y_0) & \\
\textbf{return } (\widehat{\sigma}, Y_0, \pi_0) & \\
 & \\
\underline{\mathsf{Adapt}_{pk}(\widehat{\sigma}, y_0)} & \underline{\mathsf{Ext}_{pk}(\sigma, \widehat{\sigma})} \\
\sigma := f_{\mathsf{debnd}}(\widehat{\sigma}, y_0) & y_0 := f_{\mathsf{ext}}(\widehat{\sigma}, \sigma) \\
\textbf{return } \sigma & \textbf{return } y_0
\end{array}
$$

Figure 3: $\mathsf{aSIG}_2^{\mathsf{G}}$: Generic 2-party adaptor signatures.

$$\sigma := f_{\mathsf{debnd}}(f_{\mathsf{bnd}}(\sigma, y), y), \tag{1}$$

$$f_{\mathsf{state}}(pp, y) = f_{\mathsf{shift}}(pk, m, f_{\mathsf{bnd}}(\sigma, y)). \tag{2}$$

The *extractability* property refers to that $f_{\mathsf{ext}}(\cdot, \sigma)$ and $f_{\mathsf{bnd}}(\sigma, \cdot)$ are inverse for any $\sigma \in \mathbb{D}_{\mathsf{m}}$. Formally, $\forall \sigma \in \mathbb{D}_{\mathsf{m}}, \forall y \in \mathbb{D}_{\mathsf{w}}$, we have

$$y := f_{\mathsf{ext}}(f_{\mathsf{bnd}}(\sigma, y), \sigma). \tag{3}$$

Intuitively, the adversary cannot generate the signature from a pre-signature $\widehat{\sigma}$ on the challenged message m^* and statement Y^*, as the witness y^* is determined by the challenger. Thus, the adversary cannot extract the witness to break the hardness of the relation, resolving the impossibility of constructing adaptor signatures for unique signatures. Below we further show the generic construction in Figure 3 is a 2-party adaptor signature scheme if functions f_{bnd}, f_{debnd}, f_{ext}, f_{state} and f_{shift} satisfy Equations 1, 2, and 3.

Theorem 1 *Assume that* Π_{DS} *is an* EUF-CMA *2-party signature scheme,* f_{bnd}*,* f_{debnd}*,* f_{ext}*,* f_{state} *and* f_{shift} *satisfy Equations 1, 2, and 3,* R *is a hard relation, and* NIZK = (Setup, Prove, Verf) *is a NIZK proof system for* R*. Then the resulting* Π_{AS} *is secure.*

Proof. The proof of this theorem refers to proving the correctness, unforgeability, adaptability, and extractability of $\mathsf{aSIG}_2^{\mathsf{G}}$, which are shown as several lemmas in Appendix A.1.

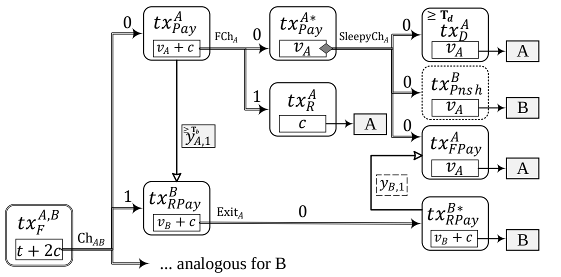

Figure 4: Transaction flow of AMBiPay. Three types are involved: (i) a double solid arrow for a shared spending address with the nonce i; (ii) a single solid arrow for a single-party address; and (iii) a single hollow arrow for the constraint condition of a transaction opened by a witness. $y_{A,1}$ is a full-line witness owned by A, and $y_{B,1}$ is a dotted-line witness owned by B. The notation $\geq \mathbf{T}$ on the upper left corner implies that the value in the box can only be force-open after the predefined timelock $\mathbf{T}$. Collateral c is prepaid by both parties and $v_A + v_B = t$.

Instantiations. To make $f_{\mathsf{bnd}}, f_{\mathsf{debnd}}, f_{\mathsf{ext}}, f_{\mathsf{state}}$ and f_{shift} more intuitive, we provide three instantiation tuples of them for ECDSA, BLS and Schnorr signatures. The instantiation details are presented in Appendix A.2.

Optimization via Offline / Online. The computation of $Y_0 := f_{\mathsf{state}}(pp, y_0)$ and $\pi_0 \leftarrow \mathsf{NIZK.Prove}(crs, Y_0, y_0)$ is costly, but it does not require any message m or signature σ. Hence, these operations can be pre-computed in an offline phase. Upon receiving σ, only the online generation of pre-signature $\widehat{\sigma} \leftarrow f_{\mathsf{bnd}}(\sigma, y_0)$ is necessary. For BLS and Schnorr signatures, the pre-signing phase does not involve any message or signature for proving $(Y; y) \in \mathsf{R}$. However, in the case of ECDSA, it depends on the specific design of Γ_{sign}. In our implementation, we use the 2-party ECDSA signature scheme proposed by Lindell [32]. This optimization achieves the efficient computation of K in the instantiated 2-party ECDSA adaptor signature scheme during the execution of Γ_{sign}, rather than being recovered from the message m and signature σ.

5. AMBiPay: Our Bi-Directional Payment Channel Protocol

AMBiPay establishes an open payment channel Ch_{AB} shared by parties A and B using a secret key sk_{AB}. Payments are bidirectional and off-chain, with only the final payment state published on the blockchain. AMBiPay includes two off-chain types of payments (Payment and Payment Revocation) and two on-chain types (Channel Closing and Revoked Payments Punishment).

5.1. High-Level Description

This subsection provides an overview of AMBiPay, depicted in Figure 4, and its formal design is detailed in Figure 5 and Figure 6.

Strong Constraint. To enforce sequential transaction execution in AMBiPay, we utilize the nonce mechanism of the account model along with TLPs and 2-party adaptor signatures. When two transactions, like tx_{Pay}^{A} and tx_{Pay}^{B}, spend from the same channel with the same nonce, TLPs come into play. Party A generates a TLP $z_{A,1}$ of her chosen witness $y_{A,1}$ upon a predetermined timelock $\mathbf{T}_b$ (typically a block generation time). Parties A and B then collaboratively create pre-signatures for transactions tx_{Pay}^{A} and tx_{RPay}^{B} using $z_{A,1}$ and $y_{A,1}$ respectively. Party A exclusively possesses the puzzle $z_{A,1}$ to adapt the signature of tx_{Pay}^{A}.

The extractability of adaptor signatures ensures that party B can extract $z_{A,1}$ from the signature and pre-signature of tx_{Pay}^{A} only if party A posts the signature on the chain. By force-opening $z_{A,1}$ with the predefined $\mathbf{T}_b$, party B can recover $y_{A,1}$ and adapt the signature of tx_{RPay}^{B}. Similarly, for tx_{FPay}^{A} and tx_{RPay}^{B*}, both pre-signatures directly embed $y_{B,1}$ without TLPs due to different nonces.

The above combination of TLPs and adaptor signatures involves convertibility between puzzle and witness fields, and a NIZK proof to validate puzzle creation. For instance, to match 2-party ECDSA or Schnorr adaptor signatures, timed commitment [35] can be chosen, which generates the puzzle as a AES ciphertext. Thus, the convertibility is ensured by choosing an adequate bit length for the modulus q to fit the AES ciphertext. The NIZK proof is realized using Σ-protocol [37] and Fiat-Shamir transformation [41].

Upon the realized strong constraint, only tx_{RPay}^{B} will be valid after party A publishes tx_{Pay}^{A} and the associated signature on-chain. Similarly, only tx_{RPay}^{A} will be valid after tx_{Pay}^{B} and its associated signature are published on-chain. Furthermore, tx_{FPay}^{A} can only be valid after tx_{RPay}^{B*} and its associated signature are posted on the chain, and tx_{FPay}^{B} can only be valid after tx_{RPay}^{A*} and its associated signature are posted on the chain. This ensures strict sequential execution of transactions, preserving the security of the channel.

Incentive Mechanism. The collateral incentive mechanism [8] in AMBiPay encourages prompt responses and facilitates fast payments. The collateral volume c can be negotiated based on the parties' trust level. When parties fully trust each other, c can be set to 0, which is ideal for one-directional payments where the receiver's address is newly created. In cases of extreme distrust, c is set to at least $v_A - v_B$ to ensure that party B must lock a matching amount of coins in Exit_A as party A does in $\mathsf{SleepyCh}_A$. That is, to prevent party A from spending her coins before the timeout $\mathbf{T}_d$, party B is also restricted from spending the same or more coins until $\mathbf{T}_d$.

5.2. Security

Intuitively, AMBiPay achieves secure and sequential transaction execution by combining the nonce mechanism, TLPs, and 2-party adaptor signatures. These components ensure predefined transaction order, addressing uncertainty and weak constraints. The collateral mechanism incentivizes cooperation and fast channel finalization. Punishment deters dishonest behavior by revoking old payment states. Script-

Parties A and B create a shared address Ch_{AB} with volume $t+2c$, and the associated secret key is shared as $sk^A_{\mathsf{Ch}_{AB}}$ and $sk^B_{\mathsf{Ch}_{AB}}$ for A and B respectively. The notation t represents the total assignable volume of Ch_{AB} and c is the collateral prepaid by each party. To refund Ch_{AB}, A owns the transaction $tx^A_{rf} := (tx^{A,0}_{rf}, tx^{A,1}_{rf})$ and the associated credential $\pi^A_{rf} := (\widehat{\sigma}^{A,0}_{rf}(Z_A), z_A, \widehat{\sigma}^{B,0}_{rf}(Z_B), \widehat{\sigma}^{A,1}_{rf}(Y_B))$, where $tx^{A,1}_{rf} := (\mathsf{Ch}_{AB}, pk_A, v_A + c, 0), tx^{A,2}_{rf} := (\mathsf{Ch}_{AB}, pk_A, v_A + c, 1)$. Analogously, B owns $tx^B_{rf} := (tx^{B,0}_{rf}, tx^{B,1}_{rf})$ and the associated $\pi^B_{rf} := (\widehat{\sigma}^{B,0}_{rf}(Z_B), z_B, \widehat{\sigma}^{A,0}_{rf}(Z_A), \widehat{\sigma}^{B,1}_{rf}(Y_A))$. Here, v_A is the initial assignable volume of A (and v_B is for B), pk_A is a public key of A (and pk_B is for B), $v_A + v_B = t, \forall I \in \{A, B\}, i \in \{0,1\}, \widehat{\sigma}^{I,i}_{rf}(Z)$ is a pre-signature on $tx^{I,i}_{rf}$. The pair $(Y; y) \in \mathsf{R}$ represents statements and witnesses of relation R, and z is the time-lock puzzle of y. Additionally, Z represents the corresponding public information (e.g., statement) of z.

Address Generation

To launch the payment channel, parties need to generate several key pairs and shared addresses, and this phase is only invoked once. First, parties invoke $\Pi_{\mathsf{AS}}.\mathsf{KG}(\lambda)$ to obtain the following key pairs.

- Party A obtains $(pk^A_{\mathsf{FCh}}, sk^A_{\mathsf{FCh}})$, $(pk^A_{\mathsf{SleepyCh}}, sk^A_{\mathsf{SleepyCh}})$, and $(pk^A_{\mathsf{Exit}}, sk^A_{\mathsf{Exit}})$.
- Party B obtains $(pk^B_{\mathsf{FCh}}, sk^B_{\mathsf{FCh}})$, $(pk^B_{\mathsf{SleepyCh}}, sk^B_{\mathsf{SleepyCh}})$, and $(pk^B_{\mathsf{Exit}}, sk^B_{\mathsf{Exit}})$.

Second, parties obtain the shared addresses, FCh_A, FCh_B, $\mathsf{SleepyCh}_A$, $\mathsf{SleepyCh}_B$, $\mathsf{Exit}A$, and Exit_B, by invoking $\Pi_{\mathsf{AS}}.\Gamma_{\mathsf{AKG}}(\lambda)$.

i-th Payment (Off-Chain)

Assume that A's and B's balance in the i-th payment are $v_{A,i}$ and $v_{B,i}$ respectively, where $v_{A,i} + v_{B,i} = t$, then A and B execute the following operations.

Payment Transactions. For parties to launch the i-th payment, assemble payment transactions $tx^A_{Pay,i} := tx(\mathsf{Ch}_{AB}, \mathsf{FCh}_A, v_{A,i} + c, 0)$, $tx^B_{Pay,i} := tx(\mathsf{Ch}_{AB}, \mathsf{FCh}_B, v_{B,i} + c, 0)$, $tx^A_{RPay,i} := tx(\mathsf{Ch}_{AB}, \mathsf{Exit}_B, v_{A,i} + c, 1)$, and $tx^B_{RPay,i} := tx(\mathsf{Ch}_{AB}, \mathsf{Exit}_A, v_{B,i} + c, 1)$. Obviously, the nonce is set as "0" in both $tx^A_{Pay,i}$ and $tx^B_{Pay,i}$, and as "1" in both $tx^A_{RPay,i}$ and $tx^B_{RPay,i}$.

Fork-Payment Transactions. For parties to separate the collateral from the balance spontaneously, assemble fork-payment transactions $tx^{A*}_{Pay,i} := (\mathsf{FCh}_A, \mathsf{SleepyCh}_A, v_{A,i}, 0)$, $tx^A_R := (\mathsf{FCh}_A, pk_A, c, 1)$, $tx^{B*}_{Pay,i} := (\mathsf{FCh}_B, \mathsf{SleepyCh}_B, v_{B,i}, 0)$, $tx^B_R := (\mathsf{FCh}_B, pk_B, c, 1)$.

Punishment Transactions. For parties to revoke the current i-th payment, assemble punishment transactions $tx^A_{Pnsh,i} := (\mathsf{SleepyCh}_B, pk_A, v_{B,i}, 0)$ and $tx^B_{Pnsh,i} := (\mathsf{SleepyCh}_A, pk_B, v_{A,i}, 0)$.

Finish-Payment Transactions. The following three types of transactions are assembled to achieve lazy-payment or fast finish-payment.

- Assemble lazy-finish transactions $tx^A_{D,i} := (\mathsf{SleepyCh}_A, pk_A, v_{A,i}, 0)$ and $tx^B_{D,i} := (\mathsf{SleepyCh}_B, pk_B, v_{B,i}, 0)$ both timelocked until time $\mathbf{T}_d$.
- Assemble fast-finish transactions $tx^A_{FPay,i} := (\mathsf{SleepyCh}_A, pk_A, v_{A,i}, 0)$ and $tx^B_{FPay,i} := (\mathsf{SleepyCh}_B, pk_B, v_{B,i}, 0)$.
- Assemble exiting transactions $tx^{A*}_{RPay,i} := (\mathsf{Exit}_B, pk_A, v_{A,i} + c, 0)$ and $tx^{B*}_{RPay,i} := (\mathsf{Exit}_A, pk_B, v_{B,i} + c, 0)$ for the above fast-finish payment.

Generating (Adaptor) Signatures. Parties jointly compute the (adaptor) signatures for the above transactions via invoking $\Pi_{\mathsf{AS}}.\Gamma_{\mathsf{pSign}}$ or $\Pi_{\mathsf{AS}}.\Gamma_{\mathsf{Sign}}$. The time-lock puzzles embedded in witnesses $y_{A,1,i}$ and $y_{A,2,i}$ (denoted as $z_{A,1,i}$ and $z_{A,2,i}$ respectively) are essential for generating the pre-signatures of $tx^A_{Pay,i}$ and $tx^B_{Pay,i}$. These time-lock puzzles ensure the proper sequential execution and prevent unauthorized early closure of the channel. If one party (e.g., A) quits at step i, the other (i.e., B) can close the channel by posting the $(i-1)$-th payment state.

- Party A obtains pre-signature $\widehat{\sigma}^A_{Pay,i}(Z_{A,1,i})$ and $z_{A,1,i}$ on transaction $tx^A_{Pay,i}$ where $Z_{A,1,i}$ is the corresponding public information (e.g., statement) of $z_{A,1,i}$, pre-signature $\widehat{\sigma}^B_{Pay,i}(Z_{A,2,i})$ on $tx^B_{Pay,i}$ (likewise for $Z_{A,2,i}$), and pre-signature $\widehat{\sigma}^A_{RPay,i}(Y_{A,2,i})$ on $tx^A_{RPay,i}$ under the shared address Ch_{AB}. Analogously, party B obtains $\widehat{\sigma}^B_{Pay,i}(Z_{A,2,i})$ and $z_{A,2,i}$ on $tx^B_{Pay,i}$, $\widehat{\sigma}^A_{Pay,i}(Z_{A,1,i})$ on $tx^A_{Pay,i}$, and $\widehat{\sigma}^B_{RPay,i}(Y_{A,1,i})$ on $tx^B_{RPay,i}$ under Ch_{AB}.
- Party A obtains signature $\sigma^{A*}_{Pay,i}$ on transaction $tx^{A*}_{Pay,i}$, and signature $\sigma^A_{R,i}$ on transaction $tx^A_{R,i}$ with regard to the shared address FCh_A. Party B obtains $\sigma^{B*}_{Pay,i}$ on $tx^{B*}_{Pay,i}$, and $\sigma^B_{R,i}$ on $tx^B_{R,i}$ under the shared address FCh_B.
- Party A obtains signature $\sigma^A_{D,i}$ on $tx^A_{D,i}$ under the shared $\mathsf{SleepyCh}_A$. Party B obtains signature $\sigma^B_{D,i}$ on $tx^B_{D,i}$ under the shared $\mathsf{SleepyCh}_B$.
- Party A obtains pre-signature $\widehat{\sigma}^A_{FPay,i}(Y_{B,1,i})$ on transaction $tx^A_{FPay,i}$ under $\mathsf{SleepyCh}_A$, and pre-signature $\widehat{\sigma}^{B*}_{RPay,i}(Y_{B,1,i})$ on transaction $tx^{B*}_{RPay,i}$ under Exit_A. Party B obtains pre-signature $\widehat{\sigma}^{B*}_{RPay,i}(Y_{B,1,i})$ and witness $y_{B,1,i}$ on transaction $tx^{B*}_{RPay,i}$ under Exit_A. Analogously, party B obtains pre-signature $\widehat{\sigma}^B_{FPay,i}(Y_{B,2,i})$ on transaction $tx^B_{FPay,i}$ under the shared address $\mathsf{SleepyCh}_B$, and pre-signature $\widehat{\sigma}^{A*}_{RPay,i}(Y_{B,2,i})$ on transaction $tx^{A*}_{RPay,i}$ under the shared address Exit_B. Party A obtains pre-signature $\widehat{\sigma}^{A*}_{RPay,i}(Y_{B,2,i})$ and witness $y_{B,2,i}$ on transaction $tx^{B*}_{RPay,i}$ under Exit_B.

i-th Payment Revocation (Off-Chain)

Once parties negotiate to revoke the i-th payment, they invoke $\Pi_{\mathsf{AS}}.\Gamma_{\mathsf{Sign}}$ to obtain respective signatures. In case one party aborts during the revocation, the other non-aborting party can close the channel via publishing the recent unrevoked payment on the chain.

- Party A obtains signature $\sigma^A_{Pnsh,i}$ on transaction $tx^A_{Pnsh,i}$ with regard to the shared address $\mathsf{SleepyCh}_B$.
- Party B obtains signature $\sigma^B_{Pnsh,i}$ on transaction $tx^B_{Pnsh,i}$ with regard to the shared address $\mathsf{SleepyCh}_A$.

Figure 5: AMBiPay protocol - Initialization and payments.

Channel Closing (On-Chain)

Both parties can unilaterally close the channel Ch_{AB} via publishing the j-th unrevoked payment.

1. Party A first invokes $\Pi_{\mathsf{AS}}.\mathsf{Adapt}$ with the inputs of pre-signature $\widehat{\sigma}^A_{Pay,j}(Z_{A,1,j})$ and puzzle $z_{A,1,j}$, and obtains the signature $\sigma^A_{Pay,j}$. Then, she publishes $(tx^A_{Pay,j}, \sigma^A_{Pay,j})$ on $\mathbb{L}$. Afterward, party A can independently receive her collateral c by posting tx^A_R along with its associated signature. Then, one of the following events will occur.
   - *Fast finish*. Party B first invokes $\Pi_{\mathsf{AS}}.\mathsf{Ext}$ with pre-signature $\widehat{\sigma}^A_{Pay,j}(Z_{A,1,j})$ and signature $\sigma^A_{Pay,j}$, to obtain the puzzle $z_{A,1,j}$. By force-opening $z_{A,1,j}$, B obtains the witness $y_{A,1,j}$ and utilizes $\widehat{\sigma}^B_{RPay,j}(Y_{A,1,j})$ and $y_{A,1,j}$ to recover signature $\sigma^B_{RPay,j}$, as well as $\widehat{\sigma}^{B*}_{RPay,j}(Y_{B,1,j})$ and $y_{B,1,j}$ to recover signature $\sigma^{B*}_{RPay,j}$ via invoking $\Pi_{\mathsf{AS}}.\mathsf{Adapt}$. After party B publishes $(tx^{B*}_{RPay,j}, \sigma^{B*}_{RPay,j})$ on $\mathbb{L}$, party A can recover signature $\sigma^A_{FPay,j}$ via the sequential invocation of $\Pi_{\mathsf{AS}}.\mathsf{Ext}$ and $\Pi_{\mathsf{AS}}.\mathsf{Adapt}$. Finally, party A publishes $(tx^A_{FPay,j}, \sigma^A_{FPay,j})$ on $\mathbb{L}$ and finish the payment fast.
   - *Lazy finish*. If party B does not respond timely, party A can publish $(tx^A_{D,j}, \sigma^A_{D,j})$ on $\mathbb{L}$ after timeout $\mathbf{T}_d$.
2. Analogously, party B invokes $\Pi_{\mathsf{AS}}.\mathsf{Adapt}$ with pre-signature $\widehat{\sigma}^B_{Pay,j}(Z_{A,2,j})$ and puzzle $z_{A,2,j}$ to obtain the signature $\sigma^B_{Pay,j}$. He then proceeds to publish $(tx^B_{Pay,j}, \sigma^B_{Pay,j})$ on $\mathbb{L}$. Afterward, party B can independently receive his collateral c by posting tx^B_R along with its associated signature. Then, one of the following events will occur.
   - *Fast finish*. Party A first invokes $\Pi_{\mathsf{AS}}.\mathsf{Ext}$ with pre-signature $\widehat{\sigma}^B_{Pay,j}(Z_{A,2,j})$ and signature $\sigma^B_{Pay,j}$, to obtain the puzzle $z_{A,2,j}$. By force-opening $z_{A,2,j}$, A obtains the witness $y_{A,2,j}$ and utilizes $\widehat{\sigma}^A_{RPay,j}(Y_{A,2,j})$ and $y_{A,2,j}$ to recover signature $\sigma^A_{RPay,j}$, as well as $\widehat{\sigma}^{A*}_{RPay,j}(Y_{B,2,j})$ and $y_{B,2,j}$ to recover signature $\sigma^{A*}_{RPay,j}$ via invoking $\Pi_{\mathsf{AS}}.\mathsf{Adapt}$. After part A publishes $(tx^{A*}_{RPay,j}, \sigma^{A*}_{RPay,j})$ on $\mathbb{L}$, party B can recover signature $\sigma^B_{FPay,j}$ via the sequential invocation of $\Pi_{\mathsf{AS}}.\mathsf{Ext}$ and $\Pi_{\mathsf{AS}}.\mathsf{Adapt}$. Finally, party A publishes $(tx^B_{FPay,j}, \sigma^B_{FPay,j})$ on $\mathbb{L}$ for fast finish.
   - *Lazy finish*. If party A does not respond timely, party B can publish $(tx^B_{D,j}, \sigma^B_{D,j})$ on $\mathbb{L}$ after timeout $\mathbf{T}_d$.

Revoked Payments Punishment (On-Chain)

Assume that party A publishes the j-th revoked payment $(tx^A_{Pay,j}, \sigma^A_{Pay,j})$ on $\mathbb{L}$, party B can publish the punishment transaction $(tx^B_{Pnsh,j}, \sigma^B_{Pnsh,j})$ on $\mathbb{L}$ before the stated timeout $\mathbf{T}_d$. Analogously, assume that party B publishes the j-th revoked payment $(tx^B_{Pay,j}, \sigma^B_{Pay,j})$ on $\mathbb{L}$, party A can publish the punishment transaction $(tx^A_{Pnsh,j}, \sigma^A_{Pnsh,j})$ on $\mathbb{L}$ before the stated timeout $\mathbf{T}_d$. In case of misbehavior, the guilty party can only redeem the collateral c, while the innocent party will receive the total t coins from Ch_{AB} along with their collateral c.

Figure 6: AMBiPay protocol - Channel closing and punishment.

less bi-party computation enhances security by eliminating complex smart contracts, reducing vulnerabilities.

We next present our central hypothesis and provide a brief summary of our analysis. Appendix C contains a formal specification of AMBiPay Π_{B} in the UC framework, which differs from the one described in Section 5, denoted as Π'''_{B}. Specifically, we substitute the protocols of 2-party aggregated key generation, 2-party signing, and 2-party pre-signing with their corresponding ideal functionalities.The former two have been defined in [8], and the ideal functionality of 2-party pre-signing is straightforward, relying on the 2-party signing protocol and NIZK proof, both of which have been defined. We prove this substitution in the following lemma.

Lemma 1 *The protocols Π_{B} and Π'''_{B} are computationally indistinguishable to the environment $\mathcal{E}$, given that $\Pi_{\mathsf{AS}}.\Gamma_{\mathsf{AKG}}$, $\Pi_{\mathsf{AS}}.\Gamma_{\mathsf{Sign}}$, and $\Pi_{\mathsf{AS}}.\Gamma_{\mathsf{pSign}}$ are UC-secure protocols of 2-party aggregated key generation, 2-party signing, and 2-party pre-signing, respectively.*

In Appendix C.1, we simulate $\mathcal{S}$ to interact with the ideal functionality $\mathcal{F}_{AM}$ presented in Appendix B, while the environment $\mathcal{E}$ to interact with $\phi_{\mathcal{F}_{AM}}$ (the ideal protocol for $\mathcal{F}_{AM}$). We then in Appendix C.2, prove that any attack against Π_{B} can also be performed against $\phi_{\mathcal{F}_{AM}}$. Thereby we conclude the theorem as follows.

Theorem 2 *The protocol Π_{B} is the UC-realization of the ideal functionality $\mathcal{F}_{AM}$.*

5.3. Extension

We further present our OVTS for efficiently generating delayed transactions. In the following, we provide a brief overview of OVTS. Assume that a committer C owns the signer S's signature σ. The committer C first creates a one-time secret / public key pair (sk_o, pk_o). Then he utilizes the pk_o to encrypt the signature and obtains the ciphertext c. Next, he calculates a time-lock puzzle z with timelock $\mathbf{T}$ for the one-time secret key sk_o. This means that anyone can recover sk_o from solving the puzzle after the timelock $\mathbf{T}$, and further employs sk_o to obtain the signature σ from the ciphertext c. Finally, the committer generates a NIZK proof for the verifiability before time $\mathbf{T}$, which is to prove the knowledge of the witnesses (σ, sk_o) satisfying that: 1) σ is valid upon the signer's public key pk_s; 2) c is a correct ciphertext of σ under the one-time public key pk_o; 3) z is the correct puzzle of sk_o with the timelock $\mathbf{T}$.

To formally present our OVTS, we denote Digital Signatures as DS = (DS.Setup, DS.KG, DS.Sign, DS.Verf), Public-Key Encryption as PKE = (PKE.Setup,

$\mathsf{PKE.KG}$, $\mathsf{PKE.Enc}$, $\mathsf{PKE.Dec}$), Time-Lock Puzzles as TLP = ($\mathsf{TLP.Setup}$, $\mathsf{TLP.PGen}$, $\mathsf{TLP.PSolve}$), and NIZK as NIZK = ($\mathsf{NIZK.Setup}$, $\mathsf{NIZK.Prove}$, $\mathsf{NIZK.Verf}$). Then, we present our OVTS as follows.

- $\mathsf{OVTS.Setup}$. This setup algorithm takes as input a security parameter λ, and invokes $\mathsf{DS.Setup}$, $\mathsf{PKE.Setup}$, $\mathsf{TLP.Setup}$, and $\mathsf{NIZK.Setup}$ to obtain pp_{ds}, pp_{pke}, pp_{tlp}, and crs_{zk}. It outputs the public parameter $pp_{ovts} := (pp_{ds}, pp_{pke}, pp_{tlp}, crs_{zk})$.
- $\mathsf{OVTS.Commit\text{-}and\text{-}Prove}$. This commit-and-prove algorithm takes as input the public parameter pp_{ovts}, and a message / signature pair (m, σ) under the signer's public key pk_s. It first parses $pp_{ovts} := (pp_{ds}, pp_{pke}, pp_{tlp}, crs_{zk})$ and generates a one-time secret/public key pair $(sk_o, pk_o) \leftarrow \mathsf{PKE.KG}(pp_{pke})$. Then it encrypts the signature via $c \leftarrow \mathsf{PKE.Enc}_{pk_o}(pp_{pke}, \sigma)$ and generates the puzzle via $z \leftarrow \mathsf{TLP.PGen}(pp_{tlp}, sk_o, r)$ where r is the randomness adopted in the TLP. Next, it computes the NIZK proof of language $\mathcal{L}_{ovts}$ via invoking $\pi = \mathsf{NIZK.Prove}(crs_{zk},\ x_{ovts}, w_{ovts})$, where $\mathcal{L}_{ovts}$ is denoted as

$$\mathcal{L}_{ovts} = \mathsf{PoK}\left\{\begin{array}{l}(x_{ovts}, w_{ovts}):\\ \mathsf{DS.Verf}_{pk_s}(pp_{ds}, m, \sigma) = 1 \wedge\\ c \leftarrow \mathsf{PKE.Enc}_{pk_o}(pp_{pke}, \sigma)\\ z = \mathsf{TLP.PGen}(pp_{tlp}, sk_o, r) \wedge\\ (sk_o, pk_o) \leftarrow \mathsf{PKE.KG}(pp_{pke})\end{array}\right\}(m),$$

  where $x_{ovts} := (pp_{ovts}, c, z, pk_s)$, $w_{ovts} := (\sigma, sk_o, r)$. Finally, this algorithm outputs the commitment $c_{ovts} := (x_{ovts}, \pi)$.
- $\mathsf{OVTS.Verf}$. This verification algorithm takes as input public parameter pp_{ovts} and commitment c_{ovts}, and it parses $c_{ovts} := (x_{ovts}, \pi)$ as well as invokes $b := \mathsf{NIZK.Verf}(x_{ovts}, \pi)$. It outputs b, where $b = 1$ means c_{ovts} is valid, and $b = 0$ invalid.
- $\mathsf{OVTS.Open}$. This open algorithm (run by committer) takes as input public parameter pp_{ovts} and commitment c_{ovts}, and it outputs the committed signature σ and randomness r adopted in generating c_{ovts}.
- $\mathsf{OVTS.ForceOpen}$. This force open algorithm takes as input public parameter pp_{ovts} and commitment c_{ovts}, and it invokes $sk_o := \mathsf{TLP.PSolve}(pp_{tlp}, z)$ and outputs $\sigma := \mathsf{PKE.Dec}_{sk_o}(pp_{pke}, c)$.

Security. The following theorems demonstrate the privacy and soundness properties of our OVTS as mentioned above. Formal proofs are provided in Appendix D.1.

Theorem 3 *Let* NIZK *be a NIZK proof system for* $\mathcal{L}_{ovts}$, PKE *be an* $\mathsf{IND\text{-}CPA}$ *encryption scheme, and* TLP *be a secure time-lock puzzle. Then, the OVTS described above satisfies the privacy property.*

Theorem 4 *Let* NIZK *be a NIZK proof system for* $\mathcal{L}_{ovts}$, PKE *be an IND-CPA encryption scheme, and* TLP *be a time-lock puzzle with perfect completeness. Then, the OVTS described above satisfies the soundness property.*

Efficiency, Compatibility, and Instantiation. OVTS operates on the entire signature with a one-time public key, but VTS involves n signature shares and n time-lock puzzles. The time-lock puzzle in OVTS is applied solely to the one-time public key, reducing the computation cost from $O(n)$ to $O(1)$. In particular, the committing-and-proving algorithm in OVTS mainly involves the computation cost of key generation in public key encryption, puzzle generation in TLP, and $\mathsf{NIZK.Prove}$, which are all with constant efficiency (i.e., independence with n). With regard to compatibility, the proposed OVTS is a generic construction without the limitation of specified digital signature, public key encryption, TLP, and NIZK. However, the committer's work may still be substantial if adopting a zk-SNARK as NIZK for a fairly complex statement [36]. Therefore, we suggest to instantiate NIZK as Σ-protocol [37] or Bulletproofs [38] if the digital signature adopted in AMBiPay is with the algebraic properties (e.g., ECDSA and Schnorr).

To facilitate an easy understanding of the above construction, we instantiate OVTS with ECDSA, Pallier encryption [39], and Homomorphic Time-Lock Puzzles (HTLP [40]). We apply the Σ-protocol and Fiat-Shamir transformation [41] to realize the NIZK proof. Detailed information could be referred to Appendix D.2.

6. Performance Analysis

We first implemented building blocks of AMBiPay including 2-party adaptor signature schemes, OVTS scheme and transaction flow, followed by conducting the efficiency evaluation via benchmarks.

6.1. Implementation

To evaluate the three Π_{AS} schemes, we instantiated the underlying 2-party signature scheme as BLS [30], ECDSA [32], and Schnorr [31] (denoted as BLS-Π_{AS}, ECDSA-Π_{AS}, Schnorr-Π_{AS} hereafter). In addition, to compare our OVTS with VTS, we focused on the instantiation of ECDSA signature schemes and utilized the implementation of VTS and HTLP available at [15] and [40], respectively. We will refer to them as OVTS-ECDSA and VTS-ECDSA in the subsequent discussion. Notice that we did not include the setup algorithm in our presentation as it can be precomputed and shared across various instances of AMBiPay. Similarly, we omitted the $\mathsf{ForceOpen}$ algorithm as its running time is predetermined by the time hardness $\mathbf{T}$.

The above schemes were implemented in C++ on a personal computer (PC), and their source code is available at https://github.com/AMBiPay/AMBiPay. Note that we did not perform any optimizations (logical and others) or concurrency. This suggests that our current implementation is a proof-of-concept and has room for substantial improvement when it comes to production-level performance. Concretely, our PC is configured with the Windows 10 operating system (64-bit) and equipped with an Intel(R) Core(TM) i7-9750H CPU with a clock speed of 2.60 GHz and 16 GB of RAM. The employed cryptographic library is Miracl V7.0 with

TABLE 2: Time cost of three Π_{AS} schemes (in ms).

Algorithm	BLS	Schnorr	ECDSA
Γ_{Sign}-Offline	0	42.1949	206.3300
Γ_{Sign}-Online	105.4810	0.4992	69.2851
Verf	342.3900	21.3054	21.0161
Γ_{pSign}-Offline	396.4870	21.2640	31.4981
Γ_{pSign}-Online	105.6701	0.5043	69.3147
pVerf	621.5080	41.6403	43.5941
Adapt	0.2055	0.0063	0.0138
Ext	0.2036	0.0066	0.0364

the chosen standard NIST curve secp256k1 and BLS curve (ate pairing embedding degree 24), both of which are with 256 bits security level. As the Paillier encryption system involved in [32] recommends the module $N \geq q^3 + q^2$, we set N as 1024 bits because the q is with 256 bits in our implementations. Thus, the size of an element in $\mathbb{Z}_q^*$, $\mathbb{Z}_N^*$, $\mathbb{G}$, $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ is 32 bytes, 128 bytes, 64 bytes, 160 bytes, 640 bytes and 1920 bytes, respectively.

For testing the feasibility of AMBiPay, we implemented it on Sepolia, an Ethereum testnet that closely resembles the Ethereum network. Sepolia offers free requests of funds and a user-friendly web interface for block explorers, based on which Web3 developers can test their projects. To conduct our simulation, we employed the MetaMask-Chrome 9 plug-in of Google Chrome to connect to Sepolia and deploy transactions while measuring overhead.

6.2. Benchmarks

2-party Adaptor Signatures. Table 2 shows the time cost of each algorithm in three Π_{AS} schemes. BLS-Π_{AS} involves more time costs than the other two, as it requires time-consuming bilinear pairing and hash-to-point operations. Fortunately, the online computation of BLS-Π_{AS} in both Γ_{Sign} and Γ_{pSign} is less than 106 ms. This is still acceptable when compared to the consensus latency of blockchains, such as 10 minutes in Bitcoin and 15 seconds in Ethereum. Except that, each algorithm in Schnorr-Π_{AS} requires not more than 50 ms, and that in ECDSA-Π_{AS} (except Γ_{Sign}-Offline) requires less than 70 ms. The adapting and extracting operations in three schemes are very fast, taking less than 1 millisecond. These results are consistent with existing 2-party adaptor signature schemes [10], [25], [26]. Notably, our proposal supports unique signatures like BLS, which is not possible with existing schemes.

We further compare the computation and communication costs of BLS-Π_{AS}, ECDSA-Π_{AS} and Schnorr-Π_{AS} in terms of Γ_{Sign} and Γ_{pSign} algorithms. These two algorithms involve the most costs (especially communication) when applied in AMBiPay. From the left sub-figure of Figure 7, the distance (i.e., online cost) of both Sign and pSign in Schnorr-Π_{AS} is shorter than that of ECDSA-Π_{AS} and that of BLS-Π_{AS}. Moreover, the time and size of Schnorr-Π_{AS} are the least among these comparing schemes, which are less than 100 ms and 0.5 KB, respectively. Although this intuitively indicates that Schnorr-Π_{AS} owns the efficiency advantage, the distance of both Sign and pSign in ECDSA-Π_{AS} is also feasible in AMBiPay. This will be demonstrated in the subsequent overhead analysis of AMBiPay. With respect to BLS-Π_{AS}, further optimizations such as choosing a more well-matched pairing-friendly curve or logical improvement in the code level, may be performed to support AMBiPay.

Optimized Verifiable Timed Signatures. We evaluated OVTS-ECDSA and VTS-ECDSA with an increasing n (i.e., the total number in secret sharing), and the threshold was set as $t = n/2$. We obtain the comparison results shown in the center and right sub-figures of Figure 7, from which OVTS-ECDSA is more efficient than VTS-ECDSA from both time and size. This is because OVTS-ECDSA does not need to generate n shares of signatures, public keys and puzzles, and consequently its efficiency is independent of n. Obviously, this independence also holds for OVTS instantiated from other digital signatures such as BLS and Schnorr. Thus, OVTS demonstrates a significant practical advantage when compared to VTS.

Deployment of Transactions. We now present the transactions in AMBiPay, and the details on transaction latency and sizes will be given later. To demonstrate the backward compatibility of AMBiPay with existing account-based currencies, we also provide a pointer to the corresponding transactions posted in Sepolia.

We assume two parties A and B to execute AMBiPay. The first step in AMBiPay is to create the funding transaction $tx_F^{A,B}$, which involves two transactions tx_A [42] and tx_B [43] to lock balance and collateral in Ch_{AB}. Next, we examine A's and B's state transactions $tx^A_{Pay,i}$ [44] and $tx^B_{RPay,i}$ [45] (when A is initiative), and these transactions are symmetric if for B to obtain initiative. There are two ways for A to claim her coins, when she posts $tx^A_{Pay,i}$ and $tx^{A*}_{Pay,i}$ [46] on Sepolia. If B refunds his balance and collateral via posting $tx^B_{RPay,i}$ and $tx^{B*}_{RPay,i}$ [47], then A can claim her funds via posting $tx^A_{FPay,i}$ [48] right away. Otherwise, if the timelock expires, A can refund her coins with $tx^A_{D,i}$. Once A posts an old state, B can post $tx^B_{Pnsh,i}$ to punish A. Finally, both parties can close the channel with their transactions, and the funds will be assigned correctly.

Comparison to Sleepy Channels. The difference between account model and UTXO model, together with the strong constraint via 2-party adaptor signatures, causes more but acceptable costs in AMBiPay (see Table 3). AMBiPay involves more transactions than Sleepy Channels in almost all phases, but the difference is subtle when putting them into the concrete Bitcoin and Ethereum testnets. For creation, Sleepy Channels involve 1.9785 KB off-chain and 0.3301 KB on-chain, and AMBiPay involves 1.9687 KB off-chain and 0.2353 KB on-chain. For updating, Sleepy Channels require 2.3515 KB off-chain, but AMBiPay only requires 2.2031 KB off-chain. The most transactions of AMBiPay and Sleepy Channels are identical and associated with signatures, except that transactions $tx_{Pay,i}$, $tx_{RPay,i}$, $tx^*_{RPay,i}$, $tx_{Fpay,i}$ in AMBiPay refer to pre-signatures.

With regard to computation, we exclude the online part because block generation time varies widely across different

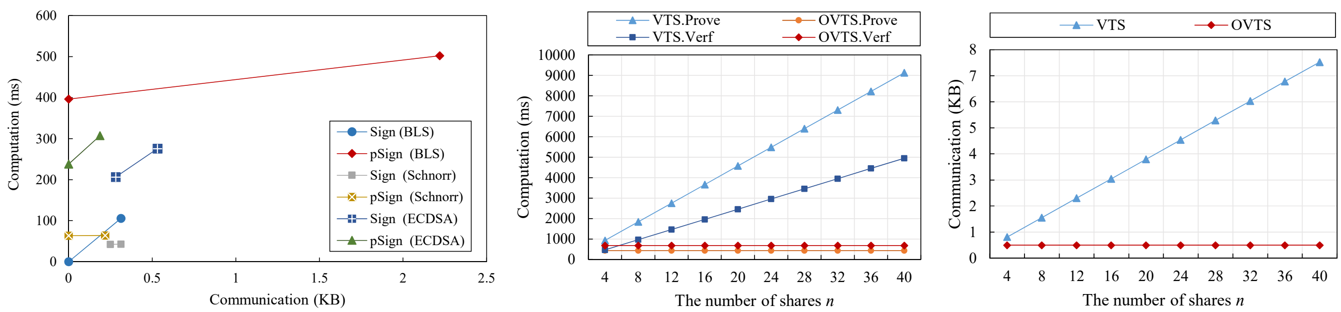

Figure 7: *Left sub-figure*: showing cost comparison of three instantiated 2-party adaptor signature schemes. We represent the offline cost as the south-west point, the total cost as the north-east point, and the online cost as the distance between the two points. *Center sub-figure and right sub-figure*: showing cost comparison of computation and communication between VTS and OVTS, respectively.

TABLE 3: Overhead comparison between Sleepy Channels and AMBiPay.

	txs off-chain				txs on-chain			
	Sleepy Channels	KB	AMBiPay	KB	Sleepy Channels	KB	AMBiPay	KB
Create	$2 \cdot (tx^A_{Pay,i} + tx^{A,B}_{FPay,i} + tx^{A*}_{FPay,i} + tx^{A,A}_{Fpay,i})$	1.9785	$2 \cdot (tx^A_{Pay,i} + tx^{A*}_{Pay,i} + tx^A_{R,i} + tx^B_{RPay,i} + tx^{B*}_{RPay,i} + tx^A_{Fpay,i})$	1.9687	tx_F	0.3301	$tx_A + tx_B$	0.2353
Update	$2 \cdot (tx^A_{Pay,i} + tx^{A,B}_{FPay,i} + tx^{A*}_{FPay,i} + tx^{A,A}_{Fpay,i} + tx^A_{Pnsh,i})$	2.3515	$2 \cdot (tx^A_{Pay,i} + tx^{A*}_{Pay,i} + tx^A_{R,i} + tx^A_{Fpay,i} + tx^B_{RPay,i} + tx^{B*}_{RPay,i} + tx^B_{Pnsh,i})$	2.2031	-	-	-	-
Close (optimistic)	-	-	-	-	$tx^A_{Pay,i}$	0.2197	$tx^A_{Pay,i} + tx^B_{RPay,i}$	0.3046
Close (slow)	-	-	-	-	$tx_{Pay,i} + tx^{A,A}_{FPay,i}$	0.4384	$tx^A_{Pay,i} + tx^{A*}_{Pay,i} + tx^A_{D,i}$	0.4218
Close (fast)	-	-	-	-	$tx^A_{Pay,i} + tx^{A,B}_{Fpay,i} + tx^{A*}_{Fpay,i}$	0.8037	$tx^A_{Pay,i} + tx^{A*}_{Pay,i} + tx^B_{RPay,i} + tx^{B*}_{RPay,i} + tx^A_{FPay,i}$	0.8671
Punish	-	-	-	-	$tx^A_{Pay,i} + tx^A_{Pnsh,i}$	0.4394	$tx^A_{Pay,i} + tx^{A*}_{Pay,i} + tx^B_{Pnsh,i}$	0.4218

currencies, ranging from 10 minutes in Bitcoin to 15 seconds in Ethereum on average. The off-line computation in both Sleepy Channels and AMBiPay is caused by creation and updating. Sleepy Channels involve about 346.4255 ms for both creation and updating, while 554.3992 ms and 415.7994 ms were for that of AMBiPay, respectively. To launch a new off-line payment, parties need to spend about 346.4255 ms in Sleepy Channels but 415.7994 ms in AMBiPay. This compromise is acceptable for reaping reliability when applying Sleepy Channels into account-based currencies. Note that we apply OVTS in both Sleepy Channels and AMBiPay for fairness. Otherwise, AMBiPay will significantly outperform Sleepy Channels, which currently adopt the less efficient VTS.

Overhead of AMBiPay. We finally conclude the overhead of AMBiPay from communication and off-chain computation. In the creation phase, the two parties together need to exchange about 2.2041 KB (including 12 off-chain txs and 2 on-chain txs). Each updating phase involves 2.2031 KB communication costs (i.e., 14 off-chain txs). During the closing phase, which happens on-chain, there are three possible situations: optimistic, slow, and fast. The optimistic situation is when they close the channel honestly (0.3046 KB, 2 txs), the slow is when one party unilaterally closes and waits for the timelock to expire before unlocking funds (0.4218 KB, 3 txs), and the fast is when one party unilaterally closes and the other party refunds immediately (0.8671 KB, 5 txs). The punishment case involves 0.4218 KB (i.e., 3 txs). Only the creation and updating phases are performed off-chain, and their respective off-chain computation costs are 554.3992 ms and 415.7994 ms. This overhead indeed demonstrates that AMBiPay is comparable to Sleepy Channels.

7. Conclusion

We introduce a new trustless and scriptless bidirectional payment channels (bi-PC) protocol for the account model, which addresses the uncertainty and weak constraint challenges faced by existing solutions. Our proposal inherits the advantages of existing bi-PC protocols for UTXO and eliminates the need for complex scripts, such as smart contracts, in bi-PC for the account model. Additionally, we introduce the first generic construction of 2-party adaptor signatures for both unique and common signatures, as well as an optimized verifiable timed signatures construction to improve efficiency. Our evaluation shows that our proposal is efficient and applicable to account-based currencies, solving the open problem raised by Aumayr et al. in CCS'22 regarding the applicability of Sleepy Channels to such systems.

References

[1] E. Docs, "Decentralized storage," 2023. [Online]. Available: https://tinyurl.com/fbduef58

[2] B. wiki, “Payment channels,” 2021. [Online]. Available: https://tinyurl.com/y8kr9cja

[3] J. Lind, I. Eyal, P. R. Pietzuch, and E. G. Sirer, “Teechan: Payment channels using trusted execution environments,” *CoRR*, vol. abs/1612.07766, 2016. [Online]. Available: http://arxiv.org/abs/1612.07766

[4] G. Avarikioti, E. Kokoris-Kogias, and R. Wattenhofer, “Brick: Asynchronous state channels,” *CoRR*, vol. abs/1905.11360, 2019. [Online]. Available: http://arxiv.org/abs/1905.11360

[5] M. M. T. Chakravarty, S. Coretti, M. Fitzi, P. Gazi, P. Kant, A. Kiayias, and A. Russell, “Hydra: Fast isomorphic state channels,” *IACR Cryptol. ePrint Arch.*, p. 299, 2020. [Online]. Available: https://eprint.iacr.org/2020/299

[6] J. Poon and T. Dryja, “The bitcoin lightning network,” 2016. [Online]. Available: https://tinyurl.com/avpj5vnv

[7] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostáková, M. Maffei, P. Moreno-Sanchez, and S. Riahi, “Generalized channels from limited blockchain scripts and adaptor signatures,” in *ASIACRYPT 2021*, ser. Lecture Notes in Computer Science, M. Tibouchi and H. Wang, Eds., vol. 13091. Springer, 2021, pp. 635–664. [Online]. Available: https://doi.org/10.1007/978-3-030-92075-3_22

[8] L. Aumayr, S. A. Thyagarajan, G. Malavolta, P. Moreno-Sanchez, and M. Maffei, “Sleepy channels: Bi-directional payment channels without watchtowers,” in *ACM CCS 2022*, ser. CCS ’22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 179–192. [Online]. Available: https://doi.org/10.1145/3548606.3559370

[9] J. Clifford, “Intro to blockchain: Utxo vs account based,” 2019. [Online]. Available: https://tinyurl.com/mwy526u3

[10] A. Erwig, S. Faust, K. Hostáková, M. Maitra, and S. Riahi, “Two-party adaptor signatures from identification schemes,” in *PKC 2021*, ser. Lecture Notes in Computer Science, J. A. Garay, Ed., vol. 12710. Springer, 2021, pp. 451–480. [Online]. Available: https://doi.org/10.1007/978-3-030-75245-3_17

[11] C. Decker and R. Wattenhofer, “A fast and scalable payment network with bitcoin duplex micropayment channels,” in *SSS 2015*, ser. Lecture Notes in Computer Science, A. Pelc and A. A. Schwarzmann, Eds., vol. 9212. Springer, 2015, pp. 3–18. [Online]. Available: https://tinyurl.com/2p2srpys

[12] C. Decker and R. Russell, “eltoo: A simple layer2 protocol for bitcoin,” 2018. [Online]. Available: https://tinyurl.com/4xsfw8zu

[13] X. Qin, S. Pan, A. Mirzaei, Z. Sui, O. Ersoy, A. Sakzad, M. Esgin, J. K. Liu, J. Yu, and T. Yuen, “Blindhub: Bitcoin-compatible privacy-preserving payment channel hubs supporting variable amounts,” in *SP 2023*. Los Alamitos, CA, USA: IEEE Computer Society, may 2023, pp. 2462–2480. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.00116

[14] S. Dziembowski, S. Faust, and K. Hostáková, “General state channel networks,” in *ACM CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 949–966. [Online]. Available: https://doi.org/10.1145/3243734.3243856

[15] S. A. K. Thyagarajan, A. Bhat, G. Malavolta, N. Döttling, A. Kate, and D. Schröder, “Verifiable timed signatures made practical,” in *ACM CCS 2020*, ser. CCS ’20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1733–1750. [Online]. Available: https://doi.org/10.1145/3372297.3417263

[16] S. A. K. Thyagarajan, G. Malavolta, F. Schmid, and D. Schröder, “Verifiable timed linkable ring signatures for scalable payments for monero,” in *ESORICS 2022*, ser. Lecture Notes in Computer Science, V. Atluri, R. D. Pietro, C. D. Jensen, and W. Meng, Eds., vol. 13555. Springer, 2022, pp. 467–486. [Online]. Available: https://doi.org/10.1007/978-3-031-17146-8_23

[17] M. Khabbazian, T. Nadahalli, and R. Wattenhofer, “Outpost: A responsive lightweight watchtower,” in *AFT 2019*. ACM, 2019, pp. 31–40. [Online]. Available: https://doi.org/10.1145/3318041.3355464

[18] Z. Avarikioti, O. S. T. Litos, and R. Wattenhofer, “Cerberus channels: Incentivizing watchtowers for bitcoin,” in *FC 2020*, ser. Lecture Notes in Computer Science, J. Bonneau and N. Heninger, Eds., vol. 12059. Springer, 2020, pp. 346–366. [Online]. Available: https://doi.org/10.1007/978-3-030-51280-4_19

[19] A. Mirzaei, A. Sakzad, J. Yu, and R. Steinfeld, “FPPW: A fair and privacy preserving watchtower for bitcoin,” in *FC 2021*, ser. Lecture Notes in Computer Science, N. Borisov and C. Díaz, Eds., vol. 12675. Springer, 2021, pp. 151–169. [Online]. Available: https://doi.org/10.1007/978-3-662-64331-0_8

[20] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, “Multi-party virtual state channels,” in *EUROCRYPT 2019*, ser. Lecture Notes in Computer Science, Y. Ishai and V. Rijmen, Eds., vol. 11476. Springer, 2019, pp. 625–656. [Online]. Available: https://doi.org/10.1007/978-3-030-17653-2_21

[21] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, “Sprites and state channels: Payment networks that go faster than lightning,” in *FC 2019*, ser. Lecture Notes in Computer Science, I. Goldberg and T. Moore, Eds., vol. 11598. Springer, 2019, pp. 508–526. [Online]. Available: https://doi.org/10.1007/978-3-030-32101-7_30

[22] J. Jiao, S. Kan, S. Lin, D. Sanán, Y. Liu, and J. Sun, “Semantic understanding of smart contracts: Executable operational semantics of solidity,” in *SP 2020*. IEEE, 2020, pp. 1695–1712. [Online]. Available: https://doi.org/10.1109/SP40000.2020.00066

[23] A. Prut, “Ripple (xrp) ? quick introduction,” 2019. [Online]. Available: https://tinyurl.com/f9h7e3f2

[24] ——, “Stellar (xlm) ? quick introduction,” 2020. [Online]. Available: https://tinyurl.com/4v6yn59x

[25] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, “Anonymous multi-hop locks for blockchain scalability and interoperability,” in *NDSS 2019*. The Internet Society, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/anonymous-multi-hop-locks-for-blockchain-scalability-and-interoperability/

[26] S. A. K. Thyagarajan, G. Malavolta, and P. Moreno-Sanchez, “Universal atomic swaps: Secure exchange of coins across all blockchains,” in *SP 2022*. IEEE, 2022, pp. 1299–1316. [Online]. Available: https://doi.org/10.1109/SP46214.2022.9833731

[27] L. Fournier, “One-time verifiably encrypted signatures a.k.a. adaptor signatures,” 2019. [Online]. Available: https://tinyurl.com/2djh2adu

[28] S. Bursuc and S. Mauw, “Contingent payments from two-party signing and verification for abelian groups,” in *CSF 2022*. IEEE, 2022, pp. 195–210. [Online]. Available: https://doi.org/10.1109/CSF54842.2022.9919674

[29] B. Edgington, “Upgrading ethereum,” 2023. [Online]. Available: https://tinyurl.com/2p969w9b

[30] D. Boneh, M. Drijvers, and G. Neven, “Compact multi-signatures for smaller blockchains,” in *ASIACRYPT 2018*, ser. Lecture Notes in Computer Science, T. Peyrin and S. D. Galbraith, Eds., vol. 11273. Springer, 2018, pp. 435–464. [Online]. Available: https://doi.org/10.1007/978-3-030-03329-3_15

[31] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, “Secure distributed key generation for discrete-log based cryptosystems,” *J. Cryptol.*, vol. 20, no. 1, pp. 51–83, 2007. [Online]. Available: https://doi.org/10.1007/s00145-006-0347-3

[32] Y. Lindell, “Fast secure two-party ECDSA signing,” in *CRYPTO 2017*, ser. Lecture Notes in Computer Science, J. Katz and H. Shacham, Eds., vol. 10402. Springer, 2017, pp. 613–644. [Online]. Available: https://doi.org/10.1007/978-3-319-63715-0_21

[33] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, “Aggregate and verifiably encrypted signatures from bilinear maps,” in *EUROCRYPT 2003*, ser. Lecture Notes in Computer Science, E. Biham, Ed., vol. 2656. Springer, 2003, pp. 416–432. [Online]. Available: https://doi.org/10.1007/3-540-39200-9_26

[34] A. D. Santis, S. Micali, and G. Persiano, "Non-interactive zero-knowledge proof systems," in *CRYPTO 1987*, ser. Lecture Notes in Computer Science, C. Pomerance, Ed., vol. 293. Springer, 1987, pp. 52–72. [Online]. Available: https://doi.org/10.1007/3-540-48184-2_5

[35] D. Boneh and M. Naor, "Timed commitments," in *CRYPTO 2000*, ser. Lecture Notes in Computer Science, M. Bellare, Ed., vol. 1880. Springer, 2000, pp. 236–254. [Online]. Available: https://doi.org/10.1007/3-540-44598-6_15

[36] D. Boneh and C. Komlo, "Threshold signatures with private accountability," in *CRYPTO 2022*, ser. Lecture Notes in Computer Science, Y. Dodis and T. Shrimpton, Eds., vol. 13510. Springer, 2022, pp. 551–581. [Online]. Available: https://doi.org/10.1007/978-3-031-15985-5_19

[37] T. Attema and R. Cramer, "Compressed ς-protocol theory and practical application to plug & play secure algorithmics," in *CRYPTO 2020*, ser. Lecture Notes in Computer Science, D. Micciancio and T. Ristenpart, Eds., vol. 12172. Springer, 2020, pp. 513–543. [Online]. Available: https://doi.org/10.1007/978-3-030-56877-1_18

[38] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *SP 2018*, 2018, pp. 315–334.

[39] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT 1999*, ser. Lecture Notes in Computer Science, J. Stern, Ed., vol. 1592. Springer, 1999, pp. 223–238. [Online]. Available: https://doi.org/10.1007/3-540-48910-X_16

[40] G. Malavolta and S. A. K. Thyagarajan, "Homomorphic time-lock puzzles and applications," in *CRYPTO 2019*, ser. Lecture Notes in Computer Science, A. Boldyreva and D. Micciancio, Eds., vol. 11692. Springer, 2019, pp. 620–649. [Online]. Available: https://doi.org/10.1007/978-3-030-26948-7_22

[41] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, ser. Lecture Notes in Computer Science, A. M. Odlyzko, Ed., vol. 263. Springer, 1986, pp. 186–194. [Online]. Available: https://doi.org/10.1007/3-540-47721-7_12

[42] S. Testnet, "tx_a for our evaluation on the ethereum testnet." 2023. [Online]. Available: https://tinyurl.com/dpwb4xdh

[43] ——, "tx_b for our evaluation on the ethereum testnet." 2023. [Online]. Available: https://tinyurl.com/mpjkz37b

[44] ——, "$tx^a_{Pay,i}$ for our evaluation on the ethereum testnet." 2023. [Online]. Available: https://tinyurl.com/9fdzs363

[45] ——, "$tx^b_{RPay,i}$ for our evaluation on the ethereum testnet." 2023. [Online]. Available: https://tinyurl.com/yy5xzzph

[46] ——, "$tx^{A*}_{Pay,i}$ for our evaluation on the ethereum testnet." 2023. [Online]. Available: https://tinyurl.com/2p8933uf

[47] ——, "$tx^{B*}_{RPay,i}$ for our evaluation on the ethereum testnet." 2023. [Online]. Available: https://tinyurl.com/3945tzdt

[48] ——, "$tx^A_{FPay,i}$ for our evaluation on the ethereum testnet." 2023. [Online]. Available: https://tinyurl.com/mvh5wjf5

[49] M. Backes and D. Hofheinz, "How to break and repair a universally composable signature functionality," in *Information Security, 7th International Conference, ISC 2004, Palo Alto, CA, USA, September 27-29, 2004, Proceedings*, ser. Lecture Notes in Computer Science, K. Zhang and Y. Zheng, Eds., vol. 3225. Springer, 2004, pp. 61–72. [Online]. Available: https://doi.org/10.1007/978-3-540-30144-8_6

[50] J. Camenisch, R. Chaabouni, and A. Shelat, "Efficient protocols for set membership and range proofs," in *ASIACRYPT 2008*, ser. Lecture Notes in Computer Science, J. Pieprzyk, Ed., vol. 5350. Springer, 2008, pp. 234–252. [Online]. Available: https://doi.org/10.1007/978-3-540-89255-7_15

Appendix A. 2-party Adaptor Signatures

A.1. Proof of Our Generic Construction

The related lemmas of $\mathsf{aSIG}_2^{\mathsf{G}}$ are proved as follows.

Lemma 2 *The* ***Pre-Signature Correctness*** *of* $\mathsf{aSIG}_2^{\mathsf{G}}$ *can be proved under the assumptions of Theorem 1.*

Proof. As shown in Figure 3, we denote m as an arbitrary message, and $(sk_i, pk_i) \leftarrow \mathsf{KG}(pp)$ for $i \in \{0,1\}$. Then, $\sigma \leftarrow \Gamma_{\mathsf{sign}\langle sk_0, sk_1\rangle}\ (pk_0, pk_1, m)$, $\widehat{\sigma} \leftarrow f_{\mathsf{bnd}}(\sigma, y_0), Y_0 := f_{\mathsf{state}}(pp, y_0)$ and $\pi_0 \leftarrow \mathsf{NIZK.Prove}(crs, Y_0, y_0)$. The first equation $\mathsf{pVerf}_{pk}(Y_0, m, \widehat{\sigma},\ \pi_0) = 1$ is trivial to show with $pk := \mathsf{AKG}(pk_0, pk_1)$, $\mathsf{NIZK.Verf}(crs, Y_0, \mathsf{NIZK.Prove}(crs, Y_0, y_0)) = 1$ and $f_{\mathsf{shift}}(pk, m, \widehat{\sigma}) = f_{\mathsf{shift}}(pk, m,\ f_{\mathsf{bnd}}(\sigma, y)) = f_{\mathsf{state}}(pp, y)$ (cf., Equation 2). Then, it holds that $\sigma := f_{\mathsf{debnd}}(f_{\mathsf{bnd}}(\sigma, y_0), y_0)$ according to Equation 1, and thus the second equation $\mathsf{Verf}_{pk}(m, \mathsf{Adapt}_{pk}(\widehat{\sigma}, y_0)) = 1$ is true. Finally, we show that $(Y_0, y_0') \in \mathsf{R}$ in accordance with $y_0' := f_{\mathsf{ext}}(\widehat{\sigma}, \sigma) = f_{\mathsf{ext}}(\widehat{\sigma}, f_{\mathsf{debnd}}(\widehat{\sigma}, y_0)) = y_0$ (cf., Equation 3).

To establish the **2-aEUF-CMA-Security** of $\mathsf{aSIG}_2^{\mathsf{G}}$, a $\mathcal{PPT}$ adversary $\mathcal{A}$ is assumed to win the $\mathsf{aSigForge}^b_{\mathcal{A},\mathsf{aSIG}_2}$ experiment with non-trivial probability. Using this assumption, we can construct a $\mathcal{PPT}$ simulator that can win the EUF-CMA experiment of Π_{DS} or break the hardness of R with non-trivial probability. Thus, the shown contradiction supports the proof of **2-aEUF-CMA-Security**. The specific proof is provided via the following Lemma.

Lemma 3 *The* ***2-aEUF-CMA-Security*** *of* $\mathsf{aSIG}_2^{\mathsf{G}}$ *can be proved under the assumptions of Theorem 1.*

Proof. We prove this lemma via defining the following games, together with discussing the equivalence among these games, and finally complete the reduction.

Game G_0. This game is the original experiment aSigForge defined in Section 4.1. The adversary $\mathcal{A}$ obtains the partial secret / public key pair (sk_b, pk_b) as well as the other party's public key pk_{1-b}. Also, $\mathcal{A}$ can query the honest relation oracle $\mathcal{O}_H$, the corrupted relation oracle $\mathcal{O}_C$, the pre-signing oracle $\mathcal{O}_{\Gamma_{\mathsf{pS}}}$ and the signing oracle $\mathcal{O}_{\Gamma_{\mathsf{S}}}$. $\mathcal{A}$'s goal is to output a valid forged signature σ^* for its chosen message m^* and statement Y^*. The correspondence between G_0 and aSigForge implies that $\Pr[\mathsf{aSigForge}_{\mathcal{A},\mathsf{aSIG}_2{}^b}(\lambda) = 1] = \Pr[G_0 = 1]$.

Game G_1. Game G_1 is a modified version of G_0, where the only difference is that after $\mathcal{A}$ outputs the forged signature σ^*, this game verifies whether σ^* is equal to the adapted signature from $\widehat{\sigma}$ under y^*. If true, game G_1 halts, and we denote this as event E_1.

Below we prove the probability of E_1 happening is negligible, namely $\Pr[\mathsf{E}_1] \leq \mathsf{negl}(\lambda)$. We assume that $\mathcal{A}$ outputs σ^* satisfying E_1, and we simulate $\mathcal{S}$ to solve the computational problem of R upon $\mathcal{A}$ with non-trivial probability. In particular, $\mathcal{S}$ generates $(sk_{1-b}, pk_{1-b}) \leftarrow \mathsf{KG}(pp)$ and responds to $\mathcal{A}$'s $\mathcal{O}_H$, $\mathcal{O}_C$, $\mathcal{O}_{\Gamma_{\mathsf{S}}}$ and $\mathcal{O}_{\Gamma_{\mathsf{pS}}}$ queries as presented in G_1. After obtaining the challenged m^* and Y^* from $\mathcal{A}$, $\mathcal{S}$ randomly chooses $\widehat{\sigma}$, and computes $\pi^* \leftarrow \mathsf{NIZK.Sim}(crs, Y^*)$ such that $\mathsf{pVerf}_{pk}(Y^*, m^*, \widehat{\sigma}, \pi^*) = 1$. Then, $\mathcal{A}$ will output a forgery σ^* where E_1 happens, namely, $\mathsf{Adapt}_{pk}(\widehat{\sigma}, y^*) = \sigma^*$. Upon the pre-signature correctness, $\mathcal{S}$ can extract y^* for $(Y^*, y^*) \in \mathsf{R}$ via invoking $\mathsf{Ext}_{pk}(\sigma^*, \widehat{\sigma})$.

$\mathcal{A}$'s view in the above simulation and G_1 is indistinguishable, since Y^* is an instance of R and has the same probability distribution as the public output of RG. This means that the probability of S solving the computational problem of R is equal to the probability of event E_1 happening. Thus, we summarize that E_1 only occurs with negligible probability, and further $\Pr[G_0 = 1] = \Pr[G_1 = 1] + \Pr[\mathsf{E}_1] \leq \Pr[G_1 = 1] + \mathsf{negl}(\lambda)$.

Game G_2. Games G_2 and G_1 are analogous except that there is a modification of $\mathcal{O}_{\Gamma_{pS}}$ in G_2. Upon the queried message m and Y_i, the modified $\mathcal{O}_{\Gamma_{pS}}$ first retrieves $(Y_i; y_i) \leftarrow \mathcal{Q}_H \cup \mathcal{Q}_C$. Then the oracle obtains a signature via $\sigma \leftarrow \mathcal{O}_{\Gamma_{\mathsf{S}}}(m)$, and further computes $\widehat{\sigma} \leftarrow f_{\mathsf{bnd}}(\sigma, y_i), \pi_i \leftarrow \mathsf{NIZK.Prove}(crs, Y_i, y_i)$. This will not make the game abort, and thus $\Pr[G_2 = 1] = \Pr[G_1 = 1]$.

Game G_3. This game works as G_2 except the oracle $\mathcal{O}_{\Gamma_{pS}}$ is removed. The indistinguishability between G_2 and G_3 is presented as follows.

In G_2, $\mathcal{A}$ can query two oracles $\mathcal{O}_{\Gamma_S}$ and $\mathcal{O}_{\Gamma_{pS}}$, which means that $\mathcal{A}$ can obtain the witness y_i in $\mathcal{O}_{\Gamma_{pS}}$ upon the adaptability. Thus, $\mathcal{A}$ can only query $\mathcal{O}_{\Gamma_S}$ (i.e., in G_3, the capability of $\mathcal{O}_{\Gamma_{pS}}$ is removed), and then adaptively chooses a statement / witness pair $(Y_i, y_i) \in \mathsf{R}$ to compute $\widehat{\sigma}_i \leftarrow f_{\mathsf{bnd}}(\sigma_i, y_i), \pi_i \leftarrow \mathsf{NIZK.Prove}(crs, Y_i, y_i)$. Here, it follows that $\Pr[G_3 = 1] = \Pr[G_2 = 1]$.

Game G_4. This game is similar to G_3, but it changes the method of generating pre-signature after receiving the challenge message m^* from $\mathcal{A}$. Instead of invoking RG and Γ_{pSign} to obtain the statement and pre-signature, G_4 randomly chooses Y^* and $\widehat{\sigma}$, and computes $\pi^* \leftarrow \mathsf{NIZK.Sim}(crs, Y^*)$ such that $\mathsf{pVerf}_{pk}(Y^*, m^*, \widehat{\sigma}, \pi^*) = 1$. The distribution of $(\widehat{\sigma}, Y^*, \pi^*)$ in G_4 appears identical to that in G_3 from the perspective of $\mathcal{A}$, which is consistent with the zero knowledge of NIZK and the correctness of pVerf. Therefore, we have that $\Pr[G_4 = 1] = \Pr[G_3 = 1]$.

From the above transition, the indistinguishability between the original experiment $\mathsf{aSigForge}$ (game G_0) and the final game G_4 has been shown. Now we only need to show there exists a simulator $\mathcal{S}$ to simulate G_4 completely and further employ $\mathcal{A}$ to win the $\mathsf{SigForge}$ game. In particular, instead of creating the secret / public key and computing the signature via Γ_{Sign}, $\mathcal{S}$ directly adopts its oracle $\mathcal{O}_{\mathsf{SIG}_2}$ in the $\mathsf{SigForge}$ game. Thus, the simulation of G_4 has been achieved by $\mathcal{S}$. With winning the $\mathsf{SigForge}$ game, $\mathcal{S}$ utilizes $\mathcal{A}$'s forgery (m^*, σ^*) as its answer to the $\mathsf{SigForge}$ game. It should be noted that $\mathcal{A}$ wins $\mathsf{aSigForge}$ only if it has not requested m^* to neither $\mathcal{O}_{\mathsf{pS}}$ nor $\mathcal{O}_{\mathsf{S}}$. Consequently, m^* has not been queried to $\mathcal{O}_{\mathsf{SIG}_2}$ either, and as a result, (m^*, σ^*) is a valid solution to the $\mathsf{SigForge}$ game.

In summary of the above games and simulation, we have $\Pr[G_0 = 1] \leq \Pr[G_4 = 1] + \mathsf{negl}(\lambda)$. Due to the perfect simulation of G_4 by $\mathcal{S}$, it also follows that $\Pr[G_4 = 1] = \Pr[\mathsf{SigForge}_{\mathcal{S}^{\mathcal{A}}, \mathsf{SIG}_2}(\lambda) = 1]$. Thus, we have that

$$\Pr[\mathsf{aSigForge}_{\mathcal{S}^{\mathcal{A}}, \mathsf{aSIG}_2}(\lambda) = 1] \tag{4}$$
$$\leq \Pr[\mathsf{SigForge}_{\mathcal{S}^{\mathcal{A}}, \mathsf{SIG}_2}(\lambda) = 1] + \mathsf{negl}(\lambda). \tag{5}$$

Recall that the negligible function $\mathsf{negl}(\lambda)$ precisely captures $\mathcal{A}$'s advantage in solving the computational problem represented by R. Therefore, the probability of a successful attack on the 2-$\mathsf{aEUF\text{-}CMA}$-security of $\mathsf{aSIG}_2^{\mathsf{G}}$ is bounded by the maximum probability of successfully attacking either the hardness of R or the $\mathsf{EUF\text{-}CMA}$-security of SIG_2.

Lemma 4 *The **Pre-Signature Adaptability** of* $\mathsf{aSIG}_2^{\mathsf{G}}$ *can be proved under the assumptions of Theorem 1.*

Proof. Assume that $\mathsf{pVerf}_{pk}(Y_0, m, \widehat{\sigma}, \pi_0) = 1$, then $\mathsf{NIZK.Verf}(crs, Y_0, \pi_0) = 1$ and $f_{state}(pp, y_0) = f_{shift}(pk, m, \widehat{\sigma})$ hold. We can further recover $\sigma := f_{\mathsf{debnd}}(f_{\mathsf{bnd}}(\sigma, y_0), y_0)$, cf., Equation 1. Also, σ is generated from $\Gamma_{\mathsf{pSign}\langle sk_0, y_0; sk_1\rangle}(pp, pk_0, pk_1, m)$, and hence we have $\mathsf{Verf}_{\mathsf{AKG}(pk_0, pk_1)}(m, \sigma) = 1$. Thus, valid pre-signatures can always be adapted to valid signatures.

Lemma 5 *The **Witness Extractability** of* $\mathsf{aSIG}_2^{\mathsf{G}}$ *can be proved under the assumptions of Theorem 1.*

Proof. The proof of this lemma is similar to that of Lemma 3. The difference lies in the fact that the witness extractability can be reduced to only the $\mathsf{SigForge}$ experiment. The $\mathsf{aWitExt}_{\mathcal{A}, \mathsf{aSIG}_2}$ experiment requires $\mathcal{A}$'s forgery to satisfy $\mathsf{Verf}_{\mathsf{AKG}(pk_0, pk_1)}(m^*, \sigma^*) = 1$ and also $(Y^*, \mathsf{Ext}_{\mathsf{AKG}(pk_0, pk_1)}(\sigma^*, \widehat{\sigma}^*)) \notin \mathsf{R}$. This implies that $\mathcal{A}$'s forgery can be regarded as the simulator's valid forgery in the $\mathsf{SigForge}$ experiment.

A.2. Instantiations

Instantiation 1 (ECDSA signatures). Denote $\mathbb{G} = \langle G \rangle$ as a q-prime-order additive cyclic group, (d_0, P_0), and (d_1, P_1) as two public / secret key pairs with $P_i = d_i G, \forall i \in \{0, 1\}$. We also let $\mathsf{Uncompress}$ be a point uncompression function that inputs the x coordinate and the y coordinate sign $b \in \{+, -\}$, and it outputs the y coordinate. Assume that $\sigma = (r, s)$ is a ECDSA signature output by Γ_{Sign}, and hence the equations $K = (x_K, y_K) = s^{-1}(\mathcal{H}(m)G + rP)$ and $x_K = r \pmod q$ hold. Here, m is the message, $P = d_0 P_1 = d_1 P_0$, and $\mathcal{H}$ is a cryptographic hash function that maps input strings from $\{0, 1\}^*$ to elements in $\mathbb{Z}_q^*$. Whereafter, we define Γ_{pSign}, pVerf, Adapt and Ext in Figure 8.

We instantiate the relation $\mathsf{R} = \{(Y; y) | Y = yG\}$ as above, where R consists of group elements and their corresponding discrete logarithms. We define f_{bnd}, f_{debnd}, f_{ext}, f_{state}, and f_{shift} as follows.

$$f_{\mathsf{bnd}}(s, y) := s \cdot y^{-1} \pmod q, f_{\mathsf{debnd}}(\widehat{s}, y) := \widehat{s} \cdot y \pmod q,$$
$$f_{\mathsf{ext}}(\widehat{s}, s) := \widehat{s}^{-1} \cdot s \pmod q, f_{\mathsf{state}}(K, y) := yK,$$
$$f_{\mathsf{shift}}(pk, m, \widehat{\sigma}) := \widehat{s}^{-1}(\mathcal{H}(m)G - rP), \text{where } \widehat{\sigma} := (r, \widehat{s}).$$

It is intuitive to prove Equations 1 and 3, according to $\forall s, y \in \mathbb{Z}_q^*, f_{\mathsf{debnd}}(f_{\mathsf{bnd}}(s, y), y) = (s \cdot y^{-1}) \cdot y = s \pmod q$,

$\underline{\Gamma_{\mathsf{pSign}\langle d_0,y_0;d_1\rangle}(pp,P_0,P_1,m)}$
$(r,s) \leftarrow \Gamma_{\mathsf{sign}\langle d_0,d_1\rangle}(P_0,P_1,m)$
$\widehat{s} = s\cdot y_0^{-1} \pmod q$
$K = s^{-1}(\mathcal{H}(m)G + rP)$
$K' = \mathsf{Uncompress}(r,+)$
if $K' = K$ **then** $b = +$
else $b = -$
$Y_0 = y_0K$
$r_0 \in \mathbb{Z}_q^*, R_0 = r_0K$
$c_0 = \mathcal{H}(R_0, Y_0, K)$
$z_0 = r_0 - c_0\cdot y_0 \pmod q$
$\pi_0 = (c_0, z_0, b)$
return $(\widehat{\sigma} = (r,\widehat{s}), Y_0, \pi_0)$

$\underline{\mathsf{Adapt}_P(\widehat{\sigma}, y_0)}$
parse $\widehat{\sigma} = (r,\widehat{s})$
$s = \widehat{s}\cdot y_0 \pmod q$
return (r,s)

$\underline{\mathsf{pVerf}_P(Y_0, m, \widehat{\sigma}, \pi_0)}$
parse $\widehat{\sigma} = (r,\widehat{s})$
parse $\pi_0 = (c_0, z_0, b)$
$K = \mathsf{Uncompress}(r,b)$
$R_0' = z_0K + c_0Y_0$
$c_0' = \mathcal{H}(R_0', Y_0, K)$
if $c_0' \neq c_0$ **then return** 0
$Y_0' = \widehat{s}^{-1}(\mathcal{H}(m)G + rP)$
return $(Y_0' == Y_0)$

$\underline{\mathsf{Ext}_P(\sigma, \widehat{\sigma})}$
parse $\sigma = (r,s)$
parse $\widehat{\sigma} = (r,\widehat{s})$
$y_0 = \widehat{s}^{-1}\cdot s \pmod q$
return y

Figure 8: 2-party ECDSA adaptor signature scheme

$\underline{\Gamma_{\mathsf{pSign}\langle d_0,y_0;d_1\rangle}(pp,P_0,P_1,m)}$
$\sigma \leftarrow \Gamma_{\mathsf{sign}\langle d_0,d_1\rangle}(P_0,P_1,m)$
$\widehat{\sigma} = \sigma + y_0, Y_0 = e(y_0, G_2)$
$r_0 \in \mathbb{G}_1, R_0 = e(r_0, G_2)$
$c_0 = \mathcal{H}(R_0, Y_0, G_1, G_2, G_T)$
$Z_0 = r_0 - c_0y_0$
$\pi_0 = (c_0, Z_0)$
return $(\widehat{\sigma}, Y_0, \pi_0)$

$\underline{\mathsf{Adapt}_P(\widehat{\sigma}, y_0)}$
$\sigma = \widehat{\sigma} - y_0$
return σ

$\underline{\mathsf{pVerf}_P(Y_0, m, \widehat{\sigma}, \pi_0)}$
parse $\pi_0 = (c_0, Z_0)$
$R_0' = e(Z_0, G_2)\cdot Y_0^{c_0}$
$c_0' = \mathcal{H}(R_0', Y_0, G_1, G_2, G_T)$
if $c_0' \neq c_0$ **then return** 0
$Y_0' = e(\widehat{\sigma}, G_2)/e(\mathcal{H}_p(m), P)$
return $(Y_0' == Y_0)$

$\underline{\mathsf{Ext}_P(\sigma, \widehat{\sigma})}$
$y_0 = \widehat{\sigma} - \sigma$
return y_0

Figure 9: 2-party BLS adaptor signature scheme

$f_{\mathsf{ext}}(f_{\mathsf{bnd}}(s,y),s) = (s\cdot y^{-1})^{-1}\cdot s = y \pmod q$. With regard to Equation 2, let us arbitrarily select public key $P \in \mathbb{G}$, a correct ECDSA signature (r,s) with $K = (x_K, y_K) = s^{-1}(\mathcal{H}(m)G + rP)$ and $x_K = r \pmod q$, and a statement / witness pair $(Y;y) \in \mathsf{R}$ (i.e., $Y = yK$). Then we have

$$\begin{aligned} f_{\mathsf{shift}}(P, m, (r, f_{\mathsf{bnd}}(s,y))) &= (sy^{-1})^{-1}(\mathcal{H}(m)G + rP) \\ &= yK = f_{\mathsf{state}}(K,y), \end{aligned}$$

and hence Equation 2 also holds.

Instantiation 2 (BLS signatures). Denote $\mathbb{G}_1 = \langle G_1\rangle$ and $\mathbb{G}_2 = \langle G_2\rangle$ as cyclic additive groups, $\mathbb{G}_T = \langle G_T\rangle$ as a cyclic multiplicative group, all of which are with prime order q. A bilinear pairing is denoted as $e: \mathbb{G}_1\times\mathbb{G}_2 \to \mathbb{G}_T$ and $G_T = e(G_1, G_2)$. Also, let (d_0, P_0) and (d_1, P_1) be two public / secret key pairs with $P_i = d_iG_2, \forall i \in \{0,1\}$, σ is a BLS signature output by Γ_{Sign}, namely, $e(\sigma, G_2) = e(\mathcal{H}_p(m), P)$. Here, m is the message, $P = P_0 + P_1$, and $\mathcal{H}_p : \{0,1\}^* \to \mathbb{G}_1$ is a secure hash-to-point function. Γ_{pSign}, pVerf, Adapt, Ext are defined in Figure 9.

The relation R above is realized as $\mathsf{R} = \{((Y;y)|Y = e(y, G_2))\}$, namely, $\mathbb{G}_T$ elements and corresponding $\mathbb{G}_1$ elements with a given element G_2 in $\mathbb{G}_2$. We define f_{bnd}, f_{debnd}, f_{ext}, f_{state} and f_{shift} as follows.

$$\begin{aligned} &f_{\mathsf{bnd}}(s,y) := s + y, f_{\mathsf{debnd}}(\widehat{s},y) := \widehat{s} - y, \\ &f_{\mathsf{ext}}(\widehat{s},s) := \widehat{s} - s, f_{\mathsf{state}}(G_2,y) := e(y,G_2), \\ &f_{\mathsf{shift}}(pk,m,\widehat{s}) := e(\widehat{s},G_2)\cdot e(\mathcal{H}_p(m),pk)^{-1}. \end{aligned}$$

Equations 1 and 3 are also trivial to prove, that is, $\forall s,y \in \mathbb{G}_1,\ f_{\mathsf{debnd}}(f_{\mathsf{bnd}}(s,y),y) = (s+y) - y = s$ and $f_{\mathsf{ext}}(f_{\mathsf{bnd}}(s,y),s) = (s+y) - s = y$. With regard to Equation 2, let us select public key $P_2 \in \mathbb{G}_2$, a BLS signature σ with $e(\sigma, G_2) = e(\mathcal{H}_p(m)G, P_2)$, and a statement / witness pair $(Y;y) \in \mathsf{R}$ (i.e., $Y = e(y, G_2)$). Then we have

$$\begin{aligned} f_{\mathsf{shift}}(P_2, m, f_{\mathsf{bnd}}(\sigma,y)) &= e(s+y, G_2)\cdot e(\mathcal{H}_p(m), P_2)^{-1} \\ &= yK = f_{\mathsf{state}}(K,y), \end{aligned}$$

and hence Equation 2 also holds.

Instantiation 3 (Schnorr signatures). Denote $\mathbb{G} = \langle G\rangle$ as a q-prime-order additive cyclic group, (d_0, P_0) and (d_1, P_1) as two public / secret key pairs with $P_i = d_iG, \forall i \in \{0,1\}$. Assume that $\sigma = (R,s)$ is a signature output by Γ_{Sign}, and hence the equation $R = sG + \mathcal{H}(m,R)P$ holds. Here, m is the message, $P = P_0 + P_1$, and $\mathcal{H}: \{0,1\}^* \to \mathbb{Z}_q^*$ is the hash function as defined in ECDSA. Whereafter, we define Γ_{pSign}, pVerf, Adapt, and Ext in Figure 10.

$\underline{\Gamma_{\mathsf{pSign}\langle d_0,y_0;d_1\rangle}(pp,P_0,P_1,m)}$
$(R,s) \leftarrow \Gamma_{\mathsf{sign}\langle d_0,d_1\rangle}(P_0,P_1,m)$
$\widehat{s} = s + y_0 \pmod q$
$Y_0 = y_0G$
$r_0 \in \mathbb{Z}_q^*, R_0 = r_0G$
$c_0 = \mathcal{H}(R_0, Y_0, G)$
$z_0 = r_0 - c_0\cdot y_0 \pmod q$
$\pi_0 = (c_0, z_0)$
return $(\widehat{\sigma} = (R,\widehat{s}), Y_0, \pi_0)$

$\underline{\mathsf{Adapt}_P(\widehat{\sigma}, y_0)}$
parse $\widehat{\sigma} = (R,\widehat{s})$
$s = \widehat{s} - y_0 \pmod q$
return (R,s)

$\underline{\mathsf{pVerf}_P(Y_0, m, \widehat{\sigma}, \pi_0)}$
parse $\pi_0 = (c_0, z_0)$
parse $\widehat{\sigma} = (R,\widehat{s})$
$R_0' = z_0G + c_0Y_0$
$c_0' = \mathcal{H}(R_0', Y_0, G)$
if $c_0' \neq c_0$ **then return** 0
$Y_0' = \widehat{s}G - \mathcal{H}(m,R)P$
return $(Y_0' = Y_0)$

$\underline{\mathsf{Ext}_P(\sigma, \widehat{\sigma})}$
parse $\sigma = (R,s)$
parse $\widehat{\sigma} = (R,\widehat{s})$
$y_0 = \widehat{s} - s \pmod q$
return y_0

Figure 10: 2-party Schnorr adaptor signature scheme

In the above instantiation, we initiate the relation $\mathsf{R} = \{(Y;y)|Y = yG\}$, where R consists of group elements and their corresponding discrete logarithms. We define f_{bnd}, f_{debnd}, f_{ext}, f_{state} and f_{shift} as follows.

$$\begin{aligned} &f_{\mathsf{bnd}}(s,y) := s + y \pmod q, f_{\mathsf{debnd}}(\widehat{s},y) := \widehat{s} - y \pmod q, \\ &f_{\mathsf{ext}}(\widehat{s},s) := \widehat{s} - s \pmod q, f_{\mathsf{state}}(G,y) := yG, \\ &f_{\mathsf{shift}}(pk,m,\widehat{\sigma}) := \widehat{s}G - R - \mathcal{H}(m,R)P, \text{ where } \widehat{\sigma} = (R,\widehat{s}). \end{aligned}$$

It is intuitive to prove that Equation 1 and 3 hold, according to $\forall s,y \in \mathbb{Z}_q^*, f_{\mathsf{debnd}}(f_{\mathsf{bnd}}(s,y),y) = (s+y) - y = s \pmod q$ and $f_{\mathsf{ext}}(f_{\mathsf{bnd}}(s,y),s) = (s+y) - s = y \pmod q$. With regard to Equation 2, let us arbitrarily select public key $P \in \mathbb{G}$, a correct Schnorr signature (R,s) with

$sG = R + \mathcal{H}(m, R)P$, and a statement / witness pair $(Y; y) \in \mathsf{R}$ (i.e., $Y = yG$). Then we have

$$f_{\mathsf{shift}}(P, m, (R, f_{\mathsf{bnd}}(s, y))) = (s + y)G - R - \mathcal{H}(m, R)P \\ = yG = f_{\mathsf{state}}(G, y),$$

and hence Equation 2 also holds.

Appendix B. Ideal Functionality

In this section, we formally define bi-PC in the account model as an ideal functionality $\mathcal{F}_{AM}$, which closely follows the definition in [8]. We capture not only the security and efficiency notations, but also *a strong constraint* among the account model for ensuring the delayed finality with punishment. Once one party posts an old payment state on the chain, the other can have until timeout $\mathbf{T}_d$ to punish this misbehavior. Specially, we sidestep the setting of a negligible amount ε, but take advantage of nonce mechanism in the account model to achieve a stronger constraint among transactions. To avoid losing their balance due to misbehavior, channel owners typically close the channel using the most recent payment state.

Notations. We denote $\chi := (\chi.\mathsf{id}, \chi.\mathsf{users}, \chi.\mathsf{cash}, \chi.\mathsf{st}, \chi.\mathbf{T}_d, \chi.\mathsf{c})$ as the attribute set of a channel, where $\chi.\mathsf{id} \in \{0,1\}^*$ is the identity, $\chi.\mathsf{users}$ defines the joined two parties, $\chi.\mathsf{cash} \in \mathbb{R}_{\geq 0}$ is the maximum amount of funds to transfer, $\chi.\mathsf{st}$ is the channel state composed by a list of accounts (where each account is with an address and an associated value), $\chi.\mathbf{T}_d \in \mathbb{R}_{\geq 0}$ is the channel lifetime expressed as a non-negative real number (signifying the absolute timeout), and $\chi.\mathsf{c} \in \mathbb{R}_{\geq 0}$ is the collateral prepaid by each party.

The notation $m \overset{\tau}{\hookrightarrow} P$ denotes a message m is sent to party P at round τ, and $m \overset{\tau}{\hookleftarrow} P$ a message m received from party P at round τ. Here, message m contains an identity `MESSAGE-ID` and associated parameters ps, but we omit the identity for better readability. Our communication model assumes that party B will receive the message m in round $\tau + 1$, if party A sends m to him in round τ. Nevertheless, messages transmitted to the environment $\mathcal{E}$, the simulator $\mathcal{S}$ or ideal functionality $\mathcal{F}_{AM}$ are assumed to be received immediately in the same round.

Concise Overview. In our $\mathcal{F}_{AM}$, we do not take into account privacy, and hence $\mathcal{F}_{AM}$ directly forwards messages to simulator $\mathcal{S}$. In particular, $\mathcal{S}$ is expected to perform its responsible tasks, including the creation of signature, preparation of transaction identities, and providing / setting requested values. $\mathcal{F}_{AM}$ will return `ERROR` if $\mathcal{S}$ does not complete these tasks. The returned `ERROR` means that the aforementioned properties of payments channel are lost, and thus we focus on realizing $\mathcal{F}_{AM}$ that never returns `ERROR`.

In the interaction between $\mathcal{F}_{AM}$ and ledger $\mathbb{L}(\Delta, \Sigma, \mathcal{V})$, Δ represents an initialized upper bound (after which valid transactions are chained to the ledger). Additionally, Σ and $\mathcal{V}$ denote a signature algorithm and its corresponding verification for a transaction, ensuring the validity of signature, nonce, and absolute timelock. The notation of $\mathcal{F}_{AM}(\tau_p, k)$ is followed by that defined in [8], where parameter τ_p represents an upper bound on consecutive off-chain communication rounds between two users, and parameter k is the total state amount of a channel. Also, $\Psi(\mathsf{id}) \to (sts, addr)$ denotes a map from a channel identity id to channel states sts (e.g., $\{\chi, \chi'\}$) and channel address $addr$ (e.g., Ch_{AB}). The former definition of $\mathcal{F}_{AM}(\tau_p, k)^{\mathbb{L}(\Delta,\Sigma,\mathcal{V})}$ (abbreviated as $\mathcal{F}_{AM}$) is shown in Figure 11, and the explanation of security and efficiency is provided as follows.

Create. When $\mathcal{F}_{AM}$ receives two creating requests (`CREATE`, χ, tid_A) and (`CREATE`, χ, tid_B) from parties A and B, respectively, it expects to retrieve a funding transaction $tx_F^{A,B}$ from $\mathbb{L}$ within Δ rounds. Then, $\mathcal{F}_{AM}$ will hold $\chi.\mathsf{cash} + 2\chi.\mathsf{c}$ in Ch_{AB} via recording them in Ψ, and finally sends `CREATED` to both parties A and B.

Update. Either party can initiate this update via (`Update`, id, $\overrightarrow{\theta}$, τ_{stp}), among which id represents the channel identity, $\overrightarrow{\theta}$ is the new channel state (i.e., allocation of funds in related addresses of the channel), and τ_{stp} is the time for handling anything required during updation. When two parties agree to update the channel state, $\mathcal{S}$ prepares a vector of k transactions $\overrightarrow{tid}$ for $\mathcal{F}_{AM}$. Now, the initiator (e.g., party A) can abort with not sending `SETUP-OK`, and the other (resp., party B) can abort with not sending `UPDATE-OK`. Otherwise, the parties will move on to the revocation if the initiator receives `UPDATE-OK`. Only when $\mathcal{F}_{AM}$ receives `REVOKE` from both parties, the update is successful and both parties will receive `UPDATED`. Once an error happens, $\mathcal{F}_{AM}$ will invoke subroutine `ForceClose` to ensure that the funding transaction of channels is spent within Δ rounds.

Close. Both parties (e.g., party A) can initiate this closing via (`CLOSE`, id), and then $\mathcal{F}_{AM}$ expects to retrieve transaction tx_C (signifying the latest channel state) on $\mathbb{L}$ if it receives the closing message from the other party (resp., party B) within τ_p rounds. Additionally, there may be a scenario where one party is corrupted, and $\mathcal{F}_{AM}$ expects an older state transaction and a punishment transaction (shown in Punish) to be posted. $\mathcal{F}_{AM}$ returns `ERROR` if the funding transaction is still not spent after the above closure.

Punish. The punishing phase ensures that honest parties can either receive a refund of the locked balance with the latest channel state by time $\mathbf{T}_d$ or obtain all the balance and collateral through a punishment transaction. In the UC framework, this phase is incorporated by having environment $\mathcal{E}$ check for its completion in each round. If the expectation is not met, $\mathcal{F}_{AM}$ signals an error the next time it receives the input. The ideal functionality $\mathcal{F}_{AM}$ expects a transaction to either assign coins corresponding to the latest state of χ or transfer $\chi.\mathsf{cash} + \chi.\mathsf{c}$ coins to the honest party. If neither of these conditions is met, $\mathcal{F}_{AM}$ returns `ERROR`.

If a transaction on $\mathbb{L}$ assigns coins based on the most recent state of χ, there are two cases. The first is when the initiator of tx_{Pay}^{P} (e.g., $P := A$) has published the transaction locked until $\mathbf{T}_d$. The second is when the other party (i.e., party B) has posted transactions tx_{RPay}^{B} and tx_{RPay}^{B*} to unlock their balance and collateral. Importantly, Create: When receiving $(\texttt{CREATE}, \chi, pk_A) \xhookleftarrow{\tau_0} A$, distinguish:

Both approved: In the case a message $(\texttt{CREATE}, \chi, pk_B) \xhookleftarrow{\tau} B$ has already been received, where $\tau_0 - \tau \le \tau_p$: If $tx_F^{A,B} := (tid_A, tid_B)$ with $tid_A := tx(pk_A, \mathsf{Ch}_{AB}, v_A + c, n_A)$, $tid_B := tx(pk_B, \mathsf{Ch}_{AB}, v_B + c, n_B)$, and $\chi.\mathsf{cash} + 2\chi.\mathsf{c} = (v_A + c) + (v_B + c)$ is chained on $\mathbb{L}$ in round $\tau_1 \le \tau + \Delta + \tau_p$, then let $\Psi(\chi.\mathsf{id}) := (\{\chi\}, \mathsf{Ch}_{AB})$ and $(\texttt{CREATED}, \chi.\mathsf{id}) \xhookrightarrow{\tau_1} \chi.\mathsf{users}$. Else halt.

Wait for B: Otherwise wait if received $(\texttt{CREATE}, id) \xhookleftarrow{\tau \le \tau_0 + \tau_p} B$ (then, the case of "Both approved" happens). Halt if such a message is not detected.

Update: According to $(\texttt{UPDATE}, \mathsf{id}, \overrightarrow{\theta}, \tau_{stp}) \xhookleftarrow{\tau_0} A$, parse $(\{\chi\}, \mathsf{Ch}_{AB}) := \Psi(\mathsf{id})$, let $\chi' := \chi, \chi'.\mathsf{st} := \overrightarrow{\theta}$:

1. In round $\tau_1 \le \tau_0 + \tau_p$, let $\mathcal{S}$ record $\overrightarrow{tid}$ s.t. $|\overrightarrow{tid}| = k$. After that, $(\texttt{UPDATE-REQ}, \mathsf{id}, \overrightarrow{\theta}, \tau_{stp}, \overrightarrow{tid}) \xhookrightarrow{\tau_1} B$ and $(\texttt{SETUP}, \mathsf{id}, \overrightarrow{tid}) \xhookrightarrow{\tau_1} A$.
2. If $(\texttt{SETUP-OK}, \mathsf{id}) \xhookleftarrow{\tau_2 \le \tau_1 + \tau_{stp}} A$, then $(\texttt{SETUP-OK}, \mathsf{id}) \xhookrightarrow{\tau_3 \le \tau_2 + \tau_p} B$. Otherwise halt.
3. If $(\texttt{UPDATE-OK}, \mathsf{id}) \xhookleftarrow{\tau_3} B$, then (if B is honest or under the control of $\mathcal{S}$) send $(\texttt{UPDATE-OK}, \mathsf{id}) \xhookleftarrow{\tau_4 \le \tau_3 + \tau_p} A$. Else distinguish:
   - If B is honest or under the control of $\mathcal{S}$, halt (*reject*).
   - Else let $\Psi(\mathsf{id}) := (\{\chi, \chi'\}, \mathsf{Ch}_{AB})$, invoke $\texttt{ForceClose}(\mathsf{id})$ and halt.
4. If $(\texttt{REVOKE}, \mathsf{id}) \xhookleftarrow{\tau_4} A$, send $(\texttt{REVOKE-REQ}, \mathsf{id}) \xhookrightarrow{\tau_5 \le \tau_4 + \tau_p} B$. Otherwise let $\Psi(\mathsf{id}) := (\{\chi, \chi'\}, \mathsf{Ch}_{AB})$, invoke $\texttt{ForceClose}(\mathsf{id})$ and halt.
5. If $(\texttt{REVOKE}, \mathsf{id}) \xhookleftarrow{\tau_5} B$, $\Psi(\mathsf{id}) := (\{\chi'\}, \mathsf{Ch}_{AB})$, transmit $(\texttt{UPDATED}, \mathsf{id}, \overrightarrow{\theta}) \xhookrightarrow{\tau_6 \le \tau_5 + \tau_p} \chi.\mathsf{users}$ and halt (*accept*). Otherwise let $\Psi(\mathsf{id}) := (\{\chi, \chi'\}, \mathsf{Ch}_{AB})$, invoke $\texttt{ForceClose}(\mathsf{id})$ and halt.

Close: When receiving $(\texttt{CLOSE}, \mathsf{id}) \xhookleftarrow{\tau_0} A$, distinguish:

Both approved: In the case a message $(\texttt{CLOSE}, \mathsf{id}) \xhookleftarrow{\tau} B$ has already been received, where $\tau_0 - \tau \le \tau_p$, parse $(\{\chi\}, \mathsf{Ch}_{AB}) := \Psi(\mathsf{id})$ and distinguish:

- If $tx_C := (tid'_A, tid'_B)$ appears on $\mathbb{L}$ in round $\tau_1 \le \tau_0 + \Delta$, where $tid'_A := tx(\mathsf{Ch}_{AB}, pk_A, \chi.\mathsf{c} + \chi.\mathsf{st.bal}(A), b)$, $tid'_B := tx(\mathsf{Ch}_{AB}, pk_B, \chi.\mathsf{c} + \chi.\mathsf{st.bal}(B), 1 - b)$ and $b \in \{0, 1\}$ is the nonce of Ch_{AB}. Then set $\Psi(\mathsf{id}) := \emptyset$, transmit $(\texttt{CLOSED}, \mathsf{id}) \xhookrightarrow{\tau_1} \chi.\mathsf{users}$ and halt.
- Otherwise, if either of the parties is corrupted, invoke $\texttt{ForceClose}(\mathsf{id})$. Else, send $(\texttt{ERROR}) \xhookrightarrow{\tau_0 + \Delta} \chi.\mathsf{users}$ and halt.

Wait for B: Otherwise wait if $(\texttt{CLOSE}, \mathsf{id}) \xhookleftarrow{\tau \le \tau_0 + \tau_p} B$ (then, the case of "Both approved" happens). Terminate if such a message is not detected, invoke $\texttt{ForceClose}(\mathsf{id})$ in round $\tau_0 + \tau_p$.

Punish: (Performed after each round τ_0) $\forall (X, \mathsf{Ch}_{AB}) \in \Psi$, check if $\mathbb{L}$ lists transactions $tx_{Pay}^A := tx(\mathsf{Ch}_{AB}, pk_S, v'_A, 0)$ and $tx_{Pay}^{A*} := tx(pk_S, pk_R, v_A, 0)$ for some (pk_S, pk_R), $v'_A - v_A = \chi.\mathsf{c}$ and $A \in \chi.\mathsf{users}, B \in \chi.\mathsf{users} \backslash \{A\}$. If yes, then let $L := \{\chi.st | \chi \in X\}$ and distinguish:

Punish: If B behaves honestly and $(tx_{Pay}^A, tx_{Pay}^{A*})$ does not match the latest state in X, $tx_{Pnsh,i}^B := tx(pk_R, pk_B, \chi.\mathsf{st.bal}(A), 0)$ is chained on $\mathbb{L}$ in round $\tau_1 \le \tau_0 + \Delta$. Then in round $\tau_2 \le \tau_1 + \Delta$, two transactions $tx_{RPay}^B := tx(\mathsf{Ch}_{AB}, pk_O, v'_B, 1)$ and $tx_{RPay}^{B*} := tx(pk_O, pk_B, v'_B, 0)$ for some address pk_O and $v'_B = \chi.\mathsf{st.bal}(B) + \chi.\mathsf{c}$, appear on $\mathbb{L}$, let $\Psi(\mathsf{id}) := \emptyset$, transmit $(\texttt{PUNISHED}, \mathsf{id}) \xhookrightarrow{\tau_2}$ B and halt.

Close: Either $\Psi(\mathsf{id}) := \emptyset$ before round $\tau_0 + \Delta$ (i.e., the channel was closed without any dispute) or after round $\tau_1 \le \tau_0 + \Delta$ two transactions $tx_{RPay}^B := tx(\mathsf{Ch}_{AB}, pk_O, v'_B, 1)$ and $tx_{RPay}^{B*} := tx(pk_O, pk_B, v'_B, 0)$ for some address pk_O, and $v'_B = \chi.\mathsf{st.bal}(B) + \chi.\mathsf{c}$, appear on $\mathbb{L}$ before three transactions $tx_{Pay}^A := tx(\mathsf{Ch}_{AB}, pk_S, v'_A, 0)$, $tx_{Pay}^{A*} := tx(pk_S, pk_R, v_A, 0)$, and $tx_{FPay}^A := tx(pk_R, pk_A, \chi.\mathsf{st.bal}(A), 0)$ for some addresses (pk_S, pk_R), appear on $\mathbb{L}$. Set $\Psi(\mathsf{id}) := \emptyset$ and transmit $(\texttt{CLOSED}, \mathsf{id}) \xhookrightarrow{\tau_2 \le \tau_1 + \Delta} \chi.\mathsf{users}$. Otherwise, transaction $tx_D^A := tx(pk_R, pk_A, \chi.\mathsf{st.bal}(A), 0)$ is chained on $\mathbb{L}$ in round $\tau_3 \le \chi.T + \Delta$. Let $\Psi(\mathsf{id}) := \emptyset$ and send $(\texttt{CLOSED}, \mathsf{id}) \xhookrightarrow{\tau_3} \chi.\mathsf{users}$ and halt.

Error: Otherwise, output $(\texttt{ERROR}) \xhookrightarrow{\tau_0 + \Delta} \chi.\mathsf{users}$.

Subroutine $\texttt{ForceClose}(\mathsf{id})$: Denote τ_0 as the present round and $(\chi, tx) := \Psi(\mathsf{id})$. If tx remains unspent on $\mathbb{L}$ for Δ rounds, then $(\texttt{ERROR}) \xhookrightarrow{\tau_0 + \Delta} \chi.\mathsf{users}$ and halt. Otherwise, in latest round $\chi.\tau + \Delta$, message $m \in \{\texttt{CLOSED}, \texttt{PUNISHED}, \texttt{ERROR}\}$ is returned by Punish.

Figure 11: Ideal Functionality $\mathcal{F}_{AM}(\tau_p, k)^{\mathbb{L}(\Delta, \Sigma, \mathcal{V})}$ of bi-PC for account model

party B does not lose coins to party A in this scenario.

Appendix C. UC Protocol

The UC framework was introduced in Appendix B, where we defined notation and described a protocol that can be modeled within this framework. To capture any aspect that goes beyond the protocol execution and communication model, we model the environment. We also replace the 2-party cryptographic protocols in Π_{AS}, namely aggregated key generation Γ_{AKG}, signing Γ_{Sign}, and pre-signing Γ_{pSign} with idealized versions $\mathcal{F}_{\mathsf{AKG}}$, $\mathcal{F}_{\mathsf{Sign}}$, and $\mathcal{F}_{\mathsf{pSign}}$, respectively. Lastly, we incorporate the feature of being able to close PC in an honest manner with a single on-chain transaction. This is achieved by constructing a transaction that spends from the funding transaction and immediately distributes each user's balance.

To make the protocol more concise, we assume that honest users perform certain checks that are typically carried out, such as validating input parameters, checking the availability of the channels to be updated or closed, verifying the validity of the new state, and confirming the availability of sufficient funds. These checks can be performed via a protocol wrapper that filters out invalid messages from the environment. We adopt the wrapper defined in [7] for PC and employ the same approach for the ideal functionality.

Sleepy channel protocol Π_{B}

Create

Party A receives $(\texttt{CREATE}, \mathsf{id}, \chi, pk_A) \xleftarrow{\tau_0} \mathcal{E}$

1. Generate $(pk^A_{\mathsf{FCh}}, sk^A_{\mathsf{FCh}})$, $(pk^A_{\mathsf{SleepyCh}}, sk^A_{\mathsf{SleepyCh}})$, and $(pk^A_{\mathsf{Exit}}, sk^A_{\mathsf{Exit}})$. Let $pkey^A_{set}$ denote the set of public keys corresponding to these key pairs.
2. Extract $v_{A,0}$ and $v_{B,0}$ from $\chi.st$, and $c := \chi.c$.
3. Send $(\texttt{createInfo}, \mathsf{id}, pk_A, pkey^A_{set}) \xhookrightarrow{\tau_0} B$.
4. If $(\texttt{createInfo}, \mathsf{id}, pk_B, pkey^B_{set}) \xleftarrow{\tau_0+1} B$, continue. Otherwise, remain idle.
5. Using $pkey^A_{set}$ and $pkey^B_{set}$, A together with B run $\mathcal{F}_{\mathsf{AKG}}$ to generate the following set of shared addresses: $addr_{set} := \{\mathsf{Ch}_{AB}, \mathsf{FCh}_A, \mathsf{FCh}_B, \mathsf{SleepyCh}_A, \mathsf{SleepyCh}_B, \mathsf{Exit}_A, \mathsf{Exit}_B\}$ which takes τ_g rounds. If an error occurs, abort.
6. Generate $tx^{A,B}_F := (tid_A, tid_B)$ with $tid_A := tx(pk_A, \mathsf{Ch}_{AB}, v_{A,0} + c, n_A), tid_B := tx(pk_B, \mathsf{Ch}_{AB}, v_{B,0} + c, n_B)$.
7. Let $tx_{set,0} \leftarrow \mathsf{GTxs}(addr_{set}, pkey^A_{set}, pkey^B_{set}, c, v_{A,0}, v_{B,0})$.
8. Let $sig^A_{set,0} \leftarrow \mathsf{SignTxs}^A(tx_{set,0}, addr_{set}, pkey^A_{set} \cup pkey^B_{set})$.
9. A signs the output tid_A to obtain the signature σ_{tid_A} and sends $(\texttt{createFund}, \mathsf{id}, \sigma_{tid_A}) \xrightarrow{\tau_0+1+\tau_g+\tau_s} A$.
10. If $(\texttt{createFund}, \mathsf{id}, \sigma_{tid_B}) \xleftarrow{\tau_0+2+\tau_g+\tau_s} B$, post $(tx^{A,B}_F, \{\sigma_{tid_A}, \sigma_{tid_B}\})$ to $\mathbb{L}$.
11. If $tx^{A,B}_F$ is accepted by $\mathbb{L}$ in round $\tau_1 \leq \tau_0 + 2 + \tau_g + \tau_s + \Delta$, store $\Psi^A(\mathsf{id}) := (tx^{A,B}_F, tx_{set,0}, sig^A_{set,0}, addr_{set}, pkey^A_{set}, pkey^B_{set})$ and $(\texttt{CREATED}, \mathsf{id}) \xhookrightarrow{\tau_1} \mathcal{E}$.

Update

Party A receives $(\texttt{UPDATE}, \mathsf{id}, \overrightarrow{\theta}, \tau_{stp}) \xleftarrow{\tau_0} \mathcal{E}$

1. $(\texttt{updateReq}, \mathsf{id}, \overrightarrow{\theta}, \tau_{stp}) \xhookrightarrow{\tau_0} B$.

Party B receives $(\texttt{updateReq}, \mathsf{id}, \overrightarrow{\theta}, \tau_{stp}) \xleftarrow{\tau_0} A$

1. Retrieve $(tx^{A,B}_F, tx_{set,i-1}, sig^B_{set,i-1}, addr_{set}, pkey^A_{set}, pkey^B_{set}) = \Psi^B(\mathsf{id})$.
2. Extract $v_{A,i}$ and $v_{B,i}$ from $\overrightarrow{\theta}$, and c from $tx^{A,B}_F$.
3. Let $tx_{set,i} \leftarrow \mathsf{GTxs}(addr_{set}, pkey^A_{set}, pkey^B_{set}, c, v_{A,i}, v_{B,i})$.
4. Let $\overrightarrow{tid} := (tx^A_{Pay,i}.\mathsf{id}, tx^B_{Pay,i}.\mathsf{id}, tx^A_{RPay,i}.\mathsf{id}, tx^B_{RPay,i}.\mathsf{id})$ be a tuple of identities of transactions $tx^A_{Pay,i}$, $tx^B_{Pay,i}$, $tx^A_{RPay,i}$, and $tx^B_{RPay,i}$.
5. Send $(\texttt{UPDATE-REQ}, \mathsf{id}, \overrightarrow{\theta}, \tau_{stp}, \overrightarrow{tid}) \xhookrightarrow{\tau_0} \mathcal{E}$.
6. Send $(\texttt{updateInfo}, \mathsf{id}) \xhookrightarrow{\tau_0} A$.

Party A receives $(\texttt{updateInfo}, \mathsf{id}) \xleftarrow{\tau_0+2} B$

1. Retrieve $(tx^{A,B}_F, tx_{set,i-1}, sig^A_{set,i-1}, addr_{set}, pkey^A_{set}, pkey^B_{set}) = \Psi^A(\mathsf{id})$.
2. Extract $v_{A,i}$ and $v_{B,i}$ from $\overrightarrow{\theta}$, and c from $tx^{A,B}_F$.
3. Let $tx_{set,i} \leftarrow \mathsf{GTxs}(addr_{set}, pkey^A_{set}, pkey^B_{set}, c, v_{A,i}, v_{B,i})$.
4. Let $\overrightarrow{tid} := (tx^A_{Pay,i}.\mathsf{id}, tx^B_{Pay,i}.\mathsf{id}, tx^A_{RPay,i}.\mathsf{id}, tx^B_{RPay,i}.\mathsf{id})$ be a tuple of identities of transactions $tx^A_{Pay,i}$, $tx^B_{Pay,i}$, $tx^A_{RPay,i}$, and $tx^B_{RPay,i}$.
5. $(\texttt{SETUP}, \mathsf{id}, \overrightarrow{tid}) \xhookrightarrow{\tau_0+2} \mathcal{E}$.
6. If $(\texttt{SETUP-OK}, \mathsf{id}) \xleftarrow{\tau_1 \leq \tau_0+2+\tau_{stp}} \mathcal{E}$, send $(\texttt{updateCom}, \mathsf{id}) \xhookrightarrow{\tau_1} B$.
7. Wait for one round.
8. $\mathsf{SignTxs}^A(tx_{set,i}, addr_{set}, pkey^A_{set} \cup pkey^B_{set})$.

Party B receives $(\texttt{updateCom}, \mathsf{id}) \xleftarrow{\tau_1 \leq \tau_0+2+\tau_{stp}} A$

1. $(\texttt{SETUP-OK}, \mathsf{id}) \xhookrightarrow{\tau_1} \mathcal{E}$.
2. If not $(\texttt{UPDATE-OK}, \mathsf{id}) \xhookrightarrow{\tau_1} \mathcal{E}$, remain idle.
3. $\mathsf{SignTxs}^A(tx_{set,i}, addr_{set}, pkey^A_{set} \cup pkey^B_{set})$.

Party A in round $\tau_1 + 1 + \tau_s$

1. If $sig^A_{set,i}$ is output by $\mathsf{SignTxs}^A$, send $(\texttt{UPDATE-OK}, \mathsf{id}) \xrightarrow{\tau_1+1+\tau_s} \mathcal{E}$. Otherwise, invoke $\texttt{ForceClose}(\mathsf{id})$ and remain idle.
2. If not $(\texttt{REVOKE}, \mathsf{id}) \xleftarrow{\tau_1+1+\tau_s} \mathcal{E}$, remain idle.
3. A and B jointly execute the interactive protocol $\mathcal{F}_{\mathsf{Sign}}$ to sign the punishing transaction $tx^B_{Pnsh,i-1}$ and obtain the signature $\sigma^B_{Pnsh,i-1}$. The protocol requires τ_r rounds, and A receives the output $\sigma^B_{Pnsh,i-1}$ after its completion. If an error occurs, invoke $\texttt{ForceClose}(\mathsf{id})$.
4. $(\texttt{REVOKE}, \mathsf{id}, \sigma^B_{Pnsh,i-1}) \xrightarrow{\tau_1+1+\tau_s+\tau_r} B$.

Party B in round $\tau_1 + \tau_s$

1. If $sig^B_{set,i}$ is not output by $\mathsf{SignTxs}^A$, invoke $\texttt{ForceClose}(\mathsf{id})$ and remain idle.
2. Participate in the signing of $tx^B_{Pnsh,i-1}$.

3. Upon $(\texttt{REVOKE}, \mathsf{id}, \sigma^B_{Pnsh,i-1}) \xleftarrow{\tau_1+1+\tau_s+\tau_r} A$, continue. Otherwise, invoke $\texttt{ForceClose}(\mathsf{id})$ and remain idle.
4. Send $(\texttt{REVOKE-REQ}, \mathsf{id}) \xhookrightarrow{\tau_1+1+\tau_s+\tau_r} \mathcal{E}$.
5. If not $(\texttt{REVOKE}, \mathsf{id}) \xhookleftarrow{\tau_1+1+\tau_s+\tau_r} \mathcal{E}$, remain idle.
6. B and A jointly execute the interactive protocol $\mathcal{F}_{\mathsf{Sign}}$ to generate the signature $\sigma^A_{Pnsh,i-1}$ for the punishment transaction $tx^A_{Pnsh,i-1}$. The protocol takes τ_r rounds, and B receives the output $\sigma^A_{Pnsh,i-1}$ after its completion. If an error occurs, invoke $\texttt{ForceClose}(\mathsf{id})$.
7. Send $(\texttt{REVOKE}, \mathsf{id}, \sigma^A_{Pnsh,i-1}) \xhookrightarrow{\tau_1+1+\tau_s+2\tau_r} A$.
8. $\Theta^B(id) := \Theta^B \cup \{(tx_{set,i-1}, sig^B_{set,i-1}, \sigma^A_{Pnsh,i-1})\}$.
9. $\varPsi^B(id) := (tx^{A,B}_F, tx_{set,i}, sig^B_{set,i}, addr_{set}, pkey^A_{set}, pkey^B_{set})$.
10. Send $(\texttt{UPDATED}, \mathsf{id}) \xhookrightarrow{\tau_1+2+\tau_s+2\tau_r} \mathcal{E}$.

Party A in round $\tau_1 + 2 + \tau_s + \tau_r$

1. Participate in the signature generation of $tx^A_{Pnsh,i-1}$.
2. If $(\texttt{REVOKE}, \mathsf{id}, \sigma^A_{Pnsh,i-1}) \xhookleftarrow{\tau_1+3+\tau_s+2\tau_r} B$ and $\sigma^A_{Pnsh,i-1}$ is valid, proceed to the next step. Else, invoke $\texttt{ForceClose}(\mathsf{id})$.
3. $\Theta^A(id) := \Theta^A \cup \{(tx_{set,i-1}, sig^A_{set,i-1}, \sigma^B_{Pnsh,i-1})\}$.
4. $\varPsi^A(id) := (tx^{A,B}_F, tx_{set,i}, sig^A_{set,i}, addr_{set}, pkey^A_{set}, pkey^B_{set})$.
5. Send $(\texttt{UPDATED}, \mathsf{id}) \xhookrightarrow{\tau_1+3+\tau_s+2\tau_r} \mathcal{E}$.

Close

Party A receives $(\texttt{CLOSE}, \mathsf{id}) \xhookleftarrow{\tau_0} \mathcal{E}$

1. Extract $(tx^{A,B}_F, tx_{set,i}, sig^A_{set,i}, addr_{set}, pkey^A_{set}, pkey^B_{set})$ from $\varPsi^A(\mathsf{id})$.
2. Extract $v_{A,i}$ and $v_{B,i}$ from $(tx^A_{Pay,i}, tx^B_{RPay,i}) \in tx_{set,i}$, and c from $tx^{A,B}_F$.
3. Create transaction $tx_c := tx(tid'_A, tid'_B)$, where $tid'_A = tx(\mathsf{Ch}_{AB}, pk_A, v_{A,i} + c, b), tid'_B = tx(\mathsf{Ch}_{AB}, pk_B, v_{B,i} + c, 1-b), b \in \{0,1\}$ is the nonce of Ch_{AB}, pk_A is an address controlled by A and pk_B is an address controlled by B.
4. A and B jointly execute the interactive protocol $\mathcal{F}_{\mathsf{Sign}}$ to generate the signature σ_{tx_c} for the transaction tx_c. The protocol involves τ_r rounds of interaction between the parties.
5. If the signing process was successful, then publish (tx_c, σ_{tx_c}) on $\mathbb{L}$. Else, invoke $\texttt{ForceClose}(\mathsf{id})$.
6. If tx_c is chained on $\mathbb{L}$ in round $\tau_1 \leq \tau_0 + \tau_r + \Delta$, let $\Theta^A(id) := \perp$, $\varPsi^A(\mathsf{id}) := \perp$ and send $(\texttt{CLOSED}, \mathsf{id}) \xhookrightarrow{\tau_2} \mathcal{E}$.

Punish

Party A receives $\texttt{Punish} \xhookleftarrow{\tau_0} \mathcal{E}$

$\forall\ \mathsf{id} \in \{0,1\}^*$ such that $\Theta^P(\mathsf{id}) \neq \perp$

1. Iterate over all elements $(tx_{set,i}, sig^A_{set,i}, \sigma^B_{Pnsh,i})$ in $\Theta^P(\mathsf{id})$.
2. If the revoked payment $(tx^B_{Pay,i}, tx^{B*}_{Pay,i}) \in tx_{set,i}$ is on $\mathbb{L}$, post $(tx^A_{Pnsh,i}, \sigma^A_{Pnsh,i})$ on $\mathbb{L}$ before the absolute timeout $\mathbf{T}_d$.
3. Let $tx^B_{Pnsh,i}$ be accepted by $\mathbb{L}$ in round $\tau_1 \leq \tau_0 + \Delta$. Post $(tx^A_{RPay,i}, tx^{A*}_{RPay,i}, (\sigma^A_{RPay,i}, \sigma^{A*}_{RPay,i}) \in sig^A_{set,i})$.
4. After $tx^A_{RPay,i}$ and $tx^{A*}_{RPay,i}$ are accepted by $\mathbb{L}$ in round $\tau_2 \leq \tau_1 + \Delta$, set $\Theta^A(\mathsf{id}) := \perp, \varPsi^A(\mathsf{id}) := \perp$ and output $(\texttt{PUNISHED}, \mathsf{id}) \xhookrightarrow{\tau_1} \mathcal{E}$.

Subroutines

$\mathsf{ForceClose}(\mathsf{id})$:

Let τ_0 be the current round.

1. Extract $(tx^{A,B}_F, tx_{set,0}, sig^A_{set,0}, addr_{set}, pkey^A_{set}, pkey^B_{set})$ from $\varPsi^A(\mathsf{id})$ and extract $(tx^A_{Pay,j}, tx^{A*}_{Pay,j})$ from $tx_{set,0}$ and $(\sigma^A_{Pay,j}, \sigma^{A*}_{Pay,j})$ from $sig^A_{set,0}$.
2. Party A posts $(tx^A_{Pay,j}, \sigma^A_{Pay,j})$ and $(tx^{A*}_{Pay,j}, \sigma^{A*}_{Pay,j})$ on $\mathbb{L}$.
3. Let $\tau_1 \leq \tau_0 + \Delta$ be the round in which $tx^A_{Pay,j}$ and $tx^{A*}_{Pay,j}$ are accepted by $\mathbb{L}$.
4. If $tx^B_{RPay,j}$ and $tx^{B*}_{RPay,j}$ appear on $\mathbb{L}$ at or after round $\tau_2 \leq \tau_1 + \Delta$ and before $\mathbf{T}_d$, post $(tx^A_{FPay,j}, \sigma^A_{FPay,j})$ and send $(\texttt{CLOSED}, id) \xhookrightarrow{\tau_3 \leq \tau_2+\Delta} \mathcal{E}$. Otherwise, post $(tx^A_{D,j}, \sigma^A_{D,j})$ after $\mathbf{T}_d$ and send $(\texttt{CLOSED}, \mathsf{id}) \xhookrightarrow{\tau_4 \leq \mathbf{T}_d+\Delta} \mathcal{E}$.
5. Set $\varPsi^P(\mathsf{id}) := \perp$, $\Theta^P(\mathsf{id}) := \perp$.

$\mathsf{GTxs}(addr_{set}, pkey^A_{set}, pkey^B_{set}, c, v_{A,i}, v_{B,i})$:

1. On the basis of $addr_{set}$, $pkey^A_{set}$ and $pkey^B_{set}$, perform the following steps.
2. Assemble $tx^A_{Pay,i} := tx(\mathsf{Ch}_{AB}, \mathsf{FCh}_A, v_{A,i} + c, 0)$, $tx^B_{Pay,i} := tx(\mathsf{Ch}_{AB}, \mathsf{FCh}_A, v_{B,i} + c, 0)$, $tx^A_{RPay,i} = tx(\mathsf{Ch}_{AB}, \mathsf{Exit}_B, v_{A,i} + c, 1)$, and $tx^B_{RPay,i} = tx(\mathsf{Ch}_{AB}, \mathsf{Exit}_A, v_{B,i} + c, 1)$.
3. Assemble fork-payment transactions $tx^{A*}_{Pay,i} := (\mathsf{FCh}_A, \mathsf{SleepyCh}_A, v_{A,i}, 0)$, $tx^A_R := (\mathsf{FCh}_A, pk_A, c, 1)$, $tx^{B*}_{Pay,i} := (\mathsf{FCh}_B, \mathsf{SleepyCh}_B, v_{B,i}, 0)$, $tx^B_R := (\mathsf{FCh}_B, pk_B, c, 1)$.
4. Assemble lazy finish-payment transactions $tx^A_{D,i} := (\mathsf{SleepyCh}_A, pk_A, v_{A,i}, 0)$ and

$tx^B_{D,i} := (\mathsf{SleepyCh}_B, pk_B, v_{B,i}, 0)$ both are restricted to be spent until time $\mathbf{T}_d$.

5. Assemble fast finish-payment transactions $tx^A_{FPay,i} := (\mathsf{SleepyCh}_A, pk_A, v_{A,i}, 0)$ and $tx^B_{FPay,i} := (\mathsf{SleepyCh}_B, pk_B, v_{B,i}, 0)$.
6. Assemble a set of exiting transactions $tx^{A*}_{RPay,i} := (\mathsf{Exit}_B, pk_A, v_{A,i} + c, 0)$ and $tx^{B*}_{RPay,i} := (\mathsf{Exit}_A, pk_B, v_{B,i} + c, 0)$ for the above fast finish-payment.
7. Return $tx_{set} := \{tx^A_{Pay,i}, tx^B_{Pay,i}, tx^A_{RPay,i}, tx^B_{RPay,i}, tx^{A*}_{Pay,i}, tx^{B*}_{Pay,i}, tx^A_R, tx^B_R, tx^A_{Pnsh,i}, tx^B_{Pnsh,i}, tx^A_{D,i}, tx^B_{D,i}, tx^A_{FPay,i}, tx^B_{FPay,i}, tx^{A*}_{RPay,i}, tx^{B*}_{RPay,i}\}$.

$\underline{\mathsf{SignTxs}^A(tx_{set}, addr_{set}, pkey^A_{set} \cup pkey^B_{set})}$:

The party that receives the signatures first is A, denoted by the superscript of the function. Once A and B agree to execute this subroutine in the one round with identical parameters, and the following steps are performed. Upon the extracted transactions, addresses, and public keys from the parameters, party A and B jointly execute $\mathcal{F}_{Sign}$ and $\mathcal{F}_{pSign}$ to generate signatures and pre-signatures of the transactions, as described below.

1. Party A obtains pre-signature $\widehat{\sigma}^A_{Pay,i}(Y_{A,1,i})$ and witness $y_{A,1,i}$ on transaction $tx^A_{Pay,i}$, pre-signature $\widehat{\sigma}^B_{Pay,i}(Y_{A,2,i})$ on transaction $tx^B_{Pay,i}$, and pre-signature $\widehat{\sigma}^A_{RPay,i}(Y_{A,2,i})$ on transaction $tx^A_{RPay,i}$ under Ch_{AB}.
2. Party B obtains pre-signature $\widehat{\sigma}^B_{Pay,i}(Y_{A,2,i})$ and witness $y_{A,2,i}$ on transaction $tx^B_{Pay,i}$, pre-signature $\widehat{\sigma}^A_{Pay,i}(Y_{A,1,i})$ on transaction $tx^A_{Pay,i}$, and pre-signature $\widehat{\sigma}^B_{RPay,i}(Y_{A,1,i})$ on transaction $tx^B_{RPay,i}$ under Ch_{AB}.
3. Party A obtains signature $\sigma^{A*}_{Pay,i}$ on transaction $tx^{A*}_{Pay,i}$, and signature $\sigma^A_{R,i}$ on transaction $tx^A_{R,i}$ with regard to FCh_A.
4. Party B obtains signature $\sigma^{B*}_{Pay,i}$ on transaction $tx^{B*}_{Pay,i}$, and signature $\sigma^B_{R,i}$ on transaction $tx^B_{R,i}$ with regard to FCh_B.
5. Party A obtains signature $\sigma^A_{D,i}$ on transaction $tx^A_{D,i}$ under $\mathsf{SleepyCh}_A$.
6. Party B obtains signature $\sigma^B_{D,i}$ on transaction $tx^B_{D,i}$ under $\mathsf{SleepyCh}_B$.
7. Party A obtains pre-signature $\widehat{\sigma}^A_{FPay,i}(Y_{B,1,i})$ on transaction $tx^A_{FPay,i}$ under $\mathsf{SleepyCh}_A$, and pre-signature $\widehat{\sigma}^{B*}_{RPay,i}(Y_{B,1,i})$ on transaction $tx^{B*}_{RPay,i}$ under Exit_A. Party B obtains pre-signature $\widehat{\sigma}^{B*}_{RPay,i}(Y_{B,1,i})$ and witness $y_{B,1,i}$ on transaction $tx^{B*}_{RPay,i}$ under Exit_A.
8. Party B obtains pre-signature $\widehat{\sigma}^B_{FPay,i}(Y_{B,2,i})$ on transaction $tx^B_{FPay,i}$ under the shared address $\mathsf{SleepyCh}_B$, and pre-signature $\widehat{\sigma}^{A*}_{RPay,i}(Y_{B,2,i})$ on transaction $tx^{A*}_{RPay,i}$ under Exit_B. Party A obtains pre-signature $\widehat{\sigma}^{A*}_{RPay,i}(Y_{B,2,i})$ and witness $y_{B,2,i}$ on transaction $tx^{B*}_{RPay,i}$ under Exit_B.

The above operation lasts for τ_s rounds, and if signatures and pre-signatures are not obtained or are not valid for the specific transactions, proceed to the steps in CLOSE. If the subroutine is executed successfully, return to A $sig^A_{set,i} := \{(\widehat{\sigma}^A_{Pay,i}, \widehat{\sigma}^A_{RPay,i}), (\sigma^{A*}_{Pay,i}, \sigma^A_{R,i}, \sigma^A_{D,i}), (\widehat{\sigma}^A_{FPay,i}, \widehat{\sigma}^{B*}_{RPay,i}, \widehat{\sigma}^{A*}_{RPay,i})\}$ and $(y_{A,1,i}, y_{B,2,i})$, and to B $sig^B_{set,i} := \{(\widehat{\sigma}^B_{Pay,i}, \widehat{\sigma}^B_{RPay,i}), (\sigma^{B*}_{Pay,i}, \sigma^B_{R,i}, \sigma^B_{D,i}), (\widehat{\sigma}^B_{FPay,i}, \widehat{\sigma}^{A*}_{RPay,i}, \widehat{\sigma}^{B*}_{RPay,i})\}$ and $(y_{A,2,i}, y_{B,1,i})$.

Indistinguishability: We must demonstrate that, without any trusted third party, the real-world protocol described in Section 5 and its UC version are indistinguishable from the view of any polynomial-time adversary. To achieve this, we replace the 2-party cryptographic protocols in Π_{AS}, namely aggregated key generation Γ_{AKG}, signing Γ_{Sign}, and pre-signing Γ_{pSign} with idealized versions $\mathcal{F}_{\mathsf{AKG}}$, $\mathcal{F}_{\mathsf{Sign}}$, and $\mathcal{F}_{\mathsf{pSign}}$, respectively. The UC formulations can be referred to [49].

We first define Π''_{B} as the protocol Π'''_{B} presented in Section 5, but Π''_{B} replaces Γ_{AKG} with an ideal functionality $\mathcal{F}_{\mathsf{AKG}}$ for 2-party aggregated key generation.

To show that Π'''_{B} is indistinguishable from Π''_{B}, an adversary $\mathcal{A}$ is assumed to be able to distinguish between the two protocols. Then, a reduction algorithm $\mathcal{R}$ can be constructed to use $\mathcal{A}$ as a subroutine. The only difference between the two protocols is the use of Γ_{AKG} in Π''_{B} and $\mathcal{F}_{\mathsf{AKG}}$ in $\Pi B'''$, and thus, the output of $\mathcal{R}$ when using $\mathcal{A}$ can distinguish between a key share from Γ_{AKG} and one from $\mathcal{F}_{\mathsf{AKG}}$. This implies that the security of our 2-party aggregated key generation can be broken with non-trivial probability.

Next, we define Π'_{B} as Π''_{B} with the difference being that Γ_{Sign} is replaced with an ideal functionality $\mathcal{F}_{\mathsf{Sign}}$, which generates signatures locally and simulates the behavior of corrupted parties during the signing protocol. This is equivalent to the UC version of Π''_{B}.

To show that Π''_{B} is indistinguishable from Π'_{B}, an adversary $\mathcal{A}$ is assumed to be able to distinguish between the two protocols. The only difference between the two protocols is the use of Γ_{Sign} in Π''_{B} and $\mathcal{F}_{\mathsf{Sign}}$ in Π'_{B}. Therefore, if $\mathcal{A}$ can distinguish between a real interaction and a simulated one with non-trivial probability, it would imply a contradiction against the UC-secure Γ_{Sign}.

Finally, we define Π_{B} as Π'_{B} with the difference being that the 2-party pre-signing protocol Γ_{pSign} for Π_{AS} is replaced with an ideal functionality $\mathcal{F}_{\mathsf{pSign}}$, which generates pre-signatures locally and simulates the behavior of corrupted parties during the pre-signing protocol. This is also equivalent to the UC version of Π'_{B}.

To show that Π'_{B} is indistinguishable from Π_{B}, an adversary $\mathcal{A}$ is assumed to be able to distinguish between the two protocols. The only difference between the two protocols is the use of Γ_{Sign} in Π'_{B} and $\mathcal{F}_{\mathsf{pSign}}$ in Π_{B}. Therefore, if $\mathcal{A}$ can distinguish between a real interaction and a simulated one with non-trivial probability, it would imply a contradiction against the UC-secure Γ_{pSign}.

C.1. UC Simulator

We now present the pseudocode of a simulator for the ideal-world model of Π_{B} in Appendix C. The simulator runs in the ideal world and interacts with the ideal functionality $\mathcal{F}_{AM}$ and the ledger $\mathbb{L}$. During the simulation, the simulator also uses the subroutine $\mathsf{SignTxs}^P$ as described in the formal protocol. In the UC-simulation proof, the main challenge is to provide a simulated transcript indistinguishable from the real-world protocol-executing transcript, without access to the parties' secret inputs from the environment.

There are no secret inputs in the simulation of the protocol, since our model forwards all messages to the simulator. Consequently, we do not need to account for the case where both parties are honest, as the simulator can easily simulate the protocol with the knowledge of all messages forwarded to the functionality. The primary challenge in is handling the malicious parties' behavior.

Simulator for Create

Case of honest A and corrupted B

When A sends $(\texttt{CREATE}, \chi, pk_A) \xhookrightarrow{\tau_0} \mathcal{F}_{AM}$, but B does not send $(\texttt{CREATE}, \chi, pk_B) \xhookrightarrow{\tau} \mathcal{F}_{AM}$ where $|\tau_0 - \tau| \leq \tau_p$, then there are the following two possible cases.

1. If B sends $(\texttt{createInfo}, \mathsf{id}, pk_B, pkey^B_{set}) \xhookrightarrow{\tau_0} A$, then on behalf of B, send $(\texttt{CREATE}, \chi, pk_B) \xhookrightarrow{\tau_0} \mathcal{F}_{AM}$.
2. Otherwise halt.

Follow the steps as described below:

1. Let $\mathsf{id} := \chi.\mathsf{id}$, create $(pk^A_{\mathsf{FCh}}, sk^A_{\mathsf{FCh}})$, $(pk^A_{\mathsf{SleepyCh}}, sk^A_{\mathsf{SleepyCh}})$, and $(pk^A_{\mathsf{Exit}}, sk^A_{\mathsf{Exit}})$. Denote $pkey^A_{set}$ as the set of public keys. Send $(\texttt{createInfo}, \mathsf{id}, pk_A, pkey^A_{set}) \xhookrightarrow{\tau_0} B$.
2. If $(\texttt{createInfo}, \mathsf{id}, pk_B, pkey^B_{set}) \xhookleftarrow{\tau_0+1} B$, take the following actions. Otherwise remain idle.
3. Upon $pkey^A_{set}$ and $pkey^B_{set}$, the simulator, acting as A, executes $\mathcal{F}_{\mathsf{AKG}}$ with B to obtain a list of shared addresses as follows, $addr_{set} := \{\mathsf{Ch}_{AB}$, FCh_A,FCh_B, $\mathsf{SleepyCh}_A$, $\mathsf{SleepyCh}_B$, Exit_A, $\mathsf{Exit}_B\}$ which requires τ_g rounds. Abort if a failure occurs.
4. Generate $tx^{A,B}_F := (tid_A, tid_B)$ with $tid_A := tx(pk_A, \mathsf{Ch}_{AB}, v_{A,0} + c, n_A), tid_B := tx(pk_B, \mathsf{Ch}_{AB}, v_{B,0} + c, n_B)$.
5. Let $tx_{set,0} \leftarrow \mathsf{GTxs}(addr_{set}, pkey^A_{set}, pkey^B_{set}, c, v_{A,0}, v_{B,0})$.
6. Let $sig^A_{set,0} \leftarrow \mathsf{SignTxs}^A(tx_{set,0}, addr_{set}, pkey^A_{set} \cup pkey^B_{set})$.
7. Acting as A to sign the output tid_A and obtain the signature σ_{tid_A}, and send $(\texttt{createFund}, \mathsf{id}, \sigma_{tid_A}) \xhookrightarrow{\tau_0+1+\tau_g+\tau_s} A$.
8. If you receive $(\texttt{createFund}, \mathsf{id}, \sigma_{tid_B}) \xhookleftarrow{\tau_0+2+\tau_g+\tau_s} B$, post $(tx^{A,B}_F, \{\sigma_{tid_A}, \sigma_{tid_B}\})$ to $\mathbb{L}$.
9. If $tx^{A,B}_F$ is chained on $\mathbb{L}$ in round $\tau_1 \leq \tau_0 + 2 + \tau_g + \tau_s + \Delta$, store $\Psi^A(\mathsf{id}) := (tx^{A,B}_F$, $tx_{set,0}$, $sig^A_{set,0}$, $addr_{set}$, $pkey^A_{set}$, $pkey^B_{set})$ and send $(\texttt{CREATED}, \mathsf{id}) \xhookrightarrow{\tau_1} \mathcal{E}$.

Simulator for Update

Case of honest A and corrupted B

When A sends $(\texttt{UPDATE}, \mathsf{id}, \overrightarrow{\theta}, \tau_{stp}) \xhookrightarrow{\tau_0} \mathcal{F}_{AM}$, perform the following steps:

1. $(\texttt{updateReq}, \mathsf{id}, \overrightarrow{\theta}, \tau_{stp}) \xhookrightarrow{\tau_0} B$.
2. Receive $(\texttt{updateInfo}, \mathsf{id}) \xhookleftarrow{\tau_0+2} B$, take the actions as follows:
3. Retrieve $(tx^{A,B}_F, tx_{set,i-1}, sig^B_{set,i-1}, addr_{set}, pkey^A_{set}, pkey^B_{set}) := \Psi^B(\mathsf{id})$.
4. Retrieve $v_{A,i}$ and $v_{B,i}$ from $\overrightarrow{\theta}$, and c from $tx^{A,B}_F$.
5. Let $tx_{set,i} \leftarrow \mathsf{GTxs}(addr_{set}, pkey^A_{set}, pkey^B_{set}, c, v_{A,i}, v_{B,i})$.
6. Let $\overrightarrow{tid} := (tx^A_{Pay,i}.\mathsf{id}, tx^B_{Pay,i}.\mathsf{id}, tx^A_{RPay,i}.\mathsf{id}, tx^B_{RPay,i}.\mathsf{id})$ be a tuple of identities of transactions $tx^A_{Pay,i}$, $tx^B_{Pay,i}$, $tx^A_{RPay,i}$, and $tx^B_{RPay,i}$. Notify $\mathcal{F}_{AM}$ of $\overrightarrow{tid}$ in round $\tau_0 + 2$.
7. If A sends $(\texttt{SETUP-OK}, \mathsf{id}) \xhookrightarrow{\tau_1 \leq \tau_0+2+\tau_{stp}} \mathcal{F}_{AM}$, send $(\texttt{updateCom}, \mathsf{id}) \xhookrightarrow{\tau_1} B$.
8. Wait for one round.
9. If the current round is $\tau_1 + 1$, B starts performing $\mathsf{SignTxs}^A(tx_{set,i}, addr_{set}, pkey^A_{set} \cup pkey^B_{set})$, send $(\texttt{UPDATE-OK}, \mathsf{id}) \xhookrightarrow{\tau_1+1} \mathcal{F}_{AM}$ via acting as B.
10. $\mathsf{SignTxs}^A(tx_{set,i}, addr_{set}, pkey^A_{set} \cup pkey^B_{set})$.
11. If $sig^A_{set,i}$ is returned from $\mathsf{SignTxs}^A$, instruct $\mathcal{F}_{AM}$ to $(\texttt{UPDATE-OK}, \mathsf{id}) \xhookrightarrow{\tau_1+1+\tau_s} \mathcal{E}$ via A. Otherwise, invoke $\mathsf{ForceClose}^A(\mathsf{id})$ and remain idle.
12. If A fails to send $(\texttt{REVOKE}, \mathsf{id}) \xhookrightarrow{\tau_1+1+\tau_s} \mathcal{F}_{AM}$, remain idle.
13. The simulator, acting as A, executes the interactive protocol $\mathcal{F}_{\mathsf{Sign}}$ with B to create the following signature, $\sigma^B_{Pnsh,i-1}$ on the punishing transaction $tx^B_{Pnsh,i-1}$. Party A obtains $tx^B_{Pnsh,i-1}$ as output. This requires τ_r rounds. If a failure occurs, invoke $\mathsf{ForceClose}^A(\mathsf{id})$.
14. Send $(\texttt{REVOKE}, \mathsf{id}, \sigma^B_{Pnsh,i-1}) \xhookrightarrow{\tau_1+1+\tau_s+\tau_r} B$.

15. If B executes $\mathcal{F}_{\mathsf{Sign}}$ to generate a signature of $tx^B_{Pnsh,i-1}$ in round $\tau_1+2+\tau_s+\tau_r$, send (`REVOKE`, id) $\xhookrightarrow{\tau_1+2+\tau_s+\tau_r}$ $\mathcal{F}_{AM}$ by acting as B and also acts as A to participate in the signature generation.
16. If (`REVOKE`, id, $\sigma^B_{Pnsh,i-1}$) $\xhookleftarrow{\tau_1+3+\tau_s+2\tau_r}$ B and $\sigma^B_{Pnsh,i-1}$ is valid, proceed to the next step. Else, invoke $\mathsf{ForceClose}^A(\mathsf{id})$.
17. $\Theta^A(id) := \Theta^A \cup \{(tx_{set,i-1}, sig^A_{set,i-1}, \sigma^B_{Pnsh,i-1})\}$.
18. $\Psi^A(id) := (tx^{A,B}_F, tx_{set,i}, sig^A_{set,i}, addr_{set}, pkey^A_{set}, pkey^B_{set})$.

<u>Case of honest B and corrupted A</u>

When A sends (`updateReq`, id, $\overrightarrow{\theta}, \tau_{stp}$) $\xhookrightarrow{\tau_0}$ B, send (`UPDATE`, id, $\overrightarrow{\theta}, \tau_{stp}$) $\xhookrightarrow{\tau_0}$ $\mathcal{F}_{AM}$ by acting as A, if A has not yet transmitted this message. Proceed as follows:

1. Receive (`updateReq`, id, $\overrightarrow{\theta}, \tau_{stp}$) $\xhookleftarrow{\tau_0}$ A, take the following actions:
2. Retrieve $(tx^{A,B}_F, tx_{set,i-1}, sig^B_{set,i-1}, addr_{set}, pkey^A_{set}, pkey^B_{set}) := \Psi^B(\mathsf{id})$.
3. Retrieve $v_{A,i}$ and $v_{B,i}$ from $\overrightarrow{\theta}$, and c from $tx^{A,B}_F$.
4. Let $tx_{set,i} \leftarrow \mathsf{GTxs}(addr_{set}, pkey^A_{set}, pkey^B_{set}, c, v_{A,i}, v_{B,i})$.
5. Let $\overrightarrow{tid} := (tx^A_{Pay,i}.\mathsf{id}, tx^B_{Pay,i}.\mathsf{id}, tx^A_{RPay,i}.\mathsf{id}, tx^B_{RPay,i}.\mathsf{id})$ be a tuple of identities of transactions $tx^A_{Pay,i}$, $tx^B_{Pay,i}$, $tx^A_{RPay,i}$, and $tx^B_{RPay,i}$. Notify $\mathcal{F}_{AM}$ of $\overrightarrow{tid}$.
6. Send (`updateInfo`, id) $\xhookrightarrow{\tau_0}$ A.
7. When A sends (`updateCom`, id) $\xhookrightarrow{\tau_0+1+\tau_{stp}}$ B, send (`SETUP-OK`, id) $\xhookrightarrow{\tau_1}$ $\mathcal{F}_{AM}$ by acting as A.
8. (`updateCom`, id) $\xhookleftarrow{\tau_1\le\tau_0+2+\tau_{stp}}$ A.
9. If B transmits (`UPDATE-OK`, id) $\xhookrightarrow{\tau_1}$ $\mathcal{F}_{AM}$, invoke $\mathsf{SignTxs}^A(tx_{set,i}, addr_{set}, pkey^A_{set} \cup pkey^B_{set})$.
10. If $sig^B_{set,i}$ is not returned from $\mathsf{SignTxs}^A$ in round $\tau_1+\tau_s$, execute $\mathsf{ForceClose}^B(\mathsf{id})$ and remain idle.
11. If A executes $\mathcal{F}_{\mathsf{Sign}}$ to generate $\sigma^B_{Pnsh,i-1}$ in round $\tau_1+\tau_s$, transmit (`REVOKE`, id) $\xhookleftarrow{\tau_1+\tau_s}$ $\mathcal{F}_{AM}$ by acting as A. Also, act as B to participate in the signature generation.
12. If receive (`REVOKE`, id, $\sigma^B_{Pnsh,i-1}$) $\xhookleftarrow{\tau_1+1+\tau_s+\tau_r}$ A, continue. Else, invoke $\mathsf{ForceClose}^B(\mathsf{id})$ and remain idle.
13. If B fails to send (`REVOKE`, id) $\xhookrightarrow{\tau_1+1+\tau_s+\tau_r}$ $\mathcal{F}_{AM}$, remain idle.
14. $\mathcal{S}$, acting as B, executes $\mathcal{F}_{\mathsf{Sign}}$ with A to obtain the signature $\sigma^A_{Pnsh,i-1}$ on $tx^A_{Pnsh,i-1}$. Party B finally obtains $\sigma^A_{Pnsh,i-1}$ after τ_r. If an error occurs, invoke $\mathsf{ForceClose}^B(\mathsf{id})$.
15. Send (`REVOKE`, id, $\sigma^A_{Pnsh,i-1}$) $\xhookrightarrow{\tau_1+1+\tau_s+2\tau_r}$ A.
16. $\Theta^B(id) := \Theta^B \cup \{(tx_{set,i-1}, sig^B_{set,i-1}, \sigma^A_{Pnsh,i-1})\}$.
17. $\Psi^B(id) := (tx^{A,B}_F, tx_{set,i}, sig^B_{set,i}, addr_{set}, pkey^A_{set}, pkey^B_{set})$.

Simulator for Close

<u>Case of honest A and corrupted B</u>

When A sends (`CLOSE`, id) $\xhookrightarrow{\tau_0}$ $\mathcal{F}$, take the actions as follows.

1. Retrieve $(tx^{A,B}_F, tx_{set,i}, sig^A_{set,i}, addr_{set}, pkey^A_{set}, pkey^B_{set})$ from $\Psi^A(\mathsf{id})$.
2. Retrieve $v_{A,i}$ and $v_{B,i}$ from $(tx^A_{Pay,i}, tx^B_{RPay,i}) \in tx_{set,i}$, and c from $tx^{A,B}_F$.
3. Create transaction $tx_c := tx(tid'_A, tid'_B)$, where $tid'_A = tx(\mathsf{Ch}_{AB}, pk_A, v_{A,i}+c, b), tid'_B = tx(\mathsf{Ch}_{AB}, pk_B, v_{B,i}+c, 1-b), b \in \{0,1\}$ is the nonce of Ch_{AB}, pk_A is A's address and pk_B is B's address.
4. The simulator, acting as A, executes $\mathcal{F}_{\mathsf{Sign}}$ with B to generate the signature σ_{tx_c} on tx_c. This requires τ_r rounds.
5. If the signature is generated successfully, publish (tx_c, σ_{tx_c}) on $\mathbb{L}$ and transmit (`CLOSED`, id) $\xhookrightarrow{\tau_0+\tau_r}$ $\mathcal{F}_{AM}$ by acting as B. Else, invoke `ForceClose`$^A(\mathsf{id})$.
6. If tx_c is chained on $\mathbb{L}$ in round $\tau_1 \le \tau_0+\tau_r+\Delta$, let $\Theta^A(id) := \perp, \Psi^A(id) := \perp$.

Simulator for Punish

<u>Case of honest A and corrupted B</u>

When A sends `PUNISH` $\xhookrightarrow{\tau_0}$ $\mathcal{F}_{AM}$, $\forall$ $\mathsf{id} \in \{0,1\}^*$ such that $\Theta^A(\mathsf{id}) \ne \perp$, take the actions as follows:

1. Derive $\{(tx_{set,i}, sig^A_{set,i}, \sigma^B_{Pnsh,i})\}_{i\in m} := \Theta^P(\mathsf{id})$ and retrieve χ from $\Psi^A(\mathsf{id})$. If $\exists i \in m$ such that the revoked payment $(tx^B_{Pay,i}, tx^{B*}_{Pay,i}) \in tx_{set,i}$ is on $\mathbb{L}$, take the following actions.
2. Post $(tx^A_{Pnsh,i}, \sigma^A_{Pnsh,i})$ on $\mathbb{L}$ before the timeout $\mathbf{T}_d$.
3. If $tx^B_{Pnsh,i}$ is chained on $\mathbb{L}$ in round $\tau_1 \le \tau_0+\Delta$, publish $(tx^A_{RPay,i}, tx^{A*}_{RPay,i}$, $(\sigma^A_{RPay,i}, \sigma^{A*}_{RPay,i}) \in sig^A_{set,i})$.
4. After $tx^A_{RPay,i}$ and $tx^{A*}_{RPay,i}$ are chained on $\mathbb{L}$ in round $\tau_2 \le \tau_1+\Delta$, set $\Theta^A(\mathsf{id}) := \perp, \Psi^A(\mathsf{id}) := \perp$.

Simulator for ForceClose(id)

Denote τ_0 as the present round.

1. Extract $(tx_F^{A,B}, tx_{set,0}, sig_{set,0}^A, addr_{set}, pkey_{set}^A, pkey_{set}^B)$ from Ψ^A(id) and extract $(tx_{Pay,j}^A, tx_{Pay,j}^{A*})$ from $tx_{set,0}$ and $(\sigma_{Pay,j}^A, \sigma_{Pay,j}^{A*})$ from $sig_{set,0}^A$.
2. Post $(tx_{Pay,j}^A, \sigma_{Pay,j}^A)$ and $(tx_{Pay,j}^{A*}, \sigma_{Pay,j}^{A*})$ on $\mathbb{L}$.
3. Denote $\tau_2 \leq \tau_1 + \Delta$ as the round when $tx_{Pay,j}^A$ and $tx_{Pay,j}^{A*}$ are accepted by $\mathbb{L}$.
4. If tx_{RPay}^B and $tx_{RPay}^{B^*}$ appear on $\mathbb{L}$ at or after round $\tau_2 \leq \tau_1 + \Delta$ and before $\mathbf{T}_d$, post $(tx_{FPay,j}^A, \sigma_{FPay,j}^A)$. Otherwise, post $(tx_{D,j}^A, \sigma_{D,j}^A)$ after $\mathbf{T}_d$. Let $\Gamma^P(id) := \perp$, $\Theta^P(id) := \perp$.

C.2. Simulation Proof

To demonstrate that the AMBiPay protocol UC-realizes $\mathcal{F}_{AM}$, we need to prove that $EXEC_{\Pi_{\mathsf{B}},\mathcal{A},\mathcal{E}}$ and $EXEC_{\mathcal{F}_{AM},\mathcal{S},\mathcal{E}}$ are with computational indistinguishability. We achieve this by showing that, for every environment, the communication with $\mathcal{S}$ and $\mathcal{F}_{AM}$ is computationally indistinguishable from that with $\mathcal{A}$ and Π_{B}. We prove this for each phase of the protocol, including Create, Update, Close, Punish, and the subroutine ForceClose.

We introduce the notation $m[\tau]$ to represent the observation of a message m by the environment in round τ, for clarity and readability. As messages transmitted to parties under adversarial control are only observed after one round, we must also simulate interactions with other functionalities, such as those for signature generation and the blockchain ledger. To account for any observable changes, such as messages directed towards parties under adversarial control or modifications to public variables, we introduce the notation $\texttt{oSet}(action, \tau)$, which captures the set of all side effects that can be observed as a result of the action taken at round τ. We use message identifiers, such as `CREATE` or `createInfo`, to refer to messages in both the ideal and real world simulations. This can check if the same objects are created and if the same checks are conducted.

To implement our construction, we need a 2-party signature scheme Σ that provides EUF-CMA security, and a ledger $\mathbb{L}(\Delta, \Sigma, \mathcal{V})$ that allows for transaction verification using Σ and has the ability to enforce absolute time-locks. Σ ensures that neither the environment nor malicious parties can act as honest parties to create signatures with a non-trivial probability, and only the simulator is capable of acting as honest parties to create signatures.

Lemma 6 *The* Create *phase in the protocol* Π_{B} *achieves the UC-realization of that in the functionality* $\mathcal{F}_{AM}$.

Proof. Let us show the scenario where party A is honest and party B is corrupt. It is worth noting that the situation is symmetric if we swap the roles of A and B.

Real World: Party A receives `CREATE` in round τ_0 and sends `createInfo` to B in the same round. If A receives `createInfo` in round $\tau_0 + 1$, it performs the action $a_0 :=$ "run shared address generation" in round $\tau_0 + 1$. If a_0 is successful, A creates the transactions for the channel and then performs $a_1 :=$ "create signatures" in round $\tau_0+1+\tau_g$. If a_1 is completed, A signs $tx_F^{A,B}$ and transmits the signature via `createFund` to B in round $\tau_0 + 1 + \tau_g + \tau_s$. When A obtains `createFund` from B in round $\tau_0 + 2 + \tau_g + \tau_s$, it performs action $a_2 :=$ "publishing $tx_F^{A,B}$ on $\mathbb{L}$". If $tx_F^{A,B}$ is chained on $\mathbb{L}$ in round $\tau_1 \leq \tau_0 + 2 + \tau_g + \tau_s + \Delta$, A outputs `CREATED`. We denote $EXEC_{\Pi_{\mathsf{B}},\mathcal{A},\mathcal{E}}^{Create} := \{\texttt{createInfo}[\tau_0 + 1], \texttt{oSet}(a_0, \tau_0+1), \texttt{oSet}(a_1, \tau_0+1+\tau_g), createFund[\tau_0 + 2 + \tau_g + \tau_s], \texttt{oSet}(a_2, \tau_0 + 2 + \tau_g + \tau_s), \texttt{CREATED}[\tau_1]\}$ as the execution domain.

Ideal World: Party A sends `CREATE` in round τ_0 to $\mathcal{F}_{AM}$ and the simulator transmits `createInfo` to B. When B sends `createInfo` to A, the simulator notifies $\mathcal{F}_{AM}$ and performs a_0 in round τ_0+1. If a_0 is successful, the simulator creates the transactions for the channel and performs a_1 in round $\tau_0 + 1 + \tau_g$. If a_1 is successful, the simulator acts as A to generate the signature of $tx_F^{A,B}$ and transmits `createFund` to A in round $\tau_0 + 1 + \tau_g + \tau_s$. If B transmits `createFund` to A in round $\tau_0 + 2 + \tau_g + \tau_s$, the simulator performs a_2 in round $\tau_0+2+\tau_g+\tau_s+\Delta$. If the funding tx is chained on $\mathbb{L}$ in round $\tau_1 \leq \tau_0+2+\tau_g+\tau_s+\Delta$, $\mathcal{F}_{AM}$ outputs `CREATED` in round τ_1. We denote the execution domain as $EXEC_{\mathcal{F}_{AM},\mathcal{S},\mathcal{E}}^{Create} := \{\texttt{createInfo}[\tau_0+1], \texttt{oSet}(a_0, \tau_0+1), \texttt{oSet}(a_1, \tau_0 + 1 + \tau_g), \texttt{createFund}[\tau_0 + 2 + \tau_g + \tau_s], \texttt{oSet}(a_2, \tau_0 + 2 + \tau_g + \tau_s), \texttt{CREATED}[\tau_1]\}$.

Lemma 7 *The* ForceClose *subroutine in the protocol* Π_{B} *is the UC-realization of that in the functionality* $\mathcal{F}_{AM}$.

Proof. The following is the case of honest A and corrupted B, and the scenario is symmetric in reverse.

Real World: Action $a_0 :=$ "post $(tx_{Pay,j}^A, \sigma_{Pay,j}^A)$ and $(tx_{Pay,j}^{A*}, \sigma_{Pay,j}^{A*})$ on $\mathbb{L}$" is performed by A in round τ_0, using the latest state. After the transactions $tx_{Pay,j}^A$ and $tx_{Pay,j}^{A*}$ appear on $\mathbb{L}$ in round τ_1 (where $\tau_1 \leq \tau_0 + \Delta$), there are two possible scenarios. First, the transactions $tx_{RPay,j}^B$ and $tx_{RPay,j}^{B*}$ appear on $\mathbb{L}$ in round τ_2 (where $\tau_2 \leq \tau_1 + \Delta$) and before $\mathbf{T}_d$. In this situation, A publishes $(tx_{Fpay,j}^A, \sigma_{Fpay,j}^A)$, and this action is denoted a_1. A also transmits `CLOSED` in round $\tau_m := \tau_3$ (where $\tau_3 \leq \tau_2 + \Delta$). Second, if the transactions $tx_{RPay,j}^B$ and $tx_{RPay,j}^{B*}$ do not appear on $\mathbb{L}$ before $\mathbf{T}_d$, A publishes $(tx_{D,j}^A, \sigma_{D,j}^A)$, and this action is denoted as a_2. A also transmits `CLOSED` in round $\tau_m := \tau_4$ (where $\tau_4 \leq \mathbf{T}_d + \Delta$). Therefore, we denote the execution domain as $EXEC_{\Pi_{\mathsf{B}},\mathcal{A},\mathcal{E}}^{ForceClose} := \{\texttt{oSet}(a_0, \tau_0), o \in \{\texttt{oSet}(a_1, \tau_2), \texttt{oSet}(a_2, \mathbf{T}_d)\}, \texttt{CLOSED}[\tau_m]\}$.

Ideal World: The simulator emulates the real-world behavior by performing the following actions. It conducts a_0 in round τ_0 based on the latest state. After the transactions $tx_{Pay,j}^A$ and $tx_{Pay,j}^{A*}$ appear on $\mathbb{L}$ in round τ_1 (where $\tau_1 \leq \tau_0 + \Delta$), there are two possible scenarios. First, if the transactions $tx_{RPay,j}^B$ and $tx_{RPay,j}^{B*}$ appear on $\mathbb{L}$ in round τ_2 (where $\tau_2 \leq \tau_1 + \Delta$) and before $\mathbf{T}_d$, the simulator posts

$(tx^A_{Fpay,j}, \sigma^A_{Fpay,j})$, the action of which is denoted as a_1. Second, if the transactions $tx^B_{RPay,j}$ and $tx^{B*}_{RPay,j}$ do not appear on $\mathbb{L}$ before $\mathbf{T}_d$, the simulator posts $(tx^A_{D,j}, \sigma^A_{D,j})$, the action of which is denoted as a_2. Simultaneously, if any of these transactions appears on $\mathbb{L}$, $\mathcal{F}_{AM}$ expects it to happen in round τ_m, which is either τ_3 (in case (i)) or τ_4 (in case (ii)), where $\tau_3 \leq \tau_2 + \Delta$ and $\tau_4 \leq \mathbf{T}_d + \Delta$. In this case, it returns CLOSED. Therefore, we denote the execution domain as $EXEC^{ForceClose}_{\mathcal{F}_{AM},\mathcal{S},\mathcal{E}} := \{\mathtt{oSet}(a_0, \tau_0), o \in \mathtt{oSet}(a_1, \tau_2), \mathtt{oSet}(a_2, \mathbf{T}_d), \mathtt{CLOSED}[\tau_m]\}$.

Lemma 8 *The Update phase in the protocol* Π_{B} *is the UC-realization of that in the functionality* $\mathcal{F}$.

Proof. We first focus on the case of honest A and corrupted B.

Real World: When A receives UPDATE in round τ_0, she performs the following steps in the real world: sends an updateReq to B, generates and then signs transactions for the updated state, as well as signs the revoking transaction for B and A successively. The following dependencies and execution order of these steps are visible to $\mathcal{E}$, and $EXEC^{Update}_{\Pi_{\mathsf{B}},\mathcal{A},\mathcal{E}}$ is listed for clarity.

- Initiate by sending updateReq to B in τ_0.
- Transmit SETUP to $\mathcal{E}$ in τ_0+2 if A obtained updateInfo from B.
- Send updateCom to B in $\tau_1 \leq \tau_0+2+\tau_{stp}$ if A obtained SETUP-OK from $\mathcal{E}$.
- Execute SignTxs in $\tau_1 + 1$.
- Send UPDATE-OK to $\mathcal{E}$ in $\tau_1 + 1 + \tau_s$ if the signature generation succeeds.
- Sign B's revoking transaction with B in $\tau_1 + 1 + \tau_s$ if received REVOKE from $\mathcal{E}$.
- Send Revoke to B in $\tau_1 + 1 + \tau_s + \tau_r$ if the signature generation succeeds.
- Sign A's revoking transaction with B in $\tau_1 + 2 + \tau_s + \tau_r$.
- Send UPDATED to $\mathcal{E}$ in $\tau_1 + 3 + \tau_s + 2\tau_r$ if the signature of revoking transaction is obtained from B.

Ideal World: In the idea world, when A sends UPDATE to $\mathcal{F}_{AM}$ in round τ_0, the protocol view is simulated to $\mathcal{E}$ by $\mathcal{S}$. The update phase involves the same steps as in the real world, such as notifying B, generating and signing new state transactions, and signing the revoking transactions for B and A successively. We record the actions taken by $\mathcal{S}$ and $\mathcal{F}_{AM}$ along with their dependencies, which are visible to $\mathcal{E}$. We below list $EXEC^{Update}_{\mathcal{F}_{AM},\mathcal{S},\mathcal{E}}$ for clarity.

- $\mathcal{S}$ transmits updateReq to B in τ_0.
- $\mathcal{F}_{AM}$ transmits SETUP to $\mathcal{E}$ in $\tau_0 + 2$, if it obtained updateInfo from B.
- $\mathcal{S}$ transmits updateCom to B in $\tau_1 \leq \tau_0 + 2 + \tau_{stp}$, if it obtained SETUP-OK from $\mathcal{E}$.
- $\mathcal{S}$ signs the transactions at $\tau_1 + 1$.
- $\mathcal{F}_{AM}$ transmits UPDATE-OK to $\mathcal{E}$ in $\tau_1 + 1 + \tau_s$ if the signature generation succeeds, after being controlled by $\mathcal{S}$.
- $\mathcal{S}$ signs B' revoking transaction in $\tau_1+1+\tau_s$, if it obtained REVOKE from $\mathcal{E}$.
- $\mathcal{S}$ transmits REVOKE to B in $\tau_1+1+\tau_s+\tau_r$, if the signature generation succeeds.
- $\mathcal{S}$ signs A's revoking transaction with B in $\tau_1+2+\tau_s+\tau_r$.
- $\mathcal{F}_{AM}$ transmits UPDATED to $\mathcal{E}$ in $\tau_1 + 3 + \tau_s + 2\tau_r$ if the signature of revoking transaction is obtained from B.

We next focus on the case of honest B and corrupted A.

Real World: When A receives UPDATE in τ_0, she generates and then signs transactions of the updated state, as well as signs the revoking transaction for A and B successively. Below, we list the steps that $\mathcal{E}$ can observe, along with the related dependencies, denoted as $EXEC^{Update}_{\Pi_{\mathsf{B}},\mathcal{A},\mathcal{E}}$.

- Send UPDATE-REQ to $\mathcal{E}$ in τ_0 if it obtained updateReq from A.
- Send an updateInfo message to A in τ_0.
- Send a SETUP-OK message to $\mathcal{E}$ in $\tau_1 \leq \tau_0 + 2 + \tau_{stp}$ if it received an updateCom message from A.
- $\mathcal{A}$ generates and signs the transactions for the new state in τ_1.
- If the previous signing was successful, $\mathcal{A}$ signs B's revoking transaction with A in $\tau_1 + \tau_s$.
- Send a REVOKE-REQ message to $\mathcal{E}$ in $\tau_1 + 1 + \tau_s$ after receiving a REVOKE message from A.
- Sign A's revoking transaction with A in $\tau_1 + 1 + \tau_s + \tau_r$.
- Send a REVOKE message to A in $\tau_1 + 1 + \tau_s + 2\tau_r$ in case the signature generation of revoking transaction succeeds.
- Send an UPDATED message to $\mathcal{E}$ in $\tau_1 + 2 + \tau_s + 2\tau_r$.

Ideal World: When A sends an UPDATE message to $\mathcal{F}_{AM}$ in round τ_0, the execution of the protocol view to $\mathcal{E}$ is simulated by $\mathcal{S}$. Similar to the above real world, the update phase involves generating and then signing transactions of the updated state, as well as signing the revoking transaction for B and A successively. Below, we list the steps that $\mathcal{E}$ can observe, along with the related dependencies and whether they are performed by $\mathcal{S}$ or $\mathcal{F}_{AM}$, denoted as $EXEC^{Update}_{\mathcal{F}_{AM},\mathcal{S},\mathcal{E}}$.

- UPDATE-REQ will be sent to $\mathcal{E}$ in τ_0 by $\mathcal{F}_{AM}$ if received updateReq from A.
- updateInfo will be sent to A in τ_0 by $\mathcal{S}$.
- SETUP-OK will be sent to $\mathcal{E}$ in $\tau_1 \leq \tau_0+2+\tau_{stp}$ by $\mathcal{F}_{AM}$ if it receives updateCom from A.
- SignTxs will be done in τ_1 by $\mathcal{S}$.
- $\mathcal{S}$ will sign B's revoking transaction with A in $\tau_1 + \tau_s$ if the previous signature generation succeeds.
- REVOKE-REQ will be sent to $\mathcal{E}$ in $\tau_1 + 1 + \tau_s$ by $\mathcal{F}_{AM}$ after obtaining Revoke from A.
- $\mathcal{S}$ will sign A's revoking transaction with A in $\tau_1 + 1 + \tau_s + \tau_r$.
- $\mathcal{S}$ will send Revoke to A in $\tau_1 + 1 + \tau_s + 2\tau_r$ in case the signature generation of revoking transaction succeeds.
- UPDATED will be sent to $\mathcal{E}$ in $\tau_1 + 2 + \tau_s + 2\tau_r$ by $\mathcal{F}_{AM}$.

Lemma 9 *The Close phase in the protocol* Π_{B} *achieves the UC-realization of that in the functionality* $\mathcal{F}_{AM}$.

Proof. The case of honest A and corrupted B is shown as follows. We notice that the scenario is symmetric in reverse.

Real World: When A obtains $\texttt{CLOSE}$ in round τ_0, she generates a closing transaction tx_c according to the most recent state of the channel. Then, A signs tx_c with B to obtain the signature, and this action is denoted as a_0. If the signature generation succeeds, A posts tx_c on $\mathbb{L}$ in round $\tau_0 + \tau_r$, denoted by a_1. If tx_c is chained on $\mathbb{L}$ in round $\tau_1 \leq \tau_0+\tau_r+\Delta$, A sends the message $\texttt{CLOSED}$. However, if the signature generation fails in round $\tau_2 \geq \tau_0$, A executes the action $\texttt{ForceClose}$, denoted by a_2. We denote the execution domain as either $EXEC^{Close}_{\Pi_{\mathsf{B}},\mathcal{A},\mathcal{E}} := \{\texttt{oSet}(a_0,\tau_0)$, $\texttt{oSet}(a_1,\tau_0+\tau_r)$, $\texttt{CLOSED}[\tau_1]\}$ or $EXEC^{Close}_{\Pi_{\mathsf{B}},\mathcal{A},\mathcal{E}} := \{\texttt{oSet}(a_0,\tau_0)$, $\texttt{oSet}(a_2,\tau_2)$, $\texttt{CLOSED}[\tau_1]\}$.

Ideal World: After A obtains $\texttt{CLOSED}$ in round τ_0, $\mathcal{S}$ creates the closing transaction tx_c and generates the signature a_0 in round τ_0, while $\mathcal{F}_{AM}$ sends the message $\texttt{CLOSED}$ if tx_c is chained on $\mathbb{L}$ in round $\tau_1 \leq \tau_0+\tau_r+\Delta$. $\mathcal{S}$ also executes the action a_1 by posting tx_c on $\mathbb{L}$ in $\tau_0+\tau_r$. If the signature generation fails in round $\tau_2 \geq \tau_0$, the simulator executes the action a_2 and instructs $\mathcal{F}_{AM}$ to act similarly (i.e., acting as B to not transmit $\texttt{CLOSED}$). We denote the execution domain as $EXEC^{Close}_{\mathcal{F}_{AM},\mathcal{S},\mathcal{E}} := \{\texttt{oSet}(a_0,\tau_0)$, $\texttt{oSet}(a_1,\tau_0+\tau_r)$, $\texttt{CLOSED}[\tau_1]\}$ or $EXEC^{Close}_{\mathcal{F}_{AM},\mathcal{S},\mathcal{E}} := \{\texttt{oSet}(a_0,\tau_0)$, $\texttt{oSet}(a_2,\tau_2)\}$.

Lemma 10 *The Punish phase in the protocol* Π_{B} *achieves the UC-realization of that in the functionality* $\mathcal{F}$.

Proof. The case of honest A and corrupted B is considered below, and the scenario is symmetric in reverse.

Real World: When A obtains $\texttt{PUNISH}$ from $\mathcal{E}$ in round τ_0, it checks whether a transaction belonging to an old state is chained on $\mathbb{L}$. If it finds such a transaction, A utilizes the corresponding revocation secret to perform action a_0, which is posting a punishment transaction in round τ_0. After the punishment transaction is accepted in round $\tau_1 \leq \tau_0 + \Delta$, A performs a_1, which is posting a transaction for unlocking the collateral. If that is accepted in round $\tau_2 \leq \tau_1 + \Delta$, A returns $\texttt{PUNISHED}$. We here denote the execution domain as $EXEC^{Punish}_{\Pi_{\mathsf{B}},\mathcal{A},\mathcal{E}} := \{\texttt{oSet}(a_0,\tau_0)$, $\texttt{oSet}(a_1,\tau_1)$, $\texttt{PUNISHED}[\tau_2]\}$.

Ideal World: This case checks at the end of each round τ_0 whether there is a transaction on the ledger that spends a funding transaction of the old state. If it finds such a transaction and the other party behaves honestly, it anticipates the occurrence of a punishing transaction in round $\tau_1 \leq \tau_0+\Delta$. Also, it anticipates the occurrence of the honest party's unlocking transaction for collateral in round $\tau_2 \leq \tau_1+\Delta$. If both transactions are chained, $\mathcal{F}_{AM}$ returns $\texttt{PUNISHED}$ in round τ_2. The simulator is responsible for publishing both the punishment a_0 and the unlocking transaction for collateral a_1 in rounds τ_0 and τ_1. Therefore, we denote the execution domain in this case as $EXEC^{Punish}_{\mathcal{F}_{AM},\mathcal{S},\mathcal{E}} := \{\texttt{oSet}(a_0,\tau_0)$, $\texttt{oSet}(a_1,\tau_1)$, $\texttt{PUNISHED}[\tau_2]\}$.

Lemma 11 *The protocol* Π_{B} *is the UC-realization of the ideal functionality* $\mathcal{F}_{AM}$.

Proof. By applying Lemmas 6, 7, 8, 9, and 10 and using a standard hybrid argument, we can prove the theorem.

Appendix D. OVTS

D.1. Security Proof

Proof (Privacy of our OVTS). We prove the privacy of our OVTS against a $\mathcal{PPT}$ adversary of depth bounded by $\mathbf{T}^{\epsilon}$ for some non-negative $\epsilon < 1$. We do this by gradually changing the simulation through a series of hybrids and then show the proximity of neighboring experiments.

Hybrid H_0. This is identical to the initial operations of our OVTS.

Hybrid H_1. H_1 is the same as H_0 but without the key generation step of PKE. Instead, it randomly selects a ciphertext $c \leftarrow \mathbb{C}$ from the ciphertext space $\mathbb{C}$ of PKE as the ciphertext of the signature. As the adversary is $\mathcal{PPT}$, the indistinguishability between H_1 and H_0 follows from the $\mathsf{IND\text{-}CPA}$ security of PKE.

Hybrid H_2. In this hybrid, we replace the puzzle of a random key sk_o with a random puzzle $z \leftarrow \mathbb{P}$ from the puzzle space $\mathbb{P}$ of TLP. Since the adversary is depth-bounded, the indistinguishability between this hybrid and the previous one follows from the security of TLP.

Hybrid H_3. In the hybrid H_3, we sample a simulated common reference string crs_{ovts}. The zero-knowledge property of NIZK ensures that this change is computationally indistinguishable.

Hybrid H_4. This hybrid is similar to H_3, but with the proof π_{ovts} computed using the simulator from the underlying NIZK proof. The zero-knowledge property of NIZK ensures that the difference between neighboring hybrids remains negligible in the security parameter.

Simulator $\mathcal{S}$. The simulator remains identical to the last hybrid, and it does not use any information about the witness to compute the proof. This completes our proof.

Proof (Soundness of our OVTS). We analyze both the interactive and non-interactive versions of the protocol. The soundness of the non-interactive protocol follows from the Fiat-Shamir transformation [41] applied to the constant-round protocols. Let $\mathcal{A}$ be an adversary that successfully breaks the soundness of the protocol by generating a commitment c_{ovts} such that $\mathsf{OVTS.Verf}(pp_{ovts}, c_{ovts}) = 1$, but $\mathsf{DS.Verf} pk_s(\sigma) = 0$ where $\sigma \leftarrow \mathsf{OVTS.ForceOpen}(ppovts, c_{\mathsf{ovts}})$.

The soundness of the NIZK proof ensures that a valid proof implies that the signature σ, ciphertext c, and puzzle z are well-formed. This means that the solving process can always output the well-defined value, i.e., the one-time secret key, and furthermore, the ciphertext can be correctly recovered to the valid signature using the one-time secret key, except with negligible probability.

This contradicts to the assumption of $\mathsf{DS.Verf}_{pk_s}(\sigma) = 0$, and thus the soundness of our OVTS holds. In the non-interactive variant of the protocol, the above argument remains valid, as long as the NIZK proof is simulation-sound. Consequently, if we use a simulation-sound scheme to instantiate the NIZK proof, the resulting OVTS scheme also retains the simulation-soundness property.

D.2. An Instantiation of OVTS

- Setup. This algorithm generates the ECDSA parameter $pp_{ds} := (\mathbb{G}, G, q)$, where $\mathbb{G}$ is a cyclic group with a prime order q, and G is a generator of $\mathbb{G}$. It also generates the HTLP parameter $pp_{tlp} := (\mathcal{T}, n, g, h)$, where $\mathcal{T}$ is a hardness parameter, $n := p_1 \cdot q_1,\ g := -\widehat{g}^2 \pmod n$, $h := g^{2^{\mathcal{T}}}$, and secure primes p_1, q_1 are chosen, along with $\widehat{g} \in \mathbb{Z}_n^*$. It is worth noting that a proof [35] can be generated to prove the correct generation of h. Furthermore, it generates the common reference string crs_{zk} for NIZK proofs. Notably, the Pallier encryption parameter pp_{pke} is similar to HTLP since both parameters are chosen by the committer. Thus, the committer's public key is N, and the secret key is $\lambda := (p_1-1)(q_1-1)$, which can be also chosen in the Commit-and-Prove phase. The public parameter is $pp_{ovts} := (pp_{ds}, pp_{tlp}, pp_{pke}, crs_{zk})$.
- Commit-and-Prove. Given the public parameter pp_{ovts} and a ECDSA signature $\sigma = (r, s)$, this algorithm first utilizes N to encrypt s and obtain $ct = (1+n)^s \cdot \alpha^n \pmod{n^2}$ where $\alpha \in \mathbb{Z}_{n^2}$. It is worth noting that r together with its uncompressed point K (whose X-coordinate is r) can be revealed without compromising the privacy of ct. Then, it generates the puzzle $Z := (u, v)$ of the secret key λ, where $u := g^\beta \pmod n, v := h^{n\cdot\beta}(1+n)^\lambda \pmod{n^2}$ for $\beta \in \mathbb{Z}_{n^2}$. Next, it generates a NIZK proof π_{ovts} of language $\mathcal{L}_{otvs} = \mathsf{PoK}\{(x_{ovts}; w_{ovts}) : ct = (1+n)^s \cdot \alpha^n \pmod{n^2} \wedge u := g^\beta \pmod n \wedge v := h^{n\cdot\beta}(1+n)^\lambda \pmod{n^2} \wedge \alpha^{n\lambda} = 1 \pmod{n^2} \wedge \mathcal{H}(m)G + rP := sK\}$, where the statement $x_{ovts} := (n, g, h, ct, u, v, r, K, m, G, P)$, the witness $w_{ovts} := (s, \alpha, \beta, \lambda)$, and P is the public key of ECDSA signature. Finally, it outputs the commitment $c_{ovts} := (x_{ovts}, \pi_{ovts})$.
- OVTS.Verf. Given a commitment $c_{ovts} := (x_{ovts}, \pi_{ovts})$, this algorithm invokes the verification algorithm of NIZK proofs to check its validity.
- OVTS.Open. The committer can use the committed secret key λ to recover the signature $\sigma := (r, s)$.
- OVTS.ForceOpen. This algorithm first computes $w := u^{2^{\mathcal{T}}} \pmod N$ by repeating squaring. Then, it recovers the secret key $\lambda := \frac{v/(w^n) \pmod{n^2} - 1}{n}$ and further decrypts ct to obtain $s := (\frac{ct^\lambda \pmod{n^2} - 1}{n}) \cdot \lambda^{-1} \pmod n$. It finally returns the signature $\sigma := (r, s)$.

Design of the NIZK proof. To realize the above NIZK proof, the intractability is to prove owing α and λ such that $\alpha^{n\lambda} = 1 \pmod{n^2}$. Thus, we applied the transform technology proposed in [50] to transform the above language as $\mathcal{L}_{otvs} = \mathsf{PoK}\{(x_{ovts}; w_{ovts}) : ct = (1+n)^s \cdot \alpha^n \pmod{n^2} \wedge u := g^\beta \pmod n \wedge v := h^{n\cdot\beta}(1+n)^\lambda \pmod{n^2} \wedge R := \alpha \cdot g_1^{r_1} \pmod n \wedge R_1 := r_1 G_2 + r_2 H_2 \wedge R_2 := (n\cdot\lambda)G_2 + r_3 H_2 \wedge R^{n\cdot\lambda} \cdot g_1^{-\delta} := 1 \pmod n \wedge (n \cdot \lambda)R_1 - \delta G_2 - \phi H_2 := 1 \wedge \mathcal{H}(m)G + rP := sK\}$, and $x_{ovts} := (n, g, h, ct, u, v, r, K, m, G, P, R, R_1, R_2, g_1, G_2, H_2), w_{ovts} := (s, \alpha, \beta, \lambda, r_1, r_2, r_3, \delta, \phi)$, where $r_1, \delta, g_1 \in \mathbb{Z}_n, r_2, r_3, \phi \in \mathbb{Z}_q, G_2, H_2 \in \mathbb{G}$.

Then, we applied the Σ-protocol and Fiat-Shamir transformation to realize the above NIZK proof as follows.

- NIZK.Prove. This algorithm randomly chooses $l_s, l_\beta, l_\lambda, l_{r_1}, l_{r_2}, l_{r_3}, l_\delta, l_\phi \in \mathbb{Z}_q,\ l_\alpha \in \mathbb{Z}_n$, and computes $T_{ct} := (1+n)^{l_s} \cdot l_\alpha^n \pmod{n^2}, T_u := g^{l_\beta} \pmod n, T_v := h^{n\cdot l_\beta}(1+n)^{l_\lambda} \pmod{n^2}, T_R := l_\alpha \cdot g_1^{l_{r_1}} \pmod n, T_{R_1} := l_{r_1}G_2 + l_{r_2}H_2, T_{R_2} := n\cdot l_\lambda G_2 + l_{r_3}H_2, T_1 := R^{n\cdot l_\lambda} \cdot g_1^{-l_\delta} \pmod{n^2}, T_2 := (n\cdot l_\lambda)R_1 - l_\delta G_2 - l_\phi H_2, T_K := l_s K$. Then, it computes $c := \mathcal{H}(x_{ovts}, T_{ct}, T_u, T_v, T_R, T_{R_1}, T_{R_2}, T_1, T_2, T_K)$ and further computes $z_s := l_s + s\cdot c \pmod{q\cdot n}, z_\alpha := l_\alpha \cdot \alpha^c \pmod{q\cdot n}, z_\beta := l_\beta + \beta\cdot c \pmod{q\cdot n}, z_\lambda := l_\lambda + \lambda\cdot c \pmod{q\cdot n}, z_{r_1} := l_{r_1} + r_1\cdot c \pmod{q\cdot n}, z_{r_2} := l_{r_2} + r_2\cdot c \pmod q, z_{r_3} := l_{r_3} + r_3\cdot c \pmod q, z_\delta := l_\delta + \delta\cdot c \pmod{q\cdot n}, z_\phi := l_\phi + \phi\cdot c \pmod q$. It finally returns the proof $\pi_{ovts} := (c, z_s, z_\alpha, z_\beta, z_\lambda, z_{r_1}, z_{r_2}, z_{r_3}, z_\delta, z_\phi)$.
- NIZK.Verf. Given a statement x_{ovts} and a proof π_{ovts}, this algorithm computes $T'_{ct} := (1+n)^{z_s} \cdot z_\alpha^n \cdot ct^{-c} \pmod{n^2}, T'_u := g^{z_\beta} \cdot u^{-c} \pmod n, T'_v := h^{n\cdot z_\beta} \cdot (1+n)^{z_\lambda} \cdot v^{-c} \pmod{n^2}, T'_R := z_\alpha \cdot g^{z_{r_1}} \cdot R^{-c} \pmod n, T'_{R_1} := z_{r_1}G_2 + z_{r_2}H_2 - cR_1, T'_{R_2} := n\cdot z_\lambda G_2 + z_{r_3}H_2 - cR_2, T'_1 := R^{n\cdot z_\lambda} \cdot g_1^{-z_\delta} \pmod{n^2}, T'_2 := n\cdot z_\lambda R_1 - z_\delta G_2 - z_\phi H_2, T'_K := z_s K - c[\mathcal{H}(m)G + rP]$, and $c' := \mathcal{H}(x_{ovts}, T'_{ct}, T'_u, T'_v, T'_R, T'_{R_1}, T'_{R_2}, T'_1, T'_2, T'_K)$. It then returns 1 if $c' = c$, and returns 0 otherwise.