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

***Abstract*—Bidirectional Payment Channels (Bi-PC) offer a compelling solution to blockchain scalability challenges by enabling the majority of two-way transactions to occur off-chain. However, many Bi-PCs rely either on a trusted third party, which compromises the decentralization of cryptocurrencies, or on scripts, which introduce on-chain coding vulnerabilities, high on-chain costs, and limited compatibility. Recently, Aumayr et al. in CCS'22 introduced Sleepy Channels, a trustless and scriptless Bi-PC protocol for unspent transaction output (UTXO) models. They raised the open question of extending Sleepy Channels to account models. Two significant challenges arise: (i) *Uncertainty*: The single-input-single-output and nonce mechanisms of the account model can lead to different parties owning transactions with the same nonce, which disrupts transaction flow and causes unexpected fund assignments. (ii) *Weak Constraint*: Transactions spending directly from account balances allow one party to unilaterally perform a fast-finish payment of old states, which leads to invalid punishments and economic losses for the counterparty.**

In response, we present *AMBiPay*, the first trustless and scriptless bi-PC protocol for the account model. Our solution leverages specialized *fork-then-sleepy* channels and a tailor-made *strong constraint* mechanism based on adaptor signatures and time-lock puzzles. This ensures the sequential execution of transactions and enables valid punishment mechanisms. However, constructing adaptor signatures for unique signatures is proven impossible by Erwig et al. in PKC'21. To overcome this challenge, we adopt the concept of 2-Party Weak Adaptor Signatures (2PWAS) as well as redefine formal security models and enhance the efficiency. Our novel construction achieves 2.52$\times$ and 1.75$\times$ time efficiency improvements for pre-signing and verification algorithms, respectively, while reducing the pre-signature size by approximately 1.47$\times$. Additionally, we introduce an optimized construction for verifiable timed signatures, reducing the computation and communication costs from $O(N)$ to $O(1)$, which is crucial for efficiency in scriptless designs. These contributions hold independent significance in the fields of adaptor signatures and verifiable timed signatures. Our evaluation shows that the time and communication costs of AMBiPay are comparable to state-of-the-art bi-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 leads to high transaction fees and slow confirmation times, which hinders the use of decentralized payments. For example, the current size of Ethereum 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 significantly improves blockchain storage, boosts transaction throughput, and lowers 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 scalability) 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. Two common strategies have emerged to address this concern, each tailored to different types of transactions: the Unspent Transaction Output (UTXO) model and the account model.

In the UTXO model, specific scripts, not Turing-complete and scripted in a stack-based language, establish a punishment mechanism without any Trusted Third Party (TTP). This mechanism operates as a deterrent, which prevents either party from establishing an outdated balance. For instance, when Alice tries to close the channel with an outdated balance, there is a short punishment time period, referred to as a *relative timelock*, during which Bob can claim the coins in that balance using a revocation key. This approach has been effectively employed in the Lightning Network [6] and Generalized Channels [7]. In contrast, the account model primarily relies on smart contracts, which are equipped with Turing-complete script languages. Smart contracts offer a versatile approach to establish trust independence in State Channels [8], [9], [10]. Leveraging the Turing-completeness, these contracts can implement mutually agreed-upon rules and streamline payments without

the need for a TTP. Even in currencies with limited smart contract functionality, such as Stellar, scripts are employed. For example, the bi-PC protocol Starlight [11] for Stellar uses relative timelocks to facilitate off-chain payments.

While the above approaches successfully address trust issues, they also introduce concerns regarding security and universality. Script-based solutions are prone to coding errors, vulnerabilities, and uncharted attack vectors, as evidenced by past Ethereum smart contract exploits like the DAO attack [12]. Moreover, they come with high on-chain computational costs, leading to elevated transaction fees, and compromise user privacy by exposing sensitive transaction details through on-chain script recording. Additionally, the absence of support for particular features in certain well-known cryptocurrencies, such as the lack of specific script types (e.g., relative timelock) in Monero and Mimblewimble, or limited smart contract capabilities in currencies like Ripple and e-CNY, imposes constraints on the universal application of script-based bi-PCs.

Recently, Sleepy Channels [13] present a trustless and scriptless bi-PC. They utilize absolute timelocks instead of relative timelocks to eliminate the need for real-time monitoring and ensure punishment within a predefined time. Verifiable Timed Signatures (VTS) [14], [15] facilitate absolute timelocks, making Sleepy Channels compatible with scriptless currencies. The collateral and negligible amount mechanisms serve to encourage fast-finish payments. In this scenario, the initiator can swiftly finalize payments if the other party responds and transfers a small amount of coins to a specified address. Conversely, if the other party remains unresponsive, lazy-finish payments are activated, requiring the initiator to wait until a predetermined time. This setup ensures that if one party, such as Alice, posts an old payment on-chain, she must wait for a set duration to complete the payments, which gives the other party, like Bob, the opportunity to penalize Alice.

Despite the compatibility of Sleepy Channels with UTXO-based currencies, their suitability for account-based currencies remains uncertain due to distinct transaction formats. In UTXO, transactions involve multiple inputs and outputs, and spend coins from previous transactions (identified by transaction hash and output index). In contrast, the account model directly spends from account balances using a single-input-single-output format, incorporating a nonce mechanism to prevent double-spending [16]. These differences pose two primary challenges for implementing Sleepy Channels in account models: i) *Uncertainty*, as generating multiple transactions with sequentially increasing nonces in the account model may lead to different parties owning transactions with the same nonce. This causes potential disruption of transaction flow and unexpected fund assignments. ii) *Weak Constraint*, where the effectiveness of negligible amount mechanism weakens because transactions in the account model spend directly from account balances. A malicious party can transfer negligible amounts from other accounts to the specified one, which allows the posting of old payment statuses and spontaneous completion of fast-finish payments without valid punishment. In light of these challenges, Aumayr et al. [13] 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.

1.1. Our Contribution

In this work, we answer the above open problem affirmatively. We present AMBiPay, a new bi-PC protocol tailored for the account model, effectively addressing the challenges outlined above. Our contributions unfold as follows.

- We design tailor-made *fork-then-sleepy* channels for the account model. In doing so, we combine the nonce mechanism, adaptor signatures and Time-lock Puzzles (TLP) to establish a *strong constraint* mechanism and effectively resolve the uncertainty challenge. Thus, we bypass the negligible amount mechanism while preserving the advantages of being TTP-free, script-independent and enabling fast-finish (see Table 1). We formalize AMBiPay in the Universal Composability (UC) framework and rigorously prove its security.
- Adaptor signatures can enforce the strong constraint mechanism, but their impossibility in unique signatures [20] will limit the versatility of AMBiPay. This limitation arises from the witness extractability of adaptor signatures and the uniqueness of unique signatures. An adversary, given a public statement of a hard relation, can create a secret/public key pair, compute the pre-signature based on the statement and the signature of a message, and then extract the witness of the hard relation directly from these signatures. This violates the hardness of the relation. To mitigate this, we leverage the concept of *2-Party Weak Adaptor Signatures* (2PWAS) to restrict the adversary's ability to generate signatures. We redefine the security models of 2PWAS to capture adversary capabilities with unique signatures and propose a more efficient 2PWAS construction applicable to both unique and common signatures.
- While AMBiPay can achieve compatibility with scriptless cryptocurrencies via VTS, the computation and communication costs for generating VTS signatures increase linearly. Thus, we propose an *Optimized Verifiable Timed Signatures* (OVTS) construction with constant computation and communication 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 TLP only to the one-time key.
- We demonstrate the feasibility of our proposals through concrete instantiations and comparisons of 2PWAS and OVTS. The implementation of AMBiPay is tested on Sepolia, an Ethereum testnet, without relying on smart contract functionality. Rigorous simulations are conducted using the MetaMask-Chrome plugin to connect to Sepolia. During these simulations, transactions are deployed, and

TABLE 1: Comparison among bi-PC protocols. Trustless means no need for a TTP during payments, scriptless implies no script, 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 [17]. 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 [17]	UTXO	✓	✓	✓	✗	$\log d$
Eltoo [18]	UTXO	✓$^-$	✗	✗	✓	2
Teechan [3]	UTXO	✗	✓	✓	✓	1
BlindHub [19]	UTXO	✗	✓	✓	✗	2
Lightning Network [6]	UTXO	✓$^-$	✗	✗	✓	1
Generalized Channels [7]	UTXO	✓$^-$	✗	✗	✓	2
Sleepy Channels [13]	UTXO	✓	✓	✓*	✗	1
Starlight [11]	Account	✓	✗	✓	✗	2
State Channels [8]	Account	✓	✗	✗	✓	1
AMBiPay	Account	✓	✓	✓	✗	1

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

the overhead is precisely measured. Evaluation results indicate that the efficiency of AMBiPay closely aligns with that of Sleepy Channels.

1.2. Related Work

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

UTXO-based PC. Duplex Channels [17] enable bi-directional channels with absolute timelocks, which achieves 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 [18] 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 goes against the decentralized nature of currencies.

Watchtower-based proposals, such as Outpost [21], Cerberus Channels [22], FPPW [23], and BlindHub [19], 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. Recently, Sleepy Channels [13] were introduced as a solution that eliminates script dependencies and TTPs. Nevertheless, the applicability of Sleepy Channels to account-based currencies remains unclear, as uncertainties and weak constraints can arise when adapting them to this context.

Account-based PC. Ethereum, an account-based cryptocurrency, allows for complex smart contracts. State Channels [8], [9], [10] 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. Furthermore, other account-based currencies like Ripple, Stellar and e-CNY lack programmable smart contract functionality. While Starlight [11] offers a bi-PC protocol for Stellar without any TTP, it still utilizes relative timelocks for payments and thus faces the script dependency issue. This has prompted an exploration of scriptless bi-PC for the account model, which facilitates diverse computations without exposing potential security risks associated with smart contracts and other scripts. To the best of our knowledge, no work currently exists in this field.

Adaptor Signatures. Adaptor signatures [20], have been widely adopted in blockchains such as PC networks [24], PC hubs [19], atomic swaps [25], and oracle-based payments [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. [20] provided a generic construction for specific signatures, while Dai et al. [28] offered a generic construction from any signature scheme and hard relation, though it requires generating a new hard relation during signing, which makes it unsuitable for unique signature schemes. Recently, Gerhart et al. [29] demonstrated that previous adaptor signature schemes are only secure in payment channels and not in PC hubs or oracle-based payments. They proposed a more secure construction using dichotomic signatures. However, these approaches cannot construct adaptor signatures from unique signatures, and their security models are limited to the existential unforgeability of 2PWAS for common signatures, which makes them unsuitable for scenarios involving unique signatures.

In a different research line, Bursuc et al. [30] introduced 2PWAS, where the pre-signing algorithm uses a witness y as input, unlike the previous definition that uses a statement Y. This weaker notion can address the previously mentioned impossibility. To safeguard their coins, the party selecting the witness keeps it confidential, which ensures only one

of the two parties with the witness can obtain the signature. This makes it challenging for adversaries to generate the signature. With hard relations in unique signatures, adversaries cannot extract the witness by generating the signature and pre-signature. However, Bursuc et al. followed the previous security definition which does not capture adversary capabilities for unique signatures, and only proposed an ECDSA-based solution without exploring construction from unique signatures. Additionally, their approach requires two Non-Interactive Zero-Knowledge (NIZK) proofs to prove and verify the correct generation of signatures. This motivates us to leverage the weaker notion from Bursuc et al. [30] to solve the impossibility, while also redefining the security models of 2PWAS and improving its efficiency.

VTS. VTS [14] enables time-locking a signature for a predefined time interval $\mathbf{T}$. Its verifiability ensures that anyone can confirm the recovery of the signature after a sequential computation of time $\mathbf{T}$. To enhance the efficiency and functionality of VTS, Verifiable Timed Discrete Logarithm (VTDL) [25] and Verifiable Timed Linkable Ring Signatures (VTLRS) [15]. These primitives are applied in PC and atomic swaps to eliminate script dependency.

2. Solution Overview

This section provides an overview of AMBiPay, initially reviewing Sleepy Channels [13] and applying them to the account model, but noting the limitations of this application. It then presents our solution.

Original Sleepy Channels. Two parties, A and B, lock coins in a shared address Ch_{AB} (referred to as the channel), as illustrated in the left sub-figure of Figure 1. Payments are facilitated through transactions tx_{Pay}, tx_{FPay} and tx^{*}_{FPay} that update both parties' balances. Each party possesses specific transaction versions, such as $(tx^{A}_{Pay}, tx^{B,A}_{FPay}, tx^{A*}_{FPay})$ for A. When A publishes tx^{A}_{Pay} to close the channel, two scenarios unfold: i) Fast-finish, if B is responsive and posts $tx^{A,B}_{FPay}$ to redeem coins, allowing A to issue tx^{A*}_{FPay} for an immediate channel closure; ii) Lazy-finish, if B is unresponsive, prompting A to wait until timelock $\mathbf{T}_d$ expires. For further payments, old transactions are revoked by jointly generating a punishment transaction from $\mathsf{SleepyCh}$. If one party maliciously issues an old payment status, the honest party can use the punishment transaction to claim the balance of the malicious party.

In UTXO, $tx^{A,B}_{FPay}$ and tx^{A*}_{FPay} are executed in sequence after tx^{A}_{Pay} is posted on-chain. This ensures that a malicious party (e.g., A) must wait until timelock $\mathbf{T}_d$ if the other party refuses to post $tx^{A,B}_{FPay}$. The honest party (i.e., B) then has until $\mathbf{T}_d$ to punish the malicious party. While these constraints guarantee sequential execution and valid punishment, applying Sleepy Channels directly to the account model introduces two limitations, *Uncertainty* and *Weak Constraint*, which will be discussed later.

Sleepy Channels in the account model. Moving from UTXO to the account model (see the right sub-figure in Figure 1), A and B initiate payments by locking coins in the shared channel Ch_{AB}. In this model, parties utilize four transaction types: redeem (tx_R), payment (tx_{Pay}), redeem-payment (tx_{RPay}), and real-redeem-payment (tx^{*}_{RPay}) for spending and updating balances. Each party (e.g., A) has specific transactions (tx^{A}_{R}, tx^{A}_{Pay}, tx^{A}_{RPay}, and tx^{A*}_{RPay}) for unilateral payments. Fast-finish and lazy-finish payments use extra transaction types: tx^{A}_{FPay} and tx^{A}_{D}. The punishment transaction (tx_{Pnsh}) is also necessary to punish the party who posts an old payment status. The absolute timelock $\mathbf{T}_d$ in tx^{A}_{D} prevents A from posting an old state before $\mathbf{T}_d$, which allows B to punish A. However, two limitations emerge:

(i) Uncertainty. In the account model, we can generate multiple transactions with sequentially increasing nonces to enable the multi-input-multi-output functionality. However, this approach introduces uncertainty as parties might compete to post transactions with the same nonce. For instance, after party A posts tx^{A}_{R} with nonce "0" to redeem her collateral c, party B can post tx^{B}_{Pay} with nonce "1" and tx^{B}_{RPay} with nonce "2" to obtain $2v_B{+}c$ coins. Even if party A posts tx^{A}_{R} and tx^{A}_{Pay} simultaneously, party B can pre-post tx^{B}_{Pay} in the pending transaction pool. The completion of these transactions on-chain, without any timelock, renders the punishment transaction ineffective.

(ii) Weak Constraint. In the original Sleepy Channels, a negligible amount ε is utilized in fast-finish payments. Due to the multi-input-multi-output setting of UTXO models, the negligible amount is spent from exactly B's previous transaction output and cannot be funded through any other means. This ensures that 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. However, in the account model, where transactions directly spend from account balances, a potential vulnerability arises. Party A can initiate the process by posting the old payments of tx^{A}_{R} and tx^{A}_{pay}. Subsequently, A can transfer ε coins from other account balances to the specified address and execute the fast-finish payment even if party B is unresponsive. This significantly weakens the constraint imposed by the negligible amount mechanism, and consequently diminishes the efficacy of the punishment mechanism.

Achieving certainty and strong constraint. We introduce a redesign of *fork-then-sleepy* channels (depicted in Figure 2). This redesign creates a forking channel FCh_A that combines tx^{A}_{R} and tx^{A}_{Pay} into a single payment transaction tx^{A}_{Pay}. From this, two transactions tx^{A*}_{Pay} and tx^{A}_{R} are generated and spent from FCh_A to the sleepy channel $\mathsf{SleepyCh}_A$ 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^{A}_{Pay}. Additionally, the weak constraint between tx^{B*}_{RPay} and tx^{A}_{FPay} persists.

We further replace the negligible amount mechanism with adaptor signatures and TLPs to achieve a strong constraint. By applying adaptor signatures to both tx^{B*}_{RPay} and tx^{A}_{FPay} with the same witness, the process unfolds as follows: Initially, party A possesses only the pre-signatures of

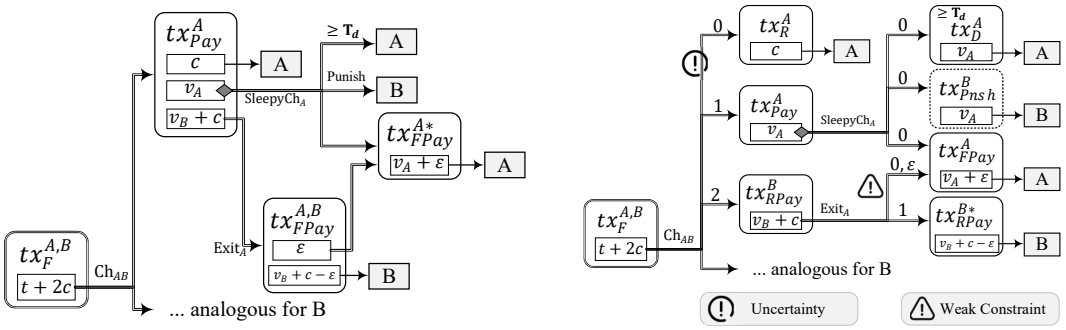

Figure 1: *Left sub-figure (cited from [13])*: Transaction flow of original Sleepy Channels in the UTXO model between parties A and B. *Right sub-figure*: Transaction flow for applying Sleepy Channels in the account model. Rounded boxes represent transactions with input and output arrows. Double lines 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.

tx_{FPay}^{A} and tx_{RPay}^{B*}, and party B has the pre-signature of tx_{RPay}^{B*} along with the witness. The pre-signature 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 witness extractability of adaptor signatures) and subsequently adapt the signature of tx_{FPay}^{A}. This guarantees a strict sequential execution between tx_{RPay}^{B*} and tx_{FPay}^{A}, akin 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 on-chain 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. The aforementioned impossibility [20] poses a challenge for the application of AMBiPay in blockchains equipped with unique signatures. For example, Beacon Chain, which employs BLS signatures for aggregatable transactions and practical consensus [31]. Intuitively, resolving the impossibility hinges on restricting $\mathcal{A}$'s ability to generate signatures. This restriction is achievable by allowing only one party (named as signature holder) to acquire the final signature in 2-party scenarios [32] [33] [34]. However, when dealing with external hard relations, parties may still collude to generate the signature. To counter this, we additionally require that the witness for a given hard relation must be chosen by one of the two parties and used during the pre-signing process. This requirement effectively thwarts collusion and resolves the earlier impossibility, as the party selecting the witness will keep it confidential to protect their coins. While Bursuc et al. [30] incorporated this additional requirement in their weak adaptor signatures, they did not formalize security definitions to capture adversary capabilities with unique signatures and only proposed an ECDSA-based design without exploring constructions from unique signatures.

When applying the previous security definition (e.g., existential unforgeability) [7], [20], [28] to 2PWAS constructed from unique signatures, we encounter a significant insufficiency. This definition involves an adversary attempting 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 definition works for common signatures, it fails for unique signatures. In the case of unique signatures, the adversary can query a different message m' along with Y^* to the oracles, extract the witness y^* based on the properties of uniqueness and witness extractability, and adapt it to forge the signature σ^* for the original challenged message m^*. To address this, our redefined existential unforgeability adds two oracles: the honest relation oracle and the corrupted relation oracle. The former oracle only returns the statement, while the latter oracle returns both the statement and the witness. The adversary can query both oracles, but the challenged statement Y^* must remain uncorrupted. This approach ensures the existential unforgeability of 2PWAS even for unique signatures, and closes the gap left by the previous security definition.

To ensure the versatility of AMBiPay, we propose a more efficient generic construction of 2PWAS for both unique and common signatures. We improve the efficiency by reducing the number of required NIZK proofs from two to one, as the remaining proof for the hard relation is unavoidable. Our approach draws inspiration from blinding

techniques used in blind and adaptor signatures. The signature holder 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 adapted through a de-blinding operation, while simultaneously the blinding key can be extracted from the ciphertext and message.

Optimizing efficiency. Applying VTS [14] in AMBiPay can eliminate the absolute timelock, but in VTS, the prover needs to generate $O(n)$ shares of public keys, signatures, and TLPs. This results in a linear increasing computation cost [1]. Inspired by verifiably encrypted signatures [35] and VTS [14], we propose an OVTS construction, where the entire signature is encrypted using a one-time key pair, and the TLP is applied to the one-time key. NIZK proofs are used to ensure verifiability. By force-opening the TLP, one can obtain the secret key and then 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 various 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 (Existential UnForgeability under adaptive Chosen Message Attacks) and IND-CPA (INDistinguishability under Chosen Plaintext Attacks) are desirable security properties for digital signature and public key encryption schemes, respectively.

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 nonce representing 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 spending from account balance and single-input-single-output. The UTXO model facilitates multi-input-multi-output transactions, where each transaction spends from existing transaction outputs. This ensures that when a transaction is posted on-chain, multiple transaction outputs take effect simultaneously.

1. VTDL [25] also faces this issue, even though it is more efficient than VTS by committing the signing key instead of the signature itself.

Universal Composability (UC). To model security in concurrent execution scenarios, we employ the UC framework with global setup [13]. 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 [36] 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 a statement / 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 $b := 0$ signifies invalid. A NIZK proof system should fulfill completeness and soundness, and zero-knowledge.

Adaptor Signatures. An adaptor signature scheme AS = (pSign, pVerf, Adapt, Extract) [20] relates to a hard relation $(Y; y) \in \mathsf{R}$, allowing signers to create pre-signatures based on a statement Y. Publishers can adapt these into valid signatures with a witness y, and upon publication of the signatures, the signer can recover the witness y. The pre-signing algorithm pSign inputs a secret key sk, a statement Y and a message m, and it outputs a pre-signature $\widehat{\sigma}$. The verification algorithm pVerf inputs a public key pk, a statement Y, a message m, and a pre-signature $\widehat{\sigma}$, and it outputs 1 if $\widehat{\sigma}$ is valid, and 0 otherwise. The adaptor algorithm Adapt inputs a pre-signature $\widehat{\sigma}$ and a witness y, and it outputs a signature σ. The extraction algorithm Ext inputs a signature σ and a pre-signature $\widehat{\sigma}$, and it outputs a witness y.

Weak adaptor signatures introduced in [30] also consist of the above four algorithms, but they involve two hard relations (i.e., adaptor relation R_a and verification relation R_v), and the pSign algorithm takes as input the witnesses instead of the statements. For instance, R_a and R_v in [30] are instantiated as: $\mathsf{R}_a = (Y; y) | Y = yG$ and $\mathsf{R}_v = (m, pk, Z; \sigma = (r, s)) | \mathsf{Verf}_{pk}(m, \sigma) = 1 \wedge Z = (r, sG)$, where G is a generator of a additive cyclic group and $\sigma = (r, s)$ is a ECDSA signature. Here, pSign inputs a secret key sk, two witnesses y and z for R_a and R_v respectively, and a message m. It outputs a pre-signature $\widehat{\sigma}$, and two proofs, π_a for R_a and π_v for R_v. pVerf inputs a public key pk, two statements Y and Z, a message $m \in \{0,1\}^*$, a pre-signature $\widehat{\sigma}$, and two proofs, π_a for R_a and π_v for R_v.

It outputs 1 if $\widehat{\sigma}$, π_a and π_v are valid, and 0 otherwise.

TLP. A time-lock puzzle scheme TLP = (Setup, PGen, PSolve) [37] empowers concealing a value for a specific period, strictly greater than $\mathbf{T} \in \mathbb{N}$. PGen is a probabilistic algorithm that takes a hardness $\mathbf{T}$, a solution $s \in \{0,1\}^*$, and a randomness 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 for the most parallel algorithms) adversary $\mathcal{A}$ with running time $\leq \mathbf{T}^{\epsilon}(\lambda)$ can distinguish between two puzzles generated with solutions s_0 and s_1 respectively, both with timing hardness $\mathbf{T}$, except with negligible probability ϵ.

VTS. A VTS scheme VTS = (Setup, Commit-and-Prove, Verf, Open, and ForceOpen) [14] enables time-locking a signature on a specific message for a predetermined time $\mathbf{T}$. 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}, which will takes time $\mathbf{T}$. A secure VTS scheme must satisfy two security properties: (i) *soundness*, ensuring that VTS.ForceOpen 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. The Proposed AMBiPay

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

4.1. Construction

We provide an overview of AMBiPay in Figure 2 and present its formal design, which includes strong constraints, incentive mechanisms, initialization, address generation, and the four phases: payment, payment revocation, channel closing, and revoked payment punishment.

Strong Constraint. With the redesigned *fork-then-sleepy* channels presented in Section 2, we employ the nonce mechanism, along with TLPs and adaptor signatures, to ensure sequential transaction execution. When two transactions, like tx^A_{Pay} and tx^B_{Pay}, spend from the same channel with the same nonce, TLP comes into play. Party A generates a TLP $z_{A,1}$ of her chosen witness $y_{A,1}$ upon a predefined timelock $\mathbf{T}_b$ (typically a block generation time).

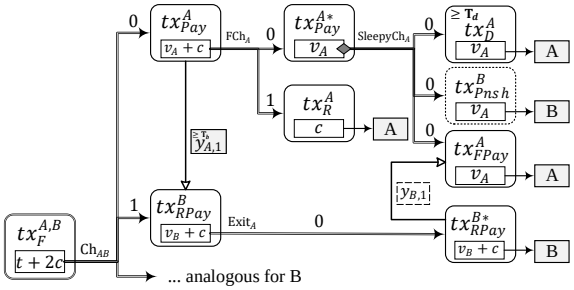

Figure 2: Transaction flow of AMBiPay. Three types are involved: (i) a double solid arrow for a shared address with the nonce i; (ii) a single solid arrow for a single-party address; and (iii) a single hollow arrow for the constraint condition of a transaction opened by a witness. $y_{A,1}$ and $y_{B,1}$ are two witness owned by A and B respectively. 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 $v_A + v_B = t$.

Parties A and B then collaboratively create pre-signatures for transactions tx^A_{Pay} and tx^B_{RPay} using $z_{A,1}$ and $y_{A,1}$ respectively. Party A exclusively possesses the puzzle $z_{A,1}$ to adapt the signature of tx^A_{Pay}. The witness extractability of adaptor signatures ensures that party B can extract $z_{A,1}$ from the signature and pre-signature of tx^A_{Pay} only if party A posts the signature on-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^B_{RPay}. Similarly, for tx^A_{FPay} and tx^{B*}_{RPay}, both pre-signatures directly embed $y_{B,1}$ without TLP 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 [37] 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 [38] and Fiat-Shamir transformation [39].

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

Incentive Mechanism. In AMBiPay, we also employ the collateral incentive mechanism [13] to encourage prompt responses and facilitate 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 unidirectional 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 $\mathbf{T}_d$.

Initialization. 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 $tx^{A,1}_{rf} := (\mathsf{Ch}_{AB}, pk_A, v_A + c, 0), tx^{A,2}_{rf} := (\mathsf{Ch}_{AB}, pk_A, v_A + c, 1)$. 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, z is the time-lock puzzle of y, and Z represents the corresponding public information (e.g., statement) of z. Below, we denote KG, Γ_{AKG}, Γ_{pSign}, Γ_{Sign}, Adapt, Ext as key generation, joint key generation, pre-signing, signing, adaption, extraction algorithms for the 2-party adaptor signature scheme Π_{AS}, respectively.

Address Generation. To launch the payment channel, parties 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 FCh_A, FCh_B, $\mathsf{SleepyCh}_A$, $\mathsf{SleepyCh}_B$, 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 they execute:

Payment Transactions. For parties to launch the i-th payment, assemble payment transactions $tx^A_{Pay,i} := tx(\mathsf{Ch}_{AB}, \mathsf{FCh}_A, v_{A,i} + c, 0), tx^B_{Pay,i} := tx(\mathsf{Ch}_{AB}, \mathsf{FCh}_B, v_{B,i}+c, 0), tx^A_{RPay,i} := tx(\mathsf{Ch}_{AB}, \mathsf{Exit}_B, v_{A,i}+c, 1)$, as well as $tx^B_{RPay,i} := tx(\mathsf{Ch}_{AB}, \mathsf{Exit}_A, v_{B,i} + c, 1)$. 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 $tx^{A*}_{Pay,i} := (\mathsf{FCh}_A, \mathsf{SleepyCh}_A, v_{A,i}, 0)$, $tx^A_{R,i} := (\mathsf{FCh}_A, pk_A, c, 1)$, $tx^{B*}_{Pay,i} := (\mathsf{FCh}_B, \mathsf{SleepyCh}_B, v_{B,i}, 0)$, $tx^B_{R,i} := (\mathsf{FCh}_B, pk_B, c, 1)$.

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 below transactions are assembled to achieve lazy-payment or fast finish.

- 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 $tx^{B*}_{RPay,i} := (\mathsf{Exit}_A, pk_B, v_{B,i} + c, 0)$.

Generating (Adaptor) Signatures. Parties jointly compute the (adaptor) signatures for the above transactions via $\Pi_{\mathsf{AS}}.\Gamma_{\mathsf{pSign}}$ or $\Pi_{\mathsf{AS}}.\Gamma_{\mathsf{Sign}}$. The TLPs 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 TLPs 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 FCh_B.
- Party A obtains signature $\sigma^A_{D,i}$ on $tx^A_{D,i}$ under $\mathsf{SleepyCh}_A$. Party B obtains signature $\sigma^B_{D,i}$ on $tx^B_{D,i}$ under $\mathsf{SleepyCh}_B$.
- 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 post the recent unrevoked payment to close the channel.

- 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 $\mathsf{SleepyCh}_A$.

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 $(tx^{A}_{Pay,j}, \sigma^{A}_{Pay,j})$ and $(tx^{A*}_{Pay,j}, \sigma^{A*}_{Pay,j})$ on $\mathbb{L}$. Afterward, party A can independently receive her collateral c by posting $tx^{A}_{R,j}$ 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 uses $\widehat{\sigma}^{B}_{RPay,j}(Y_{A,1,j})$ 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 $\Pi_{\mathsf{AS}}.\mathsf{Adapt}$. After B publishes $(tx^{B*}_{RPay,j}, \sigma^{B*}_{RPay,j})$ on $\mathbb{L}$, A can recover signature $\sigma^{A}_{FPay,j}$ via sequentially invoking $\Pi_{\mathsf{AS}}.\mathsf{Ext}$ and $\Pi_{\mathsf{AS}}.\mathsf{Adapt}$. Finally, A posts $(tx^{A}_{FPay,j}, \sigma^{A}_{FPay,j})$ on $\mathbb{L}$ and finishes 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) The version of transaction flow for party B is analogous, and the only difference is the roles are reversed.

Revoked Payment Punishment (On-Chain). Assume that party A posts the j-th revoked payment $(tx^{A}_{Pay,j}, \sigma^{A}_{Pay,j})$ on $\mathbb{L}$, party B can post 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 posts the j-th revoked payment $(tx^{B}_{Pay,j}, \sigma^{B}_{Pay,j})$ on $\mathbb{L}$, party A can post 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, but the innocent party will receive $t + c$ coins.

4.2. Security

Intuitively, AMBiPay ensures predefined transaction order, addressing uncertainty and weak constraints. The collateral mechanism incentivizes the fast channel finalization. Punishment deters dishonest behavior by revoking old payment states. Scriptless bi-party computation enhances security by eliminating vulnerabilities caused by smart contracts.

We next present our central hypothesis and summarize our analysis. Appendix A contains a formal specification of AMBiPay Π_{B} in the UC framework, which differs from the one described in Section 4, 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 [13], 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 A, we simulate $\mathcal{S}$ to interact with the ideal functionality $\mathcal{F}_{AM}$ defined in Appendix A, while the environment $\mathcal{E}$ to interact with $\phi_{\mathcal{F}_{AM}}$ (the ideal protocol for $\mathcal{F}_{AM}$). We then in Appendix B, prove that any attack against Π_{B} can be performed against $\phi_{\mathcal{F}_{AM}}$. Consequently, we conclude the theorem as follows.

Theorem 1 *The protocol Π_{B} UC-realizes the ideal functionality $\mathcal{F}_{AM}$.*

4.3. Extension

Optimized 2PWAS. To address the aforementioned impossibility and ensure the versatility of AMBiPay, we leverage the concept of 2PWAS [30] while redefining its semantics as well as its correctness and security (i.e., existential unforgeability, pre-signature adaptability, and witness extractability) (see Appendix C). In this redefinition, we eliminate the inherent NIZK proof required for R_v in the original definition of 2PWAS (introduced in Section 3).

Definition 2 ***(2-Party Weak Adaptor Signatures (2PWAS))**. This definition refers to 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 $\mathsf{aSIG}_2 = (\Gamma_{\mathsf{pSign}}, \mathsf{pVerf}, \mathsf{Adapt}, \mathsf{Ext})$. Below, we present the algorithms that differ from previous definitions.*

- $(\widehat{\sigma}, Y_i, \pi_i) \leftarrow \Gamma_{\mathsf{pSign}_{\langle sk_i, y_i; sk_{1-i}\rangle}}(pk_0, pk_1, m)$. *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 public key pk, 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.*

Our Construction of 2PWAS. 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 $f_{\mathsf{bnd}} : \mathbb{D}_{\mathsf{s}} \times \mathbb{D}_{\mathsf{w}} \to \mathbb{D}_{\mathsf{c}}$ 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 signature $s \in \mathbb{D}_{\mathsf{s}}$ 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 $(Y; y) \in \mathsf{R}$.

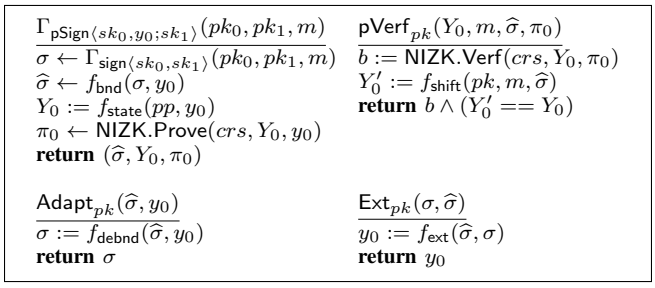

Figure 3: $\mathsf{aSIG}_2^{\mathsf{G}}$: Generic 2-party adaptor signatures.

- For verification, we define a shifting function $f_{\mathsf{shift}} : \mathbb{D}_{\mathsf{pk}} \times \mathbb{D}_{\mathsf{m}} \times \mathbb{D}_{\mathsf{ps}} \rightarrow \mathbb{D}_{\mathsf{state}}$ 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 $f_{\mathsf{debnd}} : \mathbb{D}_{\mathsf{c}} \times \mathbb{D}_{\mathsf{w}} \rightarrow \mathbb{D}_{\mathsf{s}}$ 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 signature $s \in \mathbb{D}_{\mathsf{s}}$.
- For extracting witness, we define an extraction function $f_{\mathsf{ext}} : \mathbb{D}_{\mathsf{c}} \times \mathbb{D}_{\mathsf{s}} \rightarrow \mathbb{D}_{\mathsf{w}}$ that inputs a ciphertext $c \in \mathbb{D}_{\mathsf{c}}$ together with a signature $s \in \mathbb{D}_{\mathsf{s}}$, 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 NIZK via NIZK.Setup.

For Π_{AS} to be a 2PWAS 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. Formally, for a security parameter λ, any $\mathcal{PPT}$ adversary $\mathcal{A}$, $\forall pp \in \mathbb{D}_{\mathsf{pp}}$, $\forall \sigma \in \mathbb{D}_{\mathsf{s}}$, $\forall y \in \mathbb{D}_{\mathsf{w}}$, we have

$$\Pr\{\{\sigma \vee y\} \leftarrow \mathcal{A}(pp, \widehat{\sigma}) : \widehat{\sigma} := f_{\mathsf{bnd}}(\sigma, y)\} \leq \mathsf{negl}(\lambda) \quad (1)$$

The *consistency* property is twofold: 1) the pre-signature can be recovered to the signature with the same witness y, implying the pre-signature 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, $\forall pk, \forall m \in \mathbb{D}_{\mathsf{m}}, \forall \sigma \in \mathbb{D}_{\mathsf{s}}, \forall y \in \mathbb{D}_{\mathsf{w}}$, the following two equations must hold.

$$\sigma := f_{\mathsf{debnd}}(f_{\mathsf{bnd}}(\sigma, y), y), \quad (2)$$

$$f_{\mathsf{state}}(pp, y) = f_{\mathsf{shift}}(pk, m, f_{\mathsf{bnd}}(\sigma, y)). \quad (3)$$

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

$$y := f_{\mathsf{ext}}(f_{\mathsf{bnd}}(\sigma, y), \sigma). \quad (4)$$

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 relation hardness, which resolves the aforementioned impossibility. Below, we demonstrate that the generic construction in Figure 3 constitutes a 2PWAS scheme.

Theorem 2 *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, 3, and 4,* R *is a hard relation, and* $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verf})$ *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, existential unforgeability, pre-signature adaptability, and witness extractability of $\mathsf{aSIG}_2^{\mathsf{G}}$, which are shown as several lemmas in Appendix D.

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 for BLS, Schnorr, and ECDSA signatures. Detailed instantiations can be found in Appendix E, where we also briefly review the ECDSA-based 2PWAS scheme [30] for a clearer subsequent comparison.

Comparison and Optimization via Offline / Online. Intuitively, compared to the previous 2PWAS construction, our proposal involves only one NIZK proof, and thus improves the efficiency. With regard to optimization, the computation of $Y_0 := f_{\mathsf{state}}(pp, y_0)$ and $\pi_0 \leftarrow \mathsf{NIZK.Prove}(crs, Y_0, y_0)$ 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 [34]. 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.

OVTS. We further present our OVTS for efficiently generating delayed transactions. A brief overview of OVTS is as follows. 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 pk_o to encrypt the signature and obtains the ciphertext c. Next, he calculates a time-lock puzzle z with timelock **T** for the one-time secret key sk_o. This allows anyone to recover sk_o by solving the puzzle after the timelock **T**, and subsequently use sk_o to obtain the signature σ from the ciphertext c. Finally, the committer generates a NIZK proof for the verifiability before time **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 **T**.

We next formalize our OVTS, where we denote Digital Signatures as (DS.Setup, DS.KG, DS.Sign, DS.Verf), Public-Key Encryption as (PKE.Setup, PKE.KG, PKE.Enc, PKE.Dec), TLPs as (TLP.Setup, TLP.PGen, TLP.PSolve), and NIZK as (NIZK.Setup, NIZK.Prove, NIZK.Verf).

- OVTS.Setup. This setup algorithm inputs a security parameter λ, and invokes DS.Setup, PKE.Setup, TLP.Setup, and NIZK.Setup to obtain pp_{ds}, pp_{pke}, pp_{tlp}, and crs_{zk}. It outputs the public parameter $pp_{ovts} := (pp_{ds}, pp_{pke}, pp_{tlp}, crs_{zk})$.
- OVTS.Commit-and-Prove. This commit-and-prove algorithm inputs 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

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

  and $x_{ovts} := (pp_{ovts}, c, z, pk_s, pk_o)$, $w_{ovts} := (\sigma, sk_o, r)$. Finally, it outputs the commitment $c_{ovts} := (x_{ovts}, \pi)$.
- OVTS.Verf. This verification algorithm inputs the public parameter pp_{ovts} and the commitment c_{ovts}. It parses $c_{ovts} := (x_{ovts}, \pi)$ and invokes $b := \mathsf{NIZK.Verf}(x_{ovts}, \pi)$. It outputs b, where $b = 1$ means c_{ovts} is valid, and $b = 0$ invalid.
- OVTS.Open. This open algorithm inputs the public parameter pp_{ovts} and the commitment c_{ovts}, and it outputs the committed signature σ and randomness r adopted in generating c_{ovts}.
- OVTS.ForceOpen. This force-open algorithm inputs the public parameter pp_{ovts} and the 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 F.

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

Theorem 4 *Let* NIZK *be NIZK proofs for* $\mathcal{L}_{ovts}$, PKE *be an IND-CPA encryption scheme, and* TLP *be a time-lock puzzle with perfect completeness. Then, our OVTS 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 TLPs. The TLP 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 incurs a constant computation cost, involving key generation in public key encryption, puzzle generation in TLP, and NIZK.Prove, all of which are independent of 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 [40]. Therefore, we suggest to instantiate NIZK as Σ-protocol [38] or Bulletproofs [41] 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 [42], and Homomorphic Time-Lock Puzzles (HTLP [43]). We apply the Σ-protocol and Fiat-Shamir transformation [39] to realize the NIZK proof. Detailed information could be referred to Appendix G.

5. Performance Analysis

We implemented AMBiPay building blocks, including 2PWAS schemes, the OVTS scheme, and transaction flow, followed by efficiency evaluation via benchmarks.

5.1. Implementation

To evaluate the three Π_{AS} schemes, we instantiated the underlying 2-party signature scheme as BLS [32], ECDSA [34], and Schnorr [33] (referred to as BLS-Π_{AS}, ECDSA-Π_{AS}, Schnorr-Π_{AS} hereafter). Our evaluation included a comparison with the ECDSA-based 2PWAS (ECDSA*-Π_{AS}) proposed in [30]. In our comparison of OVTS with VTS, we employed ECDSA signature instantiation and utilized the implementations of VTS and HTLP from [14] and [43], respectively. We will refer to them as OVTS-ECDSA and VTS-ECDSA subsequently. Notice that we did not compare the setup algorithm as it can be precomputed and shared across various instances of AMBiPay. Similarly, we omitted the ForceOpen algorithm as its running time is predetermined by the time hardness **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. 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 OS (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 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 used in [34] 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.

To test the feasibility of AMBiPay, we implemented it on Sepolia, an Ethereum testnet that closely resembles the Ethereum network [2]. 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 plugin of Google Chrome to connect to Sepolia and deploy transactions while measuring overhead.

5.2. Benchmarks

2PWAS Schemes. From Table 2, BLS-Π_{AS} incurs higher time costs than the other three, as it requires time-consuming bilinear pairing and hash-to-point operations. Fortunately, the online computation for BLS-Π_{AS} in both Γ_{Sign} and Γ_{pSign} is less than 105 ms. This is 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} takes not more than 45 ms, and that in ECDSA-Π_{AS} (except Γ_{Sign}-Offline and Γ_{pSign}-Offline) requires less than 75 ms. In our ECDSA-Π_{AS}, Γ_{pSign} takes 21.5495 ms, and pVerf takes 43.9521 ms. In ECDSA*-Π_{AS} [30], Γ_{pSign} and pVerf involve 54.3438 ms and 76.9452 ms, respectively. Thus, our ECDSA-Π_{AS} achieves 2.52× and 1.75× efficiency improvements in terms of Γ_{pSign} and pVerf, respectively, compared to ECDSA*-Π_{AS} [30]. The adapting and extracting operations in four schemes take less than 1 ms. These results underscore the versatility, as demonstrated by the support for unique signatures like BLS, and efficiency, characterized by more efficient Γ_{pSign} and pVerf, of our proposal in comparison to existing schemes [20], [24], [25], [30].

TABLE 2: Time costs of four Π_{AS} schemes (in ms). We omit the time cost of Γ_{Sign} required in Γ_{pSign} since all the compared schemes involve Γ_{Sign} in Γ_{pSign}.

Algorithm	ECDSA* [30]	ECDSA	BLS	Schnorr
Γ_{Sign}-Offline	210.883	210.546	0	42.833
Γ_{Sign}-Online	70.8174	70.7785	104.899	0.9517
Verf	21.0938	21.0541	348.866	21.2656
Γ_{pSign}-Offline	43.0602	21.5166	408.604	21.2684
Γ_{pSign}-Online	11.2836	0.0329	0.1952	0.0058
pVerf	76.9452	43.9521	633.531	42.6271
Adapt	0.0143	0.0127	0.2117	0.0065
Ext	0.0142	0.0335	0.2103	0.0067

We further compare the computation and communication costs of BLS-Π_{AS}, Schnorr-Π_{AS}, ECDSA-Π_{AS}, and ECDSA*-Π_{AS} [30] with a focus on Γ_{Sign} and Γ_{pSign} in the context of AMBiPay. In the left sub-figure of Figure 4, Schnorr-Π_{AS} stands out for its lower time costs in both Sign and pSign, with a substantial portion of operations (97.83% for Sign and 98.53% for pSign) executed offline. While Schnorr-Π_{AS} demonstrates efficiency, we note that the online costs of ECDSA-Π_{AS} and ECDSA*-Π_{AS} are still practical for deploying AMBiPay. Regarding BLS-Π_{AS}, potential optimizations, such as choosing a more compatible pairing-friendly curve, could enhance its support for AMBiPay.

Both Sign and pSign maintain acceptable communication costs (less than 0.55 KB) in all schemes, except for pSign in BLS-Π_{AS}, which incurs a higher cost of approximately 2.22 KB. Notably, compared to ECDSA*-Π_{AS} [30], our ECDSA-Π_{AS} is approximately 1.47× more efficient in communication costs for pSign (0.19 KB versus 0.28 KB). This positions our proposal as a practical choice for integration into the AMBiPay system, which is further supported by the subsequent AMBiPay overhead analysis.

OVTS Schemes. 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 $t = n/2$. From the comparison results shown in the center and right sub-figures of Figure 4, 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 hence its efficiency is independent of n. This also holds for OVTS instantiated from other digital signatures such as BLS and Schnorr. Thus, OVTS shows a significant practical advantage.

Deployment of Transactions. We now present the transactions in AMBiPay, and the details on transaction latency and sizes will be given later. To show the backward compatibility of AMBiPay with existing account-based currencies, we provide a pointer to those 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 [44] and tx_B [45] to lock balance and collateral in Ch_{AB}. Next, we examine A's and B's state transactions $tx_{Pay,i}^{A}$ [46] and $tx_{RPay,i}^{B}$ [47] (when A takes the initiative), and these transactions are symmetric if for B to take the initiative. There are two ways for A to claim her coins, when she posts $tx_{Pay,i}^{A}$ and $tx_{Pay,i}^{A*}$ [48] on Sepolia. If B refunds his balance and collateral via posting $tx_{RPay,i}^{B}$ and $tx_{RPay,i}^{B*}$ [49], then A can claim her funds via posting $tx_{FPay,i}^{A}$ [50] right away. Otherwise, if the timelock expires, A can refund her coins with $tx_{D,i}^{A}$. Once A posts an old state, B can post $tx_{Pnsh,i}^{B}$ 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 models and UTXO models, and the strong constraint via 2PWAS, cause 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

2. We initially implemented AMBiPay on the Stellar testnet. However, the accounts and transactions on this testnet are periodically deleted. For more reliable access, we chose to implement AMBiPay on the Ethereum network. It is important to note that our implementation on Ethereum operates seamlessly without relying on the smart contract functionality.

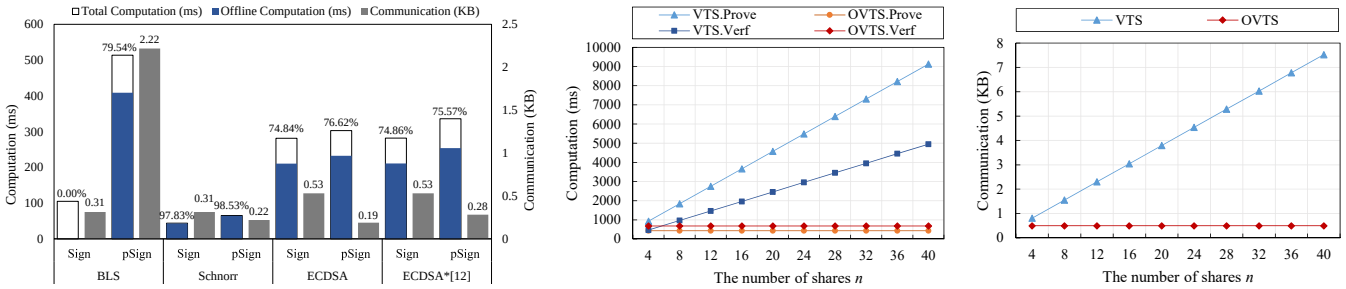

Figure 4: *Left sub-figure*: showing cost comparison of the aforementioned four 2PWAS schemes. The white histogram represents total computation cost, the blue histogram signifies offline computation cost, the gray histogram indicates communication cost and the percentage reflects the proportion of offline computation cost in the total. *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.

	Sleepy Channels		AMBiPay	
	Operations	**KB**	**Operations**	**KB**
txs off-chain				
Create	$2(tx^{A}_{Pay,i} + tx^{A,B}_{FPay,i} + tx^{A*}_{FPay,i} + tx^{A,A}_{Fpay,i})$	1.9785	$2(tx^{A}_{Pay,i} + tx^{A*}_{Pay,i} + tx^{A}_{R,i} + tx^{B}_{RPay,i} + tx^{B*}_{RPay,i} + tx^{A}_{Fpay,i})$	1.9687
Update	$2(tx^{A}_{Pay,i} + tx^{A,B}_{FPay,i} + tx^{A*}_{FPay,i} + tx^{A,A}_{Fpay,i} + tx^{A}_{Pnsh,i})$	2.3515	$2(tx^{A}_{Pay,i} + tx^{A*}_{Pay,i} + tx^{A}_{R,i} + tx^{A}_{Fpay,i} + tx^{B}_{RPay,i} + tx^{B*}_{RPay,i} + tx^{B}_{Pnsh,i})$	2.2031
txs on-chain				
Create	tx_F	0.3301	$tx_A + tx_B$	0.2353
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

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 on-chain part because block generation time varies widely across different currencies, ranging from 10 minutes in Bitcoin to 15 seconds in Ethereum on average. The off-chain 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 communication and off-chain computation costs of AMBiPay. In the creation phase, the two parties 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). The closing phase happens on-chain, involving three possible situations: optimistic, slow, and fast. The optimistic situation is when they close the channel honestly (0.3046 KB, 2 txs) [3], 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 shows that AMBiPay is comparable to Sleepy Channels.

6. Conclusion

We present a novel trustless and scriptless bidirectional payment channels (bi-PC) protocol tailored for the account model. Our solution addresses the uncertainty and weak constraint challenges in extending UTXO-based solutions to the account model. The protocol inherits the advantages of existing bi-PC protocols for UTXO and eliminates the need for complex scripts, including smart contracts. Notably, we redefine the security definitions and propose a more efficient generic construction of 2-party weak adaptor signatures for both unique and common signatures. Additionally, we introduce an optimized verifiable timed signatures construction to enhance efficiency. Our evaluation shows that our proposal is efficient and applicable to account-based currencies, and resolves the open problem raised by Aumayr et al. in CCS'22 regarding the applicability of Sleepy Channels to such systems.

3. When both parties honestly negotiate the allocation of coins in the channel, only two transactions are needed: one to close the channel and one for the counterparty to claim their balance. This efficiency is due to the single-input-single-output transaction form in the account model.

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] 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

[9] 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

[10] 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

[11] L. McCulloch, "Starlight: A layer 2 payment channel protocol for stellar." 2022. [Online]. Available: https://github.com/stellar/starlight

[12] 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

[13] 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

[14] 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

[15] 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

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

[17] 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

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

[19] 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, 2023, pp. 2462–2480. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.00116

[20] 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

[21] 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

[22] 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

[23] 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

[24] 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://eprint.iacr.org/2018/472

[25] 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

[26] V. Madathil, S. A. K. Thyagarajan, D. Vasilopoulos, L. Fournier, G. Malavolta, and P. Moreno-Sanchez, "Cryptographic oracle-based conditional payments," in *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/cryptographic-oracle-based-conditional-payments/

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

[28] W. Dai, T. Okamoto, and G. Yamamoto, "Stronger security and generic constructions for adaptor signatures," in *Progress in Cryptology - INDOCRYPT 2022 - 23rd International Conference on Cryptology in India, Kolkata, India, December 11-14, 2022, Proceedings*, ser. Lecture Notes in Computer Science, T. Isobe and S. Sarkar, Eds., vol. 13774. Springer, 2022, pp. 52–77. [Online]. Available: https://doi.org/10.1007/978-3-031-22912-1_3

[29] P. Gerhart, D. Schröder, P. Soni, and S. A. K. Thyagarajan, "Foundations of adaptor signatures," in *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part II*, ser. Lecture Notes in Computer Science, M. Joye and G. Leander, Eds., vol. 14652. Springer, 2024, pp. 161–189. [Online]. Available: https://doi.org/10.1007/978-3-031-58723-8_6

[30] 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

[31] B. Edgington, "Upgrading ethereum," 2023. [Online]. Available: https://tinyurl.com/2p969w9b

[32] 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

[33] 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

[34] 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

[35] 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

[36] 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

[37] 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

[38] 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

[39] 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

[40] 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

[41] 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. [Online]. Available: https://doi.org/10.1109/SP.2018.00020

[42] 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

[43] 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

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

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

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

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

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

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

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

[51] 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

[52] 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

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 [13]. 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-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 $\mathbb{L}(\Delta, \Gamma, \mathcal{V})$, Δ represents an initialized upper bound (after which valid transactions are chained to $\mathbb{L}$). Γ 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 [13], 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, $\Psi(\mathsf{id}) \rightarrow (sts, addr)$ 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 $\mathcal{F}_{AM}(\tau_p, k)^{\mathbb{L}(\Delta,\Gamma,\mathcal{V})}$ (abbreviated as $\mathcal{F}_{AM}$) is shown in Figure 5, and the explanation of security and efficiency is provided as follows.

Create. When $\mathcal{F}_{AM}$ receives two creating requests $(\texttt{CREATE}, \chi, tid_A)$ and $(\texttt{CREATE}, \chi, tid_B)$ 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 $(\texttt{Update}, \mathsf{id}, \overrightarrow{\theta}, \tau_{stp})$, 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 $(\texttt{CLOSE}, \mathsf{id})$, 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, party B does not lose coins to party A in this scenario.

The UC framework was introduced in Appendix A, 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 $(\texttt{createInfo}, \mathsf{id}, pk_A, pkey^A_{set}) \xrightarrow{\tau_0} B$.
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 $tx_F^{A,B} := (tid_A, tid_B)$ 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 $tx_{set,0} \leftarrow \mathsf{GTxs}(addr_{set}, pkey^A_{set}, pkey^B_{set}, c, v_{A,0}, v_{B,0})$.

Create: When receiving $(\texttt{CREATE}, \chi, pk_A) \xleftarrow{\tau_0} A$, distinguish:

Both approved: In the case a message $(\texttt{CREATE}, \chi, pk_B) \xleftarrow{\tau} B$ has already been received, where $\tau_0 - \tau \leq \tau_p$: 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 $\tau_1 \leq \tau + \Delta + \tau_p$, 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 $(\texttt{CREATE}, id) \xleftarrow{\tau \leq \tau_0 + \tau_p} B$ (then, the case of "Both approved" happens). Halt if such a message is not detected.

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

1) In round $\tau_1 \leq \tau_0 + \tau_p$, 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 $(\texttt{SETUP-OK}, \mathsf{id}) \xleftarrow{\tau_2 \leq \tau_1 + \tau_{stp}} A$, then $(\texttt{SETUP-OK}, \mathsf{id}) \xhookrightarrow{\tau_3 \leq \tau_2 + \tau_p} B$. Otherwise halt.
3) If $(\texttt{UPDATE-OK}, \mathsf{id}) \xleftarrow{\tau_3} B$, then (if B is honest or under the control of $\mathcal{S}$) send $(\texttt{UPDATE-OK}, \mathsf{id}) \xleftarrow{\tau_4 \leq \tau_3 + \tau_p} A$. 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 $(\texttt{REVOKE}, \mathsf{id}) \xleftarrow{\tau_4} A$, send $(\texttt{REVOKE-REQ}, \mathsf{id}) \xhookrightarrow{\tau_5 \leq \tau_4 + \tau_p} B$. Otherwise let $\Psi(\mathsf{id}) := (\{\chi, \chi'\}, \mathsf{Ch}_{AB})$, invoke $\texttt{ForceClose}(\mathsf{id})$ and halt.
5) If $(\texttt{REVOKE}, \mathsf{id}) \xleftarrow{\tau_5} B$, $\Psi(\mathsf{id}) := (\{\chi'\}, \mathsf{Ch}_{AB})$, transmit $(\texttt{UPDATED}, \mathsf{id}, \overrightarrow{\theta}) \xhookrightarrow{\tau_6 \leq \tau_5 + \tau_p} \chi.\mathsf{users}$ and halt (*accept*). Otherwise let $\Psi(\mathsf{id}) := (\{\chi, \chi'\}, \mathsf{Ch}_{AB})$, invoke $\texttt{ForceClose}(\mathsf{id})$ and halt.

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

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

- If $tx_C := (tid'_A, tid'_B)$ appears on $\mathbb{L}$ in round $\tau_1 \leq \tau_0 + \Delta$, 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 $(\texttt{CLOSE}, \mathsf{id}) \xleftarrow{\tau \leq \tau_0 + \tau_p} B$ (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 $\tau_1 \leq \tau_0 + \Delta$. Then in round $\tau_2 \leq \tau_1 + \Delta$, 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 $(\texttt{PUNISHED}, \mathsf{id}) \xhookrightarrow{\tau_2} \mathsf{B}$ 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 $\tau_1 \leq \tau_0 + \Delta$ 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 $(\texttt{CLOSED}, \mathsf{id}) \xhookrightarrow{\tau_2 \leq \tau_1 + \Delta} \chi.\mathsf{users}$. Otherwise, transaction $tx_D^{A} := tx(pk_R, pk_A, \chi.\mathsf{st.bal}(A), 0)$ is chained on $\mathbb{L}$ in round $\tau_3 \leq \chi.T + \Delta$. 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 5: Ideal Functionality $\mathcal{F}_{AM}(\tau_p, k)^{\mathbb{L}(\Delta, \Gamma, \mathcal{V})}$ of bi-PC for account model

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 $(\texttt{createFund}, \mathsf{id}, \sigma_{tid_A}) \xhookrightarrow{\tau_0+1+\tau_g+\tau_s} A$.
10) If $(\texttt{createFund}, \mathsf{id}, \sigma_{tid_B}) \xhookleftarrow{\tau_0+2+\tau_g+\tau_s} B$, post $(tx^{A,B}_F, \{\sigma_{tid_A}, \sigma_{tid_B}\})$ to $\mathbb{L}$.
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 $(\texttt{UPDATE}, \mathsf{id}, \overrightarrow{\theta}, \tau_{stp}) \xhookleftarrow{\tau_0} \mathcal{E}$

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

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

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 $(\texttt{updateInfo}, \mathsf{id}) \xhookleftarrow{\tau_0+2} B$

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 $(\texttt{SETUP-OK}, \mathsf{id}) \xhookleftarrow{\tau_1 \leq \tau_0+2+\tau_{stp}} \mathcal{E}$, 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 $(\texttt{updateCom}, \mathsf{id}) \xhookleftarrow{\tau_1 \leq \tau_0+2+\tau_{stp}} A$

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 $(\texttt{UPDATE-OK}, \mathsf{id}) \xhookrightarrow{\tau_1+1+\tau_s} \mathcal{E}$. Otherwise, invoke $\texttt{ForceClose}(\mathsf{id})$ and remain idle.
2) If not $(\texttt{REVOKE}, \mathsf{id}) \xhookleftarrow{\tau_1+1+\tau_s} \mathcal{E}$, 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) $(\texttt{REVOKE}, \mathsf{id}, \sigma^B_{Pnsh,i-1}) \xhookrightarrow{\tau_1+1+\tau_s+\tau_r} B$.

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 $(\texttt{REVOKE}, \mathsf{id}, \sigma^B_{Pnsh,i-1}) \xhookleftarrow{\tau_1+1+\tau_s+\tau_r} A$, 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) $\Psi^B(id) := (tx^{A,B}_F, tx_{set,i}, sig^B_{set,i}, addr_{set}, pkey^A_{set}, pkey^B_{set})$.
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) $\Psi^A(id) := (tx^{A,B}_F, tx_{set,i}, sig^A_{set,i}, addr_{set}, pkey^A_{set}, pkey^B_{set})$.
5) Send $(\texttt{UPDATED}, \mathsf{id}) \xhookleftarrow{\tau_1+3+\tau_s+2\tau_r} \mathcal{E}$.

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 $\Psi^A(\mathsf{id})$.
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 $\mathtt{ForceClose}(\mathsf{id})$.
6) If tx_c is chained on $\mathbb{L}$ in round $\tau_1 \leq \tau_0 + \tau_r + \Delta$, let $\Theta^A(id) := \bot$, $\Psi^A(\mathsf{id}) := \bot$ and send $(\mathtt{CLOSED}, \mathsf{id}) \overset{\tau_2}{\hookrightarrow} \mathcal{E}$.

Punish

Party A receives $\mathtt{Punish} \overset{\tau_0}{\hookleftarrow} \mathcal{E}$

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

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 $\Theta^A(\mathsf{id}) := \bot, \Psi^A(\mathsf{id}) := \bot$ and output $(\mathtt{PUNISHED}, \mathsf{id}) \overset{\tau_1}{\hookrightarrow} \mathcal{E}$.

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 $\Psi^A(\mathsf{id})$ 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 $(\mathtt{CLOSED}, id) \overset{\tau_3 \leq \tau_2 + \Delta}{\hookrightarrow} \mathcal{E}$. Otherwise, post $(tx^A_{D,j}, \sigma^A_{D,j})$ after $\mathbf{T}_d$ and send $(\mathtt{CLOSED}, \mathsf{id}) \overset{\tau_4 \leq \mathbf{T}_d + \Delta}{\hookrightarrow} \mathcal{E}$.
5) Set $\Psi^P(\mathsf{id}) := \bot$, $\Theta^P(\mathsf{id}) := \bot$.

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

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 $tx_{set} := \{tx^A_{Pay,i}, tx^B_{Pay,i}, tx^A_{RPay,i}, tx^B_{RPay,i}, tx^{A*}_{Pay,i}, tx^{B*}_{Pay,i}, tx^A_R, tx^B_R, tx^A_{Pnsh,i}, tx^B_{Pnsh,i}, tx^A_{D,i}, tx^B_{D,i}, tx^A_{FPay,i}, tx^B_{FPay,i}, tx^{A*}_{RPay,i}, tx^{B*}_{RPay,i}\}$.

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

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 TTP, the real-world protocol described in Section 4 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 [51].

We first define Π''_{B} as the protocol Π'''_{B} presented in Section 4, 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 $\Pi\mathsf{B}'''$, 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}.

1. UC Simulator

We now present the pseudocode of a simulator for the ideal-world model of Π_{B} in Appendix A. 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 (CREATE, χ, pk_A) $\overset{\tau_0}{\hookrightarrow} \mathcal{F}_{AM}$, but B does not send (CREATE, χ, pk_B) $\overset{\tau}{\hookrightarrow} \mathcal{F}_{AM}$ where $|\tau_0 - \tau| \leq \tau_p$, then there are the following two possible cases.

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

Follow the steps as described below:

1) Let id := χ.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 (createInfo, id, $pk_A, pkey^A_{set}$) $\overset{\tau_0}{\hookrightarrow} B$.
2) If (createInfo, id, $pk_B, pkey^B_{set}$) $\overset{\tau_0+1}{\hookleftarrow} B$, 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, $addr_{set}$:= {Ch_{AB}, FCh_A,FCh_B, $\mathsf{SleepyCh}_A$, $\mathsf{SleepyCh}_B$, Exit_A, Exit_B} which requires τ_g rounds. Abort if a failure occurs.

4) Generate $tx_F^{A,B} := (tid_A, tid_B)$ 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 $tx_{set,0} \leftarrow \mathsf{GTxs}(addr_{set}, pkey_{set}^A, pkey_{set}^B, c, v_{A,0}, v_{B,0})$.
6) Let $sig_{set,0}^A \leftarrow \mathsf{SignTxs}^A(tx_{set,0}, addr_{set}, pkey_{set}^A \cup pkey_{set}^B)$.
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 $(tx_F^{A,B}, \{\sigma_{tid_A}, \sigma_{tid_B}\})$ to $\mathbb{L}$.
9) If $tx_F^{A,B}$ is chained on $\mathbb{L}$ in round $\tau_1 \leq \tau_0 + 2 + \tau_g + \tau_s + \Delta$, store $\Psi^A(\mathsf{id}) := (tx_F^{A,B}, tx_{set,0}, sig_{set,0}^A, addr_{set}, pkey_{set}^A, pkey_{set}^B)$ 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 $(tx_F^{A,B}, tx_{set,i-1}, sig_{set,i-1}^B, addr_{set}, pkey_{set}^A, pkey_{set}^B) := \Psi^B(\mathsf{id})$.
4) Retrieve $v_{A,i}$ and $v_{B,i}$ from $\overrightarrow{\theta}$, and c from $tx_F^{A,B}$.
5) Let $tx_{set,i} \leftarrow \mathsf{GTxs}(addr_{set}, pkey_{set}^A, pkey_{set}^B, c, v_{A,i}, v_{B,i})$.
6) Let $\overrightarrow{tid} := (tx_{Pay,i}^A.\mathsf{id}, tx_{Pay,i}^B.\mathsf{id}, tx_{RPay,i}^A.\mathsf{id}, tx_{RPay,i}^B.\mathsf{id})$ be a tuple of identities of transactions $tx_{Pay,i}^A$, $tx_{Pay,i}^B$, $tx_{RPay,i}^A$, and $tx_{RPay,i}^B$. 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 $\mathsf{SignTxs}^A(tx_{set,i}, addr_{set}, pkey_{set}^A \cup pkey_{set}^B)$, send $(\texttt{UPDATE-OK}, \mathsf{id}) \xhookrightarrow{\tau_1+1} \mathcal{F}_{AM}$ via acting as B.
10) $\mathsf{SignTxs}^A(tx_{set,i}, addr_{set}, pkey_{set}^A \cup pkey_{set}^B)$.
11) If $sig_{set,i}^A$ 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, $\sigma_{Pnsh,i-1}^B$ on the punishing transaction $tx_{Pnsh,i-1}^B$. Party A obtains $tx_{Pnsh,i-1}^B$ as output. This requires τ_r rounds. If a failure occurs, invoke $\mathsf{ForceClose}^A(\mathsf{id})$.
14) Send $(\texttt{REVOKE}, \mathsf{id}, \sigma_{Pnsh,i-1}^B) \xhookrightarrow{\tau_1+1+\tau_s+\tau_r} B$.
15) If B executes $\mathcal{F}_{\mathsf{Sign}}$ to generate a signature of $tx_{Pnsh,i-1}^B$ in round $\tau_1 + 2 + \tau_s + \tau_r$, send $(\texttt{REVOKE}, \mathsf{id}) \xhookrightarrow{\tau_1+2+\tau_s+\tau_r} \mathcal{F}_{AM}$ by acting as B and also acts as A to participate in the signature generation.
16) If $(\texttt{REVOKE}, \mathsf{id}, \sigma_{Pnsh,i-1}^B) \xhookleftarrow{\tau_1+3+\tau_s+2\tau_r} B$ and $\sigma_{Pnsh,i-1}^B$ is valid, proceed to the next step. Else, invoke $\mathsf{ForceClose}^A(\mathsf{id})$.
17) $\Theta^A(id) := \Theta^A \cup \{(tx_{set,i-1}, sig_{set,i-1}^A, \sigma_{Pnsh,i-1}^B)\}$.
18) $\Psi^A(id) := (tx_F^{A,B}, tx_{set,i}, sig_{set,i}^A, addr_{set}, pkey_{set}^A, pkey_{set}^B)$.

Case of honest B and corrupted A

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

1) Receive $(\texttt{updateReq}, \mathsf{id}, \overrightarrow{\theta}, \tau_{stp}) \xhookleftarrow{\tau_0} A$, take the following actions:
2) Retrieve $(tx_F^{A,B}, tx_{set,i-1}, sig_{set,i-1}^B, addr_{set}, pkey_{set}^A, pkey_{set}^B) := \Psi^B(\mathsf{id})$.
3) Retrieve $v_{A,i}$ and $v_{B,i}$ from $\overrightarrow{\theta}$, and c from $tx_F^{A,B}$.
4) Let $tx_{set,i} \leftarrow \mathsf{GTxs}(addr_{set}, pkey_{set}^A, pkey_{set}^B, c, v_{A,i}, v_{B,i})$.
5) Let $\overrightarrow{tid} := (tx_{Pay,i}^A.\mathsf{id}, tx_{Pay,i}^B.\mathsf{id}, tx_{RPay,i}^A.\mathsf{id}, tx_{RPay,i}^B.\mathsf{id})$ be a tuple of identities of transactions $tx_{Pay,i}^A$, $tx_{Pay,i}^B$, $tx_{RPay,i}^A$, and $tx_{RPay,i}^B$. Notify $\mathcal{F}_{AM}$ of $\overrightarrow{tid}$.
6) Send $(\texttt{updateInfo}, \mathsf{id}) \xhookrightarrow{\tau_0} A$.
7) When A sends $(\texttt{updateCom}, \mathsf{id}) \xhookrightarrow{\tau_0+1+\tau_{stp}} B$, send $(\texttt{SETUP-OK}, \mathsf{id}) \xhookrightarrow{\tau_1} \mathcal{F}_{AM}$ by acting as A.
8) $(\texttt{updateCom}, \mathsf{id}) \xhookleftarrow{\tau_1 \leq \tau_0+2+\tau_{stp}} A$.
9) If B transmits $(\texttt{UPDATE-OK}, \mathsf{id}) \xhookrightarrow{\tau_1} \mathcal{F}_{AM}$, invoke $\mathsf{SignTxs}^A(tx_{set,i}, addr_{set}, pkey_{set}^A \cup pkey_{set}^B)$.
10) If $sig_{set,i}^B$ 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 $\sigma_{Pnsh,i-1}^B$ in round $\tau_1 + \tau_s$, transmit $(\texttt{REVOKE}, \mathsf{id}) \xhookrightarrow{\tau_1+\tau_s} \mathcal{F}_{AM}$ by acting as A. Also, act as B to participate in the signature generation.
12) If receive $(\texttt{REVOKE}, \mathsf{id}, \sigma_{Pnsh,i-1}^B) \xhookleftarrow{\tau_1+1+\tau_s+\tau_r} A$, continue. Else, invoke $\mathsf{ForceClose}^B(\mathsf{id})$ and remain idle.
13) If B fails to send $(\texttt{REVOKE}, \mathsf{id}) \xhookrightarrow{\tau_1+1+\tau_s+\tau_r} \mathcal{F}_{AM}$, remain idle.
14) $\mathcal{S}$, acting as B, executes $\mathcal{F}_{\mathsf{Sign}}$ with A to obtain the signature $\sigma_{Pnsh,i-1}^A$ on $tx_{Pnsh,i-1}^A$. Party B finally obtains $\sigma_{Pnsh,i-1}^A$ after τ_r. If an error occurs, invoke

ForceClose^B(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

Case of honest A and corrupted B

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 Ψ^A(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(id).
6) If tx_c is chained on $\mathbb{L}$ in round $\tau_1 \le \tau_0 + \tau_r + \Delta$, let $\Theta^A(id) := \bot, \Psi^A(id) := \bot$.

Simulator for Punish

Case of honest A and corrupted B

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

1) Derive $\{(tx_{set,i}, sig^A_{set,i}, \sigma^B_{Pnsh,i})\}_{i\in m} := \Theta^P$(id) and retrieve χ from Ψ^A(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 Θ^A(id) $:= \bot, \Psi^A$(id) $:= \bot$.

Simulator for ForceClose(id)

Denote τ_0 as the present round.

1) Extract $(tx^{A,B}_F, tx_{set,0}, sig^A_{set,0}, addr_{set}, pkey^A_{set}, pkey^B_{set})$ from Ψ^A(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) Post $(tx^A_{Pay,j}, \sigma^A_{Pay,j})$ and $(tx^{A*}_{Pay,j}, \sigma^{A*}_{Pay,j})$ on $\mathbb{L}$.
3) Denote $\tau_2 \le \tau_1 + \Delta$ as the round when $tx^A_{Pay,j}$ and $tx^{A*}_{Pay,j}$ are accepted by $\mathbb{L}$.
4) If tx^B_{RPay} and $tx^{B^*}_{RPay}$ appear on $\mathbb{L}$ at or after round $\tau_2 \le \tau_1 + \Delta$ and before $\mathbf{T}_d$, post $(tx^A_{FPay,j}, \sigma^A_{FPay,j})$. Otherwise, post $(tx^A_{D,j}, \sigma^A_{D,j})$ after $\mathbf{T}_d$. Let $\Gamma^P(id) := \bot$, $\Theta^P(id) := \bot$.

2. Simulation Proof

To demonstrate that the AMBiPay protocol UC-realizes $\mathcal{F}_{AM}$, we need to prove that $EXEC_{\Pi_B,\mathcal{A},\mathcal{E}}$ 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 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 Γ that provides EUF-CMA security, and a ledger $\mathbb{L}(\Delta, \Gamma, \mathcal{V})$ that allows for transaction verification using Γ and has the ability to enforce absolute time-locks. Γ 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 2 *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 $tx^{A,B}_F$ 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 $EXEC^{Create}_{\Pi_B,\mathcal{A},\mathcal{E}} := \{\texttt{createInfo}[\tau_0 + 1], \texttt{oSet}(a_0, \tau_0{+}1), \texttt{oSet}(a_1, \tau_0{+}1{+}\tau_g), createFund[\tau_0{+}2 + \tau_g + \tau_s], \texttt{oSet}(a_2, \tau_0 + 2 + \tau_g + \tau_s), \texttt{CREATED}[\tau_1]\}$ 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 $\tau_0{+}1$. 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 $EXEC^{Create}_{\mathcal{F}_{AM},\mathcal{S},\mathcal{E}} := \{\texttt{createInfo}[\tau_0 + 1], \texttt{oSet}(a_0, \tau_0 + 1), \texttt{oSet}(a_1, \tau_0 + 1 + \tau_g), \texttt{createFund}[\tau_0 + 2 + \tau_g + \tau_s], \texttt{oSet}(a_2, \tau_0 + 2 + \tau_g + \tau_s), \texttt{CREATED}[\tau_1]\}$.

Lemma 3 *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 ($tx^A_{Pay,j}$, $\sigma^A_{Pay,j}$) and ($tx^{A*}_{Pay,j}$, $\sigma^{A*}_{Pay,j}$) on $\mathbb{L}$" is performed by A in round τ_0, using the latest state. After the transactions $tx^A_{Pay,j}$ and $tx^{A*}_{Pay,j}$ appear on $\mathbb{L}$ in round τ_1 (where $\tau_1 \leq \tau_0 + \Delta$), there are two possible scenarios. First, the transactions $tx^B_{RPay,j}$ and $tx^{B*}_{RPay,j}$ 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 ($tx^A_{Fpay,j}$, $\sigma^A_{Fpay,j}$), 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 $tx^B_{RPay,j}$ and $tx^{B*}_{RPay,j}$ do not appear on $\mathbb{L}$ before $\mathbf{T}_d$, A publishes ($tx^A_{D,j}$, $\sigma^A_{D,j}$), 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 $EXEC^{ForceClose}_{\Pi_B,\mathcal{A},\mathcal{E}} := \{\texttt{oSet}(a_0, \tau_0), o \in \{\texttt{oSet}(a_1, \tau_2), \texttt{oSet}(a_2, \mathbf{T}_d)\}, \texttt{CLOSED}[\tau_m]\}$.

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 $tx^A_{Pay,j}$ and $tx^{A*}_{Pay,j}$ appear on $\mathbb{L}$ in round τ_1 (where $\tau_1 \leq \tau_0 + \Delta$), there are two possible scenarios. First, if the transactions $tx^B_{RPay,j}$ and $tx^{B*}_{RPay,j}$ 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 $EXEC^{ForceClose}_{\mathcal{F}_{AM},\mathcal{S},\mathcal{E}} := \{\texttt{oSet}(a_0, \tau_0), o \in \texttt{oSet}(a_1, \tau_2), \texttt{oSet}(a_2, \mathbf{T}_d), \texttt{CLOSED}[\tau_m]\}$.

Lemma 4 *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 $EXEC^{Update}_{\Pi_B,\mathcal{A},\mathcal{E}}$ is listed for clarity.

- Initiate by sending updateReq to B in τ_0.
- Transmit SETUP to $\mathcal{E}$ in $\tau_0 + 2$ 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 $EXEC^{Update}_{\Pi_B,\mathcal{A},\mathcal{E}}$.

- 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 5 *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 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 CLOSED. However, if the signature generation fails in round $\tau_2 \geq \tau_0$, A executes the action ForceClose, denoted by a_2. We denote the execution domain as either $EXEC^{Close}_{\Pi_B,\mathcal{A},\mathcal{E}}$:= $\{\mathtt{oSet}(a_0,\tau_0), \mathtt{oSet}(a_1,\tau_0 + \tau_r), \mathtt{CLOSED}[\tau_1]\}$ or $EXEC^{Close}_{\Pi_B,\mathcal{A},\mathcal{E}}$:= $\{\mathtt{oSet}(a_0,\tau_0), \mathtt{oSet}(a_2,\tau_2), \mathtt{CLOSED}[\tau_1]\}$.

Ideal World: After A obtains 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 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 CLOSED). We denote the execution domain as $EXEC^{Close}_{\mathcal{F}_{AM},\mathcal{S},\mathcal{E}}$:= $\{\mathtt{oSet}(a_0,\tau_0), \mathtt{oSet}(a_1,\tau_0 + \tau_r), \mathtt{CLOSED}[\tau_1]\}$ or $EXEC^{Close}_{\mathcal{F}_{AM},\mathcal{S},\mathcal{E}}$:= $\{\mathtt{oSet}(a_0,\tau_0), \mathtt{oSet}(a_2,\tau_2)\}$.

Lemma 6 *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 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 PUNISHED. We here denote the execution domain as $EXEC^{Punish}_{\Pi_B,\mathcal{A},\mathcal{E}}$:= $\{\mathtt{oSet}(a_0,\tau_0), \mathtt{oSet}(a_1,\tau_1), \mathtt{PUNISHED}[\tau_2]\}$.

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 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 $EXEC^{Punish}_{\mathcal{F}_{AM},\mathcal{S},\mathcal{E}}$:= $\{\mathtt{oSet}(a_0,\tau_0), \mathtt{oSet}(a_1,\tau_1), \mathtt{PUNISHED}[\tau_2]\}$.

Lemma 7 *The protocol Π_B is the UC-realization of the ideal functionality $\mathcal{F}_{AM}$.*

Proof. By applying Lemmas 2, 3, 4, 5, and 6 and using a standard hybrid argument, we can prove the theorem.

3. Redefined Security Model

Before formally defining the essential properties of aSIG_2, we first introduce four key oracles that the adversary

can query: the newly defined honest relation oracle $\mathcal{O}_H$ and corrupted relation oracle $\mathcal{O}_C$, along with the modified pre-signing oracle $\mathcal{O}_{pS}$ and signing oracle $\mathcal{O}_S$ (see Figure 6). We then present the concrete security definitions based on these oracles.

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 if, for any $\mathcal{PPT}$ adversary $\mathcal{A}$, the probability $\Pr[\mathsf{aSigForge}^b_{\mathcal{A},\mathsf{aSIG}_2}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$ holds, where 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 if, for any $\mathcal{PPT}$ adversary $\mathcal{A}$, the probability $\Pr[\mathsf{aWitExt}^b_{\mathcal{A},\mathsf{aSIG}_2}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$ holds, where 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 the previous definitions [7], [20], [28] due to the incorporation of a one-party chosen witness, specifically impacting the four oracles, the aSigForge game, and the aWitExt game. For illustration, we discuss the aSigForge game, and the aWitExt game follows a similar situation. In the previous definitions, the adversary attempts to forge a valid signature σ^* for a challenged statement Y^* and a pre-signature $\widehat{\sigma}$. The adversary can only 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 model works for common signatures, it fails 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 witness extractability, and adapt it to forge the signature σ^* for the original challenged message m^*. The chosen-witness approach in 2PWAS introduces adjustments in the aSigForge game, by additionally enabling queries to the honest and corrupted relation oracles. However, the challenged statement Y^* must remain uncorrupted. This ensures the security of 2PWAS, even in the context of unique signatures.

4. Proof of Our Generic Construction

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

Lemma 8 *The **Pre-Signature Correctness** of $\mathsf{aSIG}^{\mathsf{G}}_2$ can be proved under the assumptions of Theorem 2.*

Proof. 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 3). Then, it holds that $\sigma := f_{\mathsf{debnd}}(f_{\mathsf{bnd}}(\sigma, y_0), y_0)$ according to Equation 2, 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 4).

To establish the **2-aEUF-CMA-Security** of $\mathsf{aSIG}^{\mathsf{G}}_2$, 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 9 *The **2-aEUF-CMA-Security** of $\mathsf{aSIG}^{\mathsf{G}}_2$ can be proved under the assumptions of Theorem 2.*

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

$\underline{\mathcal{O}_H(pp)}$
$(Y_i; y_i) \in \mathsf{RG}(pp)$
$\mathcal{Q}_H := \mathcal{Q}_H \cup \{(Y_i; y_i)\}$
return Y_i

$\underline{\mathcal{O}_C(Y_i)}$
if $Y_i \in \mathcal{Q}_H$ **then**
$\mathcal{Q}_H := \mathcal{Q}_H \backslash \{(Y_i; y_i)\}$
$\mathcal{Q}_C := \mathcal{Q}_C \cup \{(Y_i; y_i)\}$
endif
return y_i

$\underline{\mathcal{O}^b_{\Gamma_S}(m)}$
retrieve $(m, Y_i) \leftarrow \mathcal{Q}_{pS}$
$\forall i, \mathcal{Q}_H := \mathcal{Q}_H \backslash \{(Y_i; y_i)\}$
$\mathcal{Q}_C := \mathcal{Q}_C \cup \{(Y_i; y_i)\}$
$\mathcal{Q}_S := \mathcal{Q}_S \cup \{m\}$
$\sigma \leftarrow \Gamma_{\mathsf{Sign}_{\langle sk_{1-b}; \cdot \rangle}}(pk_0, pk_1, m)$
return σ

$\underline{\mathcal{O}^b_{\Gamma_{pS}}(m, Y_i)}$
if $Y_i \in \mathcal{Q}_H \cup \mathcal{Q}_C$ **then**
retrieve $(Y_i; y_i) \leftarrow \mathcal{Q}_H \cup \mathcal{Q}_C$
$\mathcal{Q}_{pS} := \mathcal{Q}_{pS} \cup \{(m, Y_i)\}$
$(\widehat{\sigma}, Y_i, \pi_{1-b}) \leftarrow \Gamma_{\mathsf{pSign}_{\langle sk_{1-b}, y_i; \cdot \rangle}}(pk_0, pk_1, m)$
endif
if $m \in \mathcal{Q}_S$ **then**
$\mathcal{Q}_H := \mathcal{Q}_H \backslash \{(Y_i; y_i)\}$
$\mathcal{Q}_C := \mathcal{Q}_C \cup \{(Y_i; y_i)\}$
endif

Figure 6: Definition of involved oracles.

Game G_0. This game is the original experiment aSigForge defined in Section 4.3. 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 a statement 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 pre-signature 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 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 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 SigForge game. Thus, the simulation of G_4 has been achieved by $\mathcal{S}$. With winning the SigForge game, $\mathcal{S}$ utilizes $\mathcal{A}$'s forgery (m^*, σ^*) as its answer to the SigForge game. It should be noted that $\mathcal{A}$ wins 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 SigForge game. This also implies a breach of the hiding property, as the adversary recovers σ^* from a given pre-signature $\widehat{\sigma} \leftarrow f_{\mathsf{bnd}}(\sigma^*, y^*)$ with a non-negligible probability, which contradicts to Equation 1.

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

$$\Pr[\mathsf{aSigForge}_{\mathcal{S}^{\mathcal{A}},\mathsf{aSIG}_2}(\lambda) = 1] \tag{5}$$
$$\leq \Pr[\mathsf{SigForge}_{\mathcal{S}^{\mathcal{A}},\mathsf{SIG}_2}(\lambda) = 1] + \mathsf{negl}(\lambda). \tag{6}$$

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

$\underline{\mathsf{pVerf}_P(Y_0, m, \widehat{\sigma}, \pi_0)}$
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 $(Y'_0 == Y_0)$

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

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

Figure 7: 2-party BLS adaptor signature scheme

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-aEUF-CMA-security of $\mathsf{aSIG}_2^{\mathsf{G}}$ is bounded by the maximum probability of successfully attacking either the hardness of R or the EUF-CMA-security of SIG_2.

Lemma 10 *The **Pre-Signature Adaptability** of $\mathsf{aSIG}_2^{\mathsf{G}}$ can be proved under the assumptions of Theorem 2.*

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 $f_{state}(pp, y_0) = f_{shift}(pk, m, \widehat{\sigma})$. We can further recover $\sigma := f_{\mathsf{debnd}}(f_{\mathsf{bnd}}(\sigma, y_0), y_0)$, cf., Equation 2. In addition, 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 11 *The **Witness Extractability** of $\mathsf{aSIG}_2^{\mathsf{G}}$ can be proved under the assumptions of Theorem 2.*

Proof. The proof of this lemma is similar to that of Lemma 9. The difference lies in the fact that the witness extractability can be reduced to only the 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 SigForge experiment.

5. Instantiations

Instantiation 1 (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 $P_i = d_i G_2, \forall i \in \{0, 1\}$, 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 7.

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.

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

Equation 1 holds due to the uniform distribution of $\widehat{s}$. Equations 2 and 4 are trivial to prove: $\forall s, y \in \mathbb{G}_1$, $f_{\mathsf{debnd}}(f_{\mathsf{bnd}}(s, y), y) = (s + y) - y = s$ and $f_{\mathsf{ext}}(f_{\mathsf{bnd}}(s, y), s) = (s + y) - s = y$. For Equation 3, 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 3 also holds.

Instantiation 2 (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 $P_i = d_i G, \forall i \in \{0, 1\}$. 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 8.

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.

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

Likewise, Equation 1 holds due to the the uniform distribution of $\widehat{s}$. Equation 2 and 4 are intuitive based on the relationship $\forall s, y \in \mathbb{Z}_q^*, f_{\mathsf{debnd}}(f_{\mathsf{bnd}}(s, y), y) = (s + y) - y = s \pmod q$ and $f_{\mathsf{ext}}(f_{\mathsf{bnd}}(s, y), s) = (s+y) - s = y \pmod q$. With regard to Equation 3, let us arbitrarily select public key $P \in \mathbb{G}$, a correct Schnorr signature (R, s) with

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

$\underline{\mathsf{pVerf}_P(Y_0, m, \widehat{\sigma}, \pi_0)}$
parse $\pi_0 = (c_0, z_0)$
parse $\widehat{\sigma} = (R, \widehat{s})$
$R'_0 = z_0 G + c_0 Y_0$
$c'_0 = \mathcal{H}(R'_0, Y_0, G)$
if $c'_0 \neq c_0$ **then return** 0
$Y'_0 = \widehat{s}G - \mathcal{H}(m, R)P$
return $(Y'_0 = Y_0)$

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

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

Figure 8: 2-party Schnorr adaptor signature scheme

$\underline{\Gamma_{\mathsf{pSign}\langle d_0,y_0;d_1\rangle}(pp,P_0,P_1,m)}$
$(r,s) \leftarrow \Gamma_{\mathsf{sign}\langle d_0,d_1\rangle}(P_0,P_1,m)$
$\widehat{s} = s \cdot y_0^{-1} \pmod q$
$K = s^{-1}(\mathcal{H}(m)G + rP)$
$K' = \mathsf{Uncompress}(r,+)$
if $K' = K$ **then** $b = +$
else $b = -$
$Y_0 = y_0 K$
$r_0 \in \mathbb{Z}_q^*, R_0 = r_0 K$
$c_0 = \mathcal{H}(R_0, Y_0, K)$
$z_0 = r_0 - c_0 \cdot y_0 \pmod q$
$\pi_0 = (c_0, z_0, b)$
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 $\pi_0 = (c_0, z_0, b)$
$K = \mathsf{Uncompress}(r,b)$
$R_0' = z_0 K + c_0 Y_0$
$c_0' = \mathcal{H}(R_0', Y_0, K)$
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_0

Figure 9: 2-party ECDSA adaptor signature scheme

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

$\underline{\mathsf{Adapt}_P(\widehat{\sigma}, y_0)}$
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 $\widehat{\sigma} = (r,\widehat{s})$
parse $\pi_0 = (c_0, z_0, z_s, b)$
$K = \mathsf{Uncompress}(r,b)$
$R_0' = z_0 G + c_0 Y_0$
$R_g' = z_s G + c_0 Z_g$
$R_k' = z_s K + c_0(\mathcal{H}(m)G + rP)$
$c_0' = \mathcal{H}(R_0', R_g', R_k', Y_0, Z_g, K)$
if $c_0' \neq c_0$ **then return** 0
else return 1

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

Figure 10: Brief review of ECDSA-based 2PWAS [30]

$sG = R + \mathcal{H}(m,R)P$, 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 3 also holds.

Instantiation 3 (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 9.

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.

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

Equation 1 holds because the information leaked from $\widehat{s}$ to s is computationally indistinguishable from random, given the uniform distribution of $\widehat{s}$. It is also intuitive to prove Equations 2 and 4, according to $\forall s,y \in \mathbb{Z}_q^*, f_{\mathsf{debnd}}(f_{\mathsf{bnd}}(s,y),y) = (s \cdot y^{-1}) \cdot y = s \pmod q$, $f_{\mathsf{ext}}(f_{\mathsf{bnd}}(s,y),s) = (s \cdot y^{-1})^{-1} \cdot s = y \pmod q$. With regard to Equation 3, 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

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

and hence Equation 3 also holds.

Brief review of ECDSA-Π_{AS} [30].* For a clearer comparison, we briefly review ECDSA*-Π_{AS} [30]. As outlined in Section 3, this scheme involves two hard relations: $\mathsf{R}_a = (Y;y) | Y = yG$ and $\mathsf{R}_v = (m, pk, Z; \sigma = (r,s)) | \mathsf{Verf}_{pk}(m,\sigma) = 1 \wedge Z = (r, sG)$. Thus, we apply Σ-protocol and Fiat-Shamir transformation to construct the proofs for these two relations. Detailed algorithmic descriptions of each component in ECDSA*-Π_{AS} could be referred to Figure 10.

6. 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 IND-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 [39] 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.

7. 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 [37] 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 variant secret key λ^* such that $\alpha^{n\lambda^*} := 1 \pmod{n^2}$ [4], 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. In general, a range proof is required to verify that $s \in \mathbb{Z}_q^*$. However, by ensuring that $\mathcal{H}(m)G + rP := sK$, we confirm that s is a valid ECDSA signature, and thus we eliminate the need for separate range proofs. 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 secret key λ or variant λ^* to recover the signature $\sigma := (r, s)$.
- OVTS.ForceOpen. This algorithm first repeats squaring to compute $w := u^{2^{\mathcal{T}}} \pmod n$. Then, it recovers the variant 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 [52] 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 $c := \mathcal{H}(x_{ovts}, T_{ct}, T_u, T_v,$

4. The original puzzle λ results in a complex zk-SNARK when expressed as the correct decryption $ct^\lambda \pmod{n^2} := ns\lambda + 1 \pmod n$. To streamline and leverage an efficient Σ-protocol, we introduce a variant secret key λ^*, satisfying $\alpha^{n\lambda^*} := 1 \pmod{n^2}$. This key also ensures correct decryption due to $(\frac{ct^{\lambda^*} \pmod{n^2} - 1}{n}) \cdot \lambda^{*-1} := \frac{1 + n\lambda^* s - 1}{n} \cdot \lambda^{*-1} := s \pmod n$.

$T_R, T_{R_1}, T_{R_2}, T_1, T_2, T_K)$ 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.