UniCross: A Universal Cross-Chain Payment Protocol with On-demand Privacy and High Scalability

Chenke Wang, Yu Long, Xian Xu, Shi-Feng Sun, Yiqi Liu, Dawu Gu

***Abstract*—Cross-chain payment technologies have obtained broad affirmation from industry and academia as they enable assets to be circulated across the boundaries of various blockchains. However, existing cross-chain payment protocols are tailored for limited blockchains, inflexible in providing privacy guarantees, and unsatisfactory in scalability.**

To address these issues, this paper proposes a universal cross-chain payment framework. This framework enables payments across a wide range of blockchains since it is independent of any specific blockchain features. Moreover, this framework provides on-demand privacy and high scalability. To instantiate the framework, we introduce UniCross, a novel universal cross-chain payment protocol. Concretely, we utilize the ring learning with errors (RLWE)-based encryption scheme and propose a new non-interactive zero-knowledge (NIZK) protocol, named HybridProof, to construct UniCross. We formally define the security of the universal cross-chain payment framework and prove the universal composability (UC) security of UniCross. The proof-of-concept implementation and evaluation demonstrate that (1) UniCross consumes up to 78% and 94% less communication and computation cost than the state-of-the-art work; (2) UniCross achieves a throughput (∼360 tps) 36× that of the state-of-the-art work (∼10 tps).

***Index Terms*—Cross-chain payment, multi-chain, privacy, batching, NIZK, universal composability.**

I. Introduction

Since the advent of Bitcoin, numerous independent blockchains have been proposed, forming a complex ecosystem. As digital assets become progressively dispersed across these heterogeneous blockchains, the demand for cross-chain payment protocols has grown significantly. In a typical cross-chain payment setting, two users are operating on different blockchains, respectively, i.e., Alice on the source chain $\mathcal{L}_S$ and Bob on the target chain $\mathcal{L}_D$. To fulfill a cross-chain payment, Alice spends a unit of coin on $\mathcal{L}_S$, enabling Bob to receive an equivalent amount on $\mathcal{L}_D$. There are two critical requirements in this process. (1) *Atomicity* requires that Alice's spending on $\mathcal{L}_S$ succeeds if and only if Bob's receiving on $\mathcal{L}_D$ succeeds. (2) *Privacy*, on top of atomicity, ensures that no external observer can associate Alice with Bob.

To satisfy these requirements, early solutions [1], [2], [3], [4] relied on a trusted third party (TTP) to facilitate cross-chain payments. However, the assumption of TTP somewhat weakens security, especially in distributed cryptoeconomic systems where a TTP is not always available. Consequently, most of the recent works employ untrusted intermediaries instead of TTPs. Nevertheless, almost all existing untrusted cross-chain payment protocols are only tailored for limited cross-chain scenarios.

From the perspective of *compatibility with the underlying blockchains*, most protocols achieve atomicity (and privacy) by relying on different blockchain features, thereby only compatible with specific blockchains. For instance, [5] operates exclusively on Proof-of-Work (PoW) source chains; [6], [7], [8], [9], [10], [11], [12] require the target chain to support expressive smart contracts; [13], [14] rely on hashed timelock contracts (HTLC) implemented on both the source and target chains. Beyond the compatibility challenge brought by heterogeneous blockchains, *security and privacy objectives* may also diverge. Some protocols [5], [6], [7], [8], [9], [10], [14], [15], [16], [17] focus solely on guaranteeing atomicity, whereas others [11], [12], [13], [18], [19], [20], [21] concentrate on strengthening the privacy protection. Therefore, for users conducting payments across multiple chains (with or without privacy requirements) and for service providers offering cross-chain support, they are both required to integrate with diverse cross-chain payment protocols. This results in poor user experience and significant additional complexity for cross-chain service providers. For example, suppose that Alice at Ethereum owns some coins. She wants to pay Bob one Bitcoin coin privately, and pay Carol an Arbitrum (a sidechain of Ethereum) coin without privacy. In this case, Alice and the service providers have to integrate with two separate cross-chain payment protocols.

The above observations motivate us to put forth the *universal cross-chain payment* protocol. Such a protocol can simultaneously support payments across multiple blockchains while incorporating additional properties not in existing solutions. (1) *Optimal compatibility*: The protocol remains agnostic to specific blockchain designs. In other words, it relies solely on signature verification (SV), the most basic scripting function supported by all blockchains. Thereafter, we also refer to optimal compatibility as "scriptless" in the remainder of this paper. (2) *On-demand privacy*: The protocol provides basic atomicity guarantees by default and can be *user-configured* for privacy. Meanwhile, the operations of service providers remain the same regardless of whether the cross-chain payment is private or not.

To realize a universal cross-chain payment protocol, a highly promising approach is to adapt some Bitcoin-compatible private cross-chain payment protocols (e.g., [18], [19], [21]). These protocols ensure optimal compatibility by restricting their on-chain operations to the SV script alone. This minimizes dependencies on blockchain-specific features. More-

Chenke Wang, Yu Long, Shi-Feng Sun, and Dawu Gu are with Shanghai Jiao Tong University, Shanghai, China. Email: w_chenke@sjtu.edu.cn; longyu@sjtu.edu.cn; shifeng.sun@sjtu.edu.cn; dwgu@sjtu.edu.cn.

Xian Xu and Yiqi Liu are with East China University of Science and Technology, Shanghai, China. Email: xuxian@ecust.edu.cn, y80220049@mail.ecust.edu.cn.

over, users can flexibly turn built-in privacy measures on or off without any modifications to the server-side algorithms. Thereby, these protocols achieve on-demand privacy. Among them, A^2L^+ [19] stands out for its rigorous security guarantees and best operational performance.

Although A^2L^+ [19] appears to be a promising candidate, it falls short when simply migrated to a universal solution. A universal cross-chain payment protocol must support large-scale cross-chain payments across heterogeneous blockchains, where *scalability* is essential to achieve high throughput at low cost. However, A^2L^+ suffers from limited scalability. For instance (cf. results in Section VII), it sustains only $\sim$10 payments per second (tps) at a communication cost of 11.80 KB per payment. In contrast, UAS [17], with optimal compatibility but without privacy, reaches about 42$\times$ the throughput (i.e., $\sim$420 tps) at a communication cost of only 0.69 KB per payment. The substantial performance gap arises from A^2L^+'s reliance on the algebraic structure of *class groups* to ensure privacy. According to the implementation of A^2L^+ (cf. [22]), exponentiation and group element size in the class group (i.e., $\sim$8.71 ms and 1070 B) are 66$\times$ and 31$\times$ more expensive than those in the elliptic curve group over *Secp256k1* used in UAS (i.e., $\sim$0.13 ms and 33 B). These observations underscore the non-trivial challenge of designing a practical universal cross-chain payment protocol.

To design a universal cross-chain payment protocol with optimal compatibility and on-demand privacy while attaining notably high scalability, we design UniCross. In particular, we provide the formal definition of a universal cross-chain payment framework and a novel approach to its construction. We incorporate the batching concept into Bitcoin-compatible private cross-chain solutions and remove the dependency on the class group to boost the scalability of cross-chain servers. Here, batching refers to the service provider's ability to process multiple cross-chain payments in an aggregated manner, thereby amortizing the cost per payment.

A. Our Contribution

This work includes the following contributions.

- **The first universal cross-chain payment framework and a novel construction.** We propose the first universal cross-chain payment protocol, enabling simultaneous payments across multiple blockchains. Compared with existing cross-chain payment protocols, this framework provides more properties, including optimal compatibility, on-demand privacy, and scalability. Accordingly, we design UniCross, a universal cross-chain payment protocol achieving the above properties. The comparison with previous works is shown in Table I.
- HybridProof**: an efficient NIZK for batching.** We propose a novel zero-knowledge proof designed to facilitate our batching mechanism, named HybridProof. It features a transparent setup and achieves logarithmic proof size.
- **Formal UC security definition and proof.** We provide the first full-fledged model of a universal cross-chain payment framework in the UC framework. We rigorously prove that our construction, UniCross, is UC-secure. The approach for UC modeling and analysis in this work would be helpful for cross-chain payment protocols to come in the future.
- **Implementation and evaluation.** Experiments show that UniCross is rather practical. Specifically, (1) UniCross consumes up to 78% and 94% less communication and computation cost than A^2L^+; (2) after applying pre-computation optimization to both works, UniCross achieves a throughput ($\sim$360 tps) 36$\times$ that of A^2L^+ ($\sim$10 tps).

B. Paper Organization

We organize the remainder of the paper as follows. The related works are summarized in Section II. Section III describes the preliminary of this paper. In Section IV, the UniCross architecture and security model are presented. Following this, the detailed description of UniCross is shown in Section V. UniCross's security proof is given in Section VI. Section VII describes UniCross's implementation and comparison with other solutions. Finally, the conclusion is in Section VIII.

II. Related Work

A. Single-pair Cross-chain Solutions

Single-pair cross-chain solutions focus on payments from a single source chain to a single target chain, i.e., one-to-one. We categorize existing one-to-one cross-chain solutions based on the blockchain features they rely on. Broadly, existing works fall into the following categories.

1) Solutions Relying on Specific Consensus: To achieve secure cross-chain operations, [5], [8], [9] allow $\mathcal{L}_D$ to verify that Alice's spending transaction has already been included in $\mathcal{L}_S$, and then enforce Bob's receiving. The verification of transaction inclusion always depends on the consensus of $\mathcal{L}_S$. To reduce the verification overhead on $\mathcal{L}_D$, succinct non-interactive arguments of knowledge (SNARKs) are exploited in [8], [9] to prove the transaction inclusion. For instance, the well-known zkBridge [9] introduces a decentralized SNARK-proof system and implements it to support Proof-of-Stake (PoS) source chain, using high-performance machines as the service providers. However, such an efficient implementation of zkBridge cannot be directly utilized when source chains operate on other consensus, e.g., PoW. After that, Glimpse [5] is proposed to further reduce the cost on $\mathcal{L}_D$ by avoiding continuous relaying blocks of $\mathcal{L}_S$ to $\mathcal{L}_D$ like zkBridge. However, Glimpse assumes that $\mathcal{L}_S$ operates a PoW consensus and $\mathcal{L}_D$ supports the hash function used in the PoW consensus of $\mathcal{L}_S$, which always rules out some cross-chain combinations. Apart from limited compatibility, none of the above works explicitly addresses the privacy concerns in cross-chain payments.

2) Solutions Relying on Expressive Smart Contracts: Some cross-chain solutions [6], [7], [10], [11], [12] do not restrict the consensus operated by source chains. However, they are only compatible with target chains supporting smart contracts, a piece of code automatically executed by miners on the blockchain. Xclaim [6] exploits a collateralized intermediary, named Vault, to lock a coin of Alice in $\mathcal{L}_S$ and relies on a smart contract to verify the on-chain lock state and then issue a coin to Bob in $\mathcal{L}_D$. Alba [7] avoids relaying the on-chain state of the source chain to the target chain. Instead, it relies

TABLE I: Comparison of trustless state-of-the-art cross-chain solutions.

Scheme	Atomicity	Optimal compatibility	Privacy		Scalability	Security model for multi-chain
			Built-in privacy	On-demand privacy		
zkBridge [9]	●	○	○	○	○	○
Glimpse [5]	●	○	○	○	●	○
Xclaim [6], Alba [7], CVPC [10]	●	○	○	○	●	○
zkCross [11]	●	○	●	○	○	○
Accio [12]	●	○	●	●	●	○
AHML [15], PipeSwap [16]	●	●	○	○	●	○
UAS [17]	●	●	○	○	●	●
TumbleBit [13]	●	○	●	●	○	○
A^2L^+ [19], P2C2T [21]	●	●	●	●	○	○
SweepUC [20]	●	●	●	○	○	○
UniCross	●	●	●	●	●	●

●: support; ○: no support.

on a smart contract to verify the payment channel state of the source chain and then enforce Bob's receiving. Cross-chain virtual payment channel (CVPC) [10] provides an off-chain cross-chain solution without relying on intermediaries during the payment phase. Accordingly, it requires a smart contract to verify the correctness of the updated payment channels of the target chain. Further, considering that privacy is another crucial property for many cross-chain protocols, zkCross [11] exploits zkSNARK to prove that a transaction has been included to $\mathcal{L}_S$ without leaking Alice's identity. Thereafter, a smart contract on the target chain is used to verify the proof and then enforce Bob's receiving. Besides, to achieve both privacy and scalability, Accio [12], proposed as a payment channel hub, can be adapted to a cross-chain solution and is featured as NIZK-free, i.e., without relying on any zero-knowledge proof. However, Accio still requires a smart contract to verify the correctness of the updated payment channels, like CVPC. Although smart contracts have many benefits for favoring the support of DeFi applications, there are other blockchains (e.g., Bitcoin-main-chain and Ripple [23]) that more conservatively argue for a reduced trust base and easier script verification. For this reason, supporting blockchains without smart contract capabilities is not only theoretically challenging but also a practically relevant research goal.

3) Solutions Relying on HTLC: Solutions [13], [14] rely on HTLC, a script compatible with Bitcoin. As a secure cross-chain payment protocol, HTLC-based solutions are deployed in practice (e.g., [14]). After that, TumbleBit [13] is proposed to further provide privacy. However, HTLC-based solutions have inherent challenges that reduce their utility, which we summarize next. First, they are not compatible with several blockchains, such as Ripple [23] and Stellar [24] that do not support the computation of the HTLC in their scripting language. Second, they incur high on-chain execution cost, as well as large transaction sizes due to large scripts. Third, cross-chain transactions involved in HTLC-based solutions are distinguishable from intra-chain transactions, which degrades the fungibility of cross-chain coins.

4) Scriptless Solutions: AMHL [15] firstly utilizes adaptor signatures to facilitate atomic payments, and thus does not rely on any special script. PipeSwap [16] further replaces timelock contracts with the verifiable discrete logarithm (VTD) scheme and proposes a two-hop swap and two-hop refund mechanism to avoid the double-claiming attack incurred by VTD. Unlike all these security-only solutions, A^2L [18] is proposed as the first scriptless and private solution. Then, [19] points out the flaw in A^2L's security proof and proposes A^2L^+ with a formal proof in the game-based model. A^2L^+ achieves optimal compatibility and on-demand privacy we desire. However, it suffers from poor scalability due to its dependency on class groups. P2C2T [21] adapts A^2L^+ to strengthen on-chain indistinguishability by replacing the timelock contracts with the VTD scheme. However, P2C2T further compromises scalability. To achieve universal composability (UC) security, [19] proposes proof-of-concept A^2L^{UC} with some general-purpose cryptographic tools. SweepUC [20] cancels A^2L^{UC}'s dependency on these tools and provides the first full-fledged UC-secure cross-chain payment protocol. However, SweepUC cannot provide on-demand privacy, and thus falls short of the universal cross-chain payment protocol.

B. Many-to-many Cross-chain Solutions

To the best of our knowledge, Thyagarajan *et al.* [17] presents the only existing many-to-many cross-chain solution, offering a scriptless approach that can simultaneously facilitate secure coin transfers across multiple blockchains. However, [17] does not address privacy, which motivates our development of a universal cross-chain payment framework featuring both optimal compatibility and on-demand privacy, while maintaining the scalability demonstrated by [17].

III. Preliminaries

A. Notations

We denote the set $\{0, 1, \ldots, n-1\}$ by $[n]$. We use lowercase letters with arrows (e.g., $\vec{v}$) for column vectors and uppercase letters (e.g., A) for matrices. Polynomials are denoted by bold and lowercase letters (e.g., $\mathbf{f}$). Column vectors and matrices of polynomials are denoted by bold lowercase letters with arrows (e.g., $\vec{\mathbf{v}}$) and bold capital letters (e.g., $\mathbf{A}$), respectively. Let $\langle \cdot, \cdot \rangle$ be the inner product operation, "$\circ$" be the Hadamard product or entry-wise multiplication operation, "$\otimes$" be the tensor product operation, "$\| \cdot \|_\infty$" be the infinity norm. We use $\lceil \cdot \rceil, \lfloor \cdot \rfloor, \lfloor \cdot \rceil$ to denote the ceiling, floor, and rounding to the nearest integer functions, respectively.

Ring-related notations. Let $\mathcal{R}_q$ be the set of integer polynomials $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^d + 1)$, where $\mathbb{Z}_q = \{-(q -$

$1)/2, \cdots, (q-1)/2\}$. Let $\mathbb{S}_{\mathcal{B}}$ be the set of polynomials in $\mathcal{R}_q$ whose maximum absolute coefficients are at most $\mathcal{B} \in \mathbb{Z}^+$.

Group-related notations. Let $\mathbb{G}$ be a cyclic group of prime order p, $\mathbb{Z}_p = \{0, 1, \cdots, p-1\}$ and $\mathbb{Z}_p^* = \mathbb{Z}_p \backslash \{0\}$. For operations in $\mathbb{G}$, we have $\vec{g}^{\vec{a}} = g_0^{a_0} \cdots g_{k-1}^{a_{k-1}} \in \mathbb{G}$, $g^{\vec{a}} = (g^{a_0}, \cdots, g^{a_{k-1}})^{\mathrm{T}} \in \mathbb{G}^k$, $\vec{g}^{a} = (g_0^a, \cdots, g_{k-1}^a)^{\mathrm{T}} \in \mathbb{G}^k$.

Coefficient representation and multiplication matrices. For a polynomial vector $\vec{\mathbf{s}} \in \mathcal{R}_q^k$, define $\mathsf{Coeff}(\vec{\mathbf{s}}) \in \mathbb{Z}_q^{kd}$ to be coefficient vector of $\vec{\mathbf{s}}$. For a polynomial $\mathbf{f} = f_0 + f_1 X + \ldots + f_{d-1} X^{d-1} \in \mathcal{R}_q$, we define the multiplication matrix $\mathsf{rot}(\mathbf{f}) \in \mathbb{Z}_q^{d \times d}$ as

$$\mathsf{rot}(\mathbf{f}) = \begin{bmatrix} f_0 & -f_{d-1} & \cdots & -f_1 \\ f_1 & f_0 & \cdots & -f_2 \\ \vdots & \vdots & \vdots & \vdots \\ f_{d-1} & \cdots & f_1 & f_0 \end{bmatrix}.$$

We extend this definition for matrices over $\mathcal{R}_q$ (e.g., $\mathbf{F} \in \mathcal{R}_q^{n_1 \times n_2}$). Then, for any polynomial vector $\vec{\mathbf{x}} \in \mathcal{R}_q^{n_2}$, we have the following property over $\mathbb{Z}_q$: $\mathsf{Coeff}(\mathbf{F}\vec{\mathbf{x}}) = \mathsf{rot}(\mathbf{F})\mathsf{Coeff}(\vec{\mathbf{x}})$. Besides, the function Bin maps a signed b-bit integer to its binary representation: $\mathsf{Bin}(z) = \vec{z}$ where $\vec{z} = (z_0, \cdots, z_{b-1})^{\mathrm{T}}$ satisfies $z = z_0 + z_1 2 + \cdots + z_{b-2} 2^{b-2} - z_{b-1} 2^{b-1}$.

B. Cryptographic Primitives

Lattice-based additive homomorphic encryption. In this work, we use two lattice-based encryption schemes, i.e., the learning with errors (LWE) encryption scheme and its ring variant (RLWE) [25]. There exist efficient conversion algorithms between an RLWE ciphertext and LWE-based ciphertexts [26]. These two schemes share a set of public parameters $\mathsf{pp} = \{d, q, t, \mathcal{B}\}$. Let $\Lambda = \lceil q/t \rceil$, we leverage the following functions.

- $(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{RLWE.Gen}(1^\lambda)$. Sample $\mathsf{dk}, \mathbf{e} \leftarrow\!\$\, \mathbb{S}_{\mathcal{B}}$, $\mathbf{a} \leftarrow\!\$\, \mathcal{R}_q$. Compute $\mathsf{ek} = (-(\mathbf{a} \cdot \mathsf{dk} + \mathbf{e}), \mathbf{a})^{\mathrm{T}} \stackrel{\text{def}}{=} (\mathsf{ek}_0, \mathsf{ek}_1)^{\mathrm{T}}$ over $\mathcal{R}_q^2$ and output the encrption-decryption pair $(\mathsf{dk}, \mathsf{ek})$.
- $\mathsf{ct} \leftarrow \mathsf{RLWE.Enc}(\mathsf{ek}, \mathbf{m})$. On input the encryption key ek and a message $\mathbf{m} \in \mathcal{R}_t$, sample $\mathbf{u}, \mathbf{e}_1, \mathbf{e}_2 \leftarrow\!\$\, \mathbb{S}_{\mathcal{B}}$. Then, compute and output

$$\mathsf{ct} = \begin{pmatrix} \mathsf{ek}_0 & 1 & 0 & \Lambda \\ \mathsf{ek}_1 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{u} \\ \mathbf{e}_1 \\ \mathbf{e}_2 \\ \mathbf{m} \end{pmatrix} \stackrel{\text{def}}{=} (\mathsf{ct}[0], \mathsf{ct}[1])^{\mathrm{T}} \in \mathcal{R}_q^2.$$

- $\mathbf{m} \leftarrow \mathsf{RLWE.Dec}(\mathsf{dk}, \mathsf{ct})$. Parse $\mathsf{ct} := (\mathsf{ct}[0], \mathsf{ct}[1])$ and output $\mathbf{m} = \lfloor t/q \cdot (\mathsf{ct}[0] + \mathsf{ct}[1] \cdot \mathsf{dk}) \rceil \mod t$.
- $\mathsf{ct}_i \leftarrow \mathsf{RLWE.Extract}(\mathsf{ct}, i)$. On input the RLWE-based ciphertext ct and the index $i \in [d]$, we can extract $\mathsf{ct}_i = (b_i, \vec{c}_i) \in \mathbb{Z}_q^{d+1}$, which is an LWE ciphertext of the i-th coefficient of $\mathbf{m}$ under the secret key $\mathsf{Coeff}(\mathsf{dk})$. Here, b_i is the i_{th} coefficient of $\mathsf{ct}[0]$ and $\vec{c}_i = (\mathsf{ct}[1]_i, \mathsf{ct}[1]_{i-1}, \cdots, \mathsf{ct}[1]_0, -\mathsf{ct}[1]_{d-1}, \cdots, -\mathsf{ct}[1]_{i+1})$, where $\mathsf{ct}[1]_k$ is the k_{th} coefficient of $\mathsf{ct}[1]$.
- $m \leftarrow \mathsf{LWE.Dec}(\mathsf{dk}, \mathsf{ct}_i)$. Parse $\mathsf{ct}_i = (b, \vec{c})$ and output $m = \lfloor t/q \cdot (b + \langle \vec{c}, \mathsf{Coeff}(\mathsf{dk}) \rangle) \rceil \mod t$.

We require that the RLWE encryption scheme is *CPA-secure* and *additively homomorphic*. Given RLWE ciphertexts ct and ct', which respectively encrypt polynomials $\mathbf{m}$ and $\mathbf{m}'$, the operation $\mathsf{ct} + \mathsf{ct}'$ (resp. $\mathsf{ct} - \mathsf{ct}'$) results in an RLWE ciphertext that decrypts to $\mathbf{m} + \mathbf{m}' \in \mathcal{R}_t$ (resp. $\mathbf{m} - \mathbf{m}'$). The additive homomorphism of LWE ciphertexts is the same.

Adaptor signatures. Unlike traditional signature schemes, adaptor signatures work in a two-stage signing fashion. Concretely, the adaptor signature [27], [28], [29], [30] scheme Π_{AS} is defined with respect to a hard relation R and a digital signature scheme $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$. Inheriting the algorithms of SIG, Π_{AS} consists of four extra algorithms.

- $\hat{\sigma} \leftarrow \mathsf{PreSig}(\mathsf{sk}, m, Y)$. It allows users to generate a pre-signature $\hat{\sigma}$ on a message m w.r.t the statement Y.
- $0/1 \leftarrow \mathsf{PreVrf}(\mathsf{pk}, m, Y, \hat{\sigma})$. It can verify if the pre-signature $\hat{\sigma}$ is valid or not.
- $\sigma \leftarrow \mathsf{Adapt}(\hat{\sigma}, y)$. Given the witness y of instance Y, it can adapt a pre-signature $\hat{\sigma}$ into a complete signature σ.
- $y \leftarrow \mathsf{Extract}(\sigma, \hat{\sigma}, Y)$. Taking σ and $\hat{\sigma}$, it outputs a witness y satisfying $(Y; y) \in \mathsf{R}$.

We require a *correct* Π_{AS}, secure for *full extractability*, *pre-signature adaptability*, and *computational pre-verify soundness*. The first two security properties are defined in [28], and the last one is introduced by [29]. We refer to Appendix E-B for the formal definitions.

Non-interactive zero-knowledge proof. A non-interactive zero-knowledge (NIZK) argument for $(x; w) \in \mathsf{R}_L$ consists of three algorithms.

- $(\mathsf{crs}, \tau) \leftarrow \mathsf{Setup}(1^\lambda)$. It outputs a common reference string crs and a simulation trapdoor τ.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)$. It is executed to generate a proof π.
- $0/1 \leftarrow \mathsf{Vf}(\mathsf{crs}, x, \pi)$. It verifies if a proof π is valid or not.

A NIZK protocol is *complete* if the prover, knowing a witness of $x \in L$, can always output a valid proof. *Knowledge soundness* means that for a valid proof π of an instance x, there exists a PPT extractor Ext who gets full access to the prover's state, can extract the witness w from π, where $(x, ; w) \in \mathsf{R}_L$. *Zero-knowledge* means that there exists a PPT algorithm Sim, taking the simulation trapdoor τ as an extra input, can simulate π for $x \in L$ without knowledge of w. We refer to Appendix E-D for the formal definitions.

IV. Architecture and Security Model

A. UniCross Architecture

The architecture of UniCross is shown in Figure 1. There are four types of entities: blockchains, senders, receivers, and the tumbler.

- *Blockchains.* In our universal cross-chain payment protocol, each blockchain is maintained by a set of miners and records all transactions among its users. Any blockchain in this setting may act as either a source chain or a target chain, and these roles can overlap. We assume that all participating blockchains function as secure and immutable ledgers.
- *Senders and receivers.* Each cross-chain sender operates on a specific source chain, and each cross-chain receiver operates on a specific target chain. If either the sender or receiver desires privacy, it can utilize UniCross to hide the link between a pair of sender and receiver, and the link between the involved source chain and target chain as well.

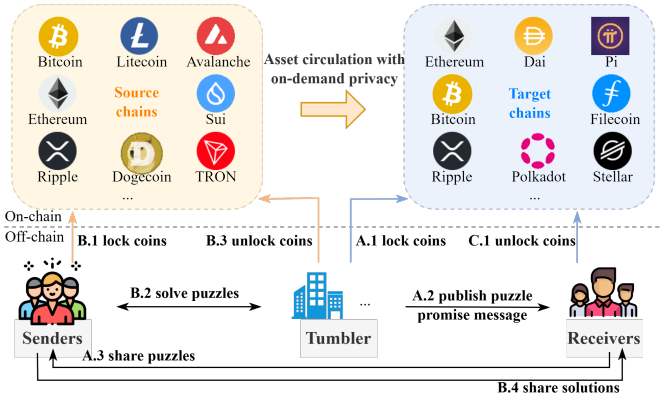


Fig. 1: Architecture of UniCross. UniCross is divided into three phases: (A) target-chain deposit; (B) source-chain transfer; (C) target-chain redemption. UniCross allows for the overlap between source chains and target chains.

- *Tumbler.* All cross-chain payments among senders and receivers are processed by one or more *trustless* cross-service providers, collectively referred to as the *tumbler*. In practice, anyone can operate a tumbler to facilitate cross-chain circulation, whether for profit or otherwise. No matter how many chains it serves for, the tumbler's functionality consists of two components. (1) On-chain component: The tumbler collects coins from senders on source chains and subsequently transfers them to receivers on target chains. To accomplish this, the tumbler holds accounts on both source and target chains. (2) Off-chain component: To minimize on-chain cost and avoid relying on specific blockchain functionalities, the tumbler binds each cross-chain payment with an independent off-chain puzzle and conducts puzzle-promise and puzzle-solving operations.

From Figure 1, each epoch contains three phases: (A) target-chain deposit, (B) source-chain transfer, (C) target-chain redemption. We emphasize that in our system, there can be any form of overlap among multi-epochs.

- *Target-chain deposit.* To fulfill the successful deposit, the tumbler locks coins on multiple target chains and publishes off-chain puzzle promise messages that convince each receiver that it will receive a coin if the puzzle is solved. After checking the validity of the deposit, each receiver processes the puzzle with its on-demand privacy and shares the puzzle with the corresponding sender for solving.
- *Source-chain transfer.* Multiple senders lock coins on their native chains (i.e., source chains), which can be unlocked if and only if these bound puzzles are solved. Then, senders process their puzzles for on-demand privacy and request the tumbler to solve these puzzles. After the tumbler solves the puzzle and unlocks the coin, the sender derives the solution and shares it with the corresponding receiver.
- *Target-chain redemption.* After receiving the solution from the sender, each receiver can unlock the coin deposited by the tumbler in the target-chain deposit phase.

We emphasize that, by exploiting the batched puzzle promise in the target-chain deposit phase, our protocol achieves state-of-the-art efficiency.

B. Security, Privacy, and Usability Goals

As previously mentioned, the universal cross-chain payment protocol must meet security, privacy, and usability goals in many-to-many cross-chain scenarios. Concretely,

- **Security goal**. This property aims to ensure *balance security* for the honest parties as in [13], also named as *atomicity*. From the system level, the cross-chain payment protocol should not be exploited to print new coins or steal coins from honest parties. From the transaction level, the sender's payment and the receiver's receipt occur atomically.
- **Privacy goal**. In many-to-many cross-chain scenarios, this goal is twofold: (1) *On-demand Privacy.* Cross-chain users have the flexibility to independently determine whether cross-chain privacy is required on a per-payment basis. (2) *Enhanced Privacy.* For each private payment, the malicious tumbler cannot identify the linkage between the sender and the receiver (i.e., identity unlinkability), nor can it identify the linkage between the involved source chain and target chain (i.e., path unlinkability).
- **Usability goal**. To support universal cross-chain payments, two core usability objectives must be satisfied: (1) *Optimal Compatibility.* UniCross should operate scriptlessly, avoiding reliance on blockchain-specific scripting languages. In other words, a wide range of blockchains can seamlessly integrate with our scheme. (2) *Scalability.* As the number of cross-chain payments from multiple chains increases, the protocol's throughput continues to perform well, without bringing in high computation and communication costs.

C. Security Model

We formalize the security and privacy of UniCross in the universal composability (UC) framework [31]. We rely on the synchronous version of the global UC framework (GUC) [32].

1) Communication Model: In our work, we assume three types of communication channels.

- For the communication between the senders/receivers and the tumbler, we assume synchronous and authenticated communication channels. Synchronous communication means that the communication proceeds in discrete rounds, and each message delivery needs 1 round. Furthermore, each communication between the party and the ideal functionality $\mathcal{F}$ requires 0 round. The computation expenses in our model are ignored for simplicity. The authenticated communication guarantees the legitimacy of the source and the integrity of each message.
- We assume that each sender-receiver pair uses anonymous communication based on the above channel. Otherwise, the tumbler can trivially link them.
- The bulletin board communication is used by the tumbler T to post the puzzle-promise message x, which is modeled by $\mathcal{F}_{\text{BB}}$ as shown in Appendix B. Roughly, posting a message x via $\mathcal{F}_{\text{BB}}.\texttt{Store}(tag, x)$ results in storing it on the board, making it possible for each receiver to load x from the board via $\mathcal{F}_{\text{BB}}.\texttt{Load}(T, tag)$.

2) Adversary Model: We consider a malicious and static adversary $\mathcal{A}$, which corrupts parties at the beginning of each epoch. Once a party is corrupted, $\mathcal{A}$ gains access to its

internal state and can fully control its behavior. All entities, including the tumbler and individual users, are susceptible to such corruption. A corrupted tumbler may generate invalid messages or refuse to participate in the protocol (e.g., denial-of-service). Similarly, corrupted users may act maliciously by refusing to cooperate with honest parties or colluding with other compromised entities.

3) Ideal Functionalities: There are three ideal functionalities to be defined.

Ledger functionality. In this framework, blockchains $\{\mathcal{L}_j^{\mathsf{SIG}}\}_{j\in[K]}$ involved in cross-chain payments are regarded as global functionalities. All $\mathcal{L}_j^{\mathsf{SIG}}$ share the same interface and behavior, and are treated as independent ledgers in the system. $\mathcal{L}_j^{\mathsf{SIG}}$ is formally defined in Appendix B. Roughly, $\mathcal{L}_j^{\mathsf{SIG}}$ uses four interfaces to communicate with other functionalities or parties.

- The $\mathcal{L}_j^{\mathsf{SIG}}$.`Update` interface records the current balance $\mathsf{bal} : \mathsf{pk} \to \mathbb{R}_{\geq 0}$ of each public key pk, which is initialized by $\mathcal{Z}$ and accessed by every entity.
- The $\mathcal{L}_j^{\mathsf{SIG}}$.`Pay` interface can be called by parties to pay c coins from pk_s to pk_r.
- The $\mathcal{L}_j^{\mathsf{SIG}}$.`Freeze` or $\mathcal{L}_j^{\mathsf{SIG}}$.`Unfreeze` interface is used to freeze c coins of or unfreeze c coins to pk respectively. The above two interfaces can be called by ideal functionalities but not by ordinary entities.

In order to model the non-immediacy of the update of the blockchain ledger, we impose a maximal delay rounds $\Delta_j \in \mathbb{N}$ on messages sent to the ledger. We denote the ledger functionality with delay Δ_j as $\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j)$.

Shared address functionality. The ideal functionality $\mathcal{F}_{s,j}$ models the opening and closing of a shared address for a ledger $\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j)$, depicted in Appendix B. Roughly, $\mathcal{F}_{s,j}$ uses two interfaces to communicate with others.

- The $\mathcal{F}_{s,j}$.`OpenSh` interface is called to open a shared address $\tilde{\mathsf{pk}}_a\&\tilde{\mathsf{pk}}_b$ by depositing c coins from pk_a. The shared address can only be closed cooperatively by P_a and P_b or with timeout. If the shared address is not closed in tlock rounds, the c coins will be automatically returned to pk_a.
- The $\mathcal{F}_{s,j}$.`CloseSh` interface is called to close the shared address $\tilde{\mathsf{pk}}_a\&\tilde{\mathsf{pk}}_b$ and transfer the c coins to pk_{out}.

Universal cross-chain payment functionality. The ideal functionality $\mathcal{F}_{\text{ucc}}$, as illustrated in Figure 2, captures the functionality and security of the universal cross-chain payment protocol. $\mathcal{F}_{\text{ucc}}$ has three interfaces as below.

- The $\mathcal{F}_{\text{ucc}}$.`TcDeposit` interface is called by receivers, which freezes coins of the tumbler and records puzzles to be solved in the following phases.
- The $\mathcal{F}_{\text{ucc}}$.`ScTransfer` interface is called by senders, which transfers coins from senders to the tumbler to solve puzzles. In detail, $\mathcal{F}_{\text{ucc}}$ freezes coins of senders and then, changes the puzzles state (i.e., from unsolved to solved) only if these coins are unfrozen into the tumbler's addresses.
- The $\mathcal{F}_{\text{ucc}}$.`TcRedeem` interface is called by receivers to redeem coins that are frozen during the target-chain deposit phase, conditioning on solved puzzles.

Note that the time of freezing coins of the senders in the source-chain transfer phase, i.e., $\mathsf{tlock}_{ps,j}$, is dependent on the maximal message delay rounds $\Delta_j \in \mathbb{N}$ of different blockchains. To prevent the adversary from inferring the pair of sender and receiver by analyzing the duration of tlock, we require that the time duration that the tumbler's coins remain frozen (i.e., tlock_{pp}) exceed all $\{\mathsf{tlock}_{ps,j}\}_{j\in[K]}$.

4) Security Definition: Let λ be the security parameter. With respect to global ledger functionalities $\{\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j)\}_{j\in[K]}$, $\mathsf{EXEC}^{\{\mathcal{L}_j^{\mathsf{SIG}}\}_{j\in[K]}}_{\mathcal{F}_{\text{ucc}},\mathcal{S},\mathcal{Z}}(\lambda)$ denotes the output of the environment $\mathcal{Z}$ when interacting with $\mathcal{F}_{\text{ucc}}$ and $\mathcal{S}$ (i.e., the simulator). And $\mathsf{EXEC}^{\{\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j),\mathcal{F}_{s,j}\}_{j\in[K]},\mathcal{F}_{\text{BB}}}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda)$ denotes the output of the environment $\mathcal{Z}$ when interacting with the real protocol Π and $\mathcal{A}$ in the $(\{\mathcal{F}_{s,j}\}_{j\in[K]},\mathcal{F}_{\text{BB}})$-hybrid model. The formal security definition is defined below.

Definition 1. *A protocol Π running in the $(\{\mathcal{F}_{s,j}\}_{j\in[K]},\mathcal{F}_{BB})$-hybrid world UC-realizes the ideal functionality $\mathcal{F}_{ucc}$ with respect to global ledgers $\{\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j)\}_{j\in[K]}$, if for any* PPT *adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that*

$$\mathsf{EXEC}^{\{\mathcal{L}_j^{\mathsf{SIG}}\}_{j\in[K]}}_{\mathcal{F}_{ucc},\mathcal{S},\mathcal{Z}}(\lambda) \approx \mathsf{EXEC}^{\{\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j),\mathcal{F}_{s,j}\}_{j\in[K]},\mathcal{F}_{BB}}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda),$$

where $\approx$ denotes the two distributions are indistinguishable.

The functionality interacts with senders $\{A_i\}_{i\in[n]}$, receivers $\{B_i\}_{i\in[n]}$, the tumbler T and the simulator $\mathcal{S}$, and a set of functionalities $\{\mathcal{L}_j^{\mathsf{SIG}}\}_{j\in[K]}$. It is parameterized by the fixed amount $\mathsf{amt} \in \mathbb{R}_{\geq} 0$ and a set of time-lock parameters tlock_{pp} and $\{\mathsf{tlock}_{ps,j}\}_{j\in[K]}$. The functionality maintains two internal lists, Puzzle and Pair, both of which are initially empty. Let $\mathsf{UserChain} : \{A_i, B_i\}_{i\in[n]} \to \{\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j)\}_{j\in[K]}$ be a mapping that indicates the blockchain to which each user belongs. Let $\mathsf{IsPrivate} = 1$ indicate that the user desires privacy, and otherwise $\mathsf{IsPrivate} = 0$. When $\mathsf{IsPrivate} = 0$, the payment amount amt can be variable, and we fix amt just for reducing the input parameters of $\mathcal{F}_{\text{ucc}}$.

`TcDeposit`

On input ("ppromise", A_i, $\mathsf{IsPrivate}_r$) from B_i, $\mathcal{F}_{\text{ucc}}$ proceeds as follows.

1. If A_i and B_i are honest, send ("promiseReq", $\diamond$, B_i, $\mathsf{IsPrivate}_r$) to $\mathcal{S}$. Otherwise, send ("promiseReq", A_i, B_i, $\mathsf{IsPrivate}_r$) to $\mathcal{S}$.
2. If T is corrupt, upon receiving ("openSh") from $\mathcal{S}$ or T is honest, go to the next step. Otherwise, abort.
   /* ("openSh"): T still requests $\mathcal{F}_{s,j}$ for opening the shared address with B_i even if T is corrupt. */
3. Call $\mathcal{L}_j^{\mathsf{SIG}}$.`Freeze`($\mathsf{pk}_T$, amt) within Δ_j rounds, where $\mathcal{L}_j^{\mathsf{SIG}} = \mathsf{UserChain}[B_i]$. Upon receiving ("failNoFund") from $\mathcal{L}_j^{\mathsf{SIG}}$, send ("failNoFund") to $\mathcal{S}$ and abort. After tlock_{pp} rounds (timeout), if $(B_i, \cdot, \cdot) \in \mathsf{Puzzle}$, remove it and call $\mathcal{L}_j^{\mathsf{SIG}}$.`Unfreeze`($\mathsf{pk}_T$, amt) within Δ_j rounds.
4. Send ("promiseRes") to $\mathcal{S}$.
5. If T is corrupt and B_i is honest, upon receiving ("invalidPromise") from $\mathcal{S}$, abort.
   /* ("invalidPromise"): the puzzle promise from T is invalid. */
6. Append $(B_i, \perp, \mathsf{IsPrivate}_r)$ to the list Puzzle.
7. Output ("promised") to B_i and stop.

`ScTransfer`

On input ("psolve", B_i, $\mathsf{IsPrivate}_s$) from A_i, $\mathcal{F}_{\text{ucc}}$ proceeds as follows.

1. Retrieve $(B_i, \cdot, \mathsf{IsPrivate}_r)$ from Puzzle. If A_i and B_i are honest, and $\mathsf{IsPrivate}_s \vee \mathsf{IsPrivate}_r = 1$, send ("psolve", A_i, $\diamond$, $\mathsf{IsPrivate}_s$) to $\mathcal{S}$. Otherwise, send ("psolve", A_i, B_i, $\mathsf{IsPrivate}_s$) to $\mathcal{S}$.
2. If B_i is corrupt, A_i is honest and receive ("noInfo") from $\mathcal{S}$, abort.
   /* ("noInfo"): the corrupt B_i does not send message to A_i. */
3. If A_i is corrupt and receive ("openSh") from $\mathcal{S}$ or A_i is honest, go to the next step. Otherwise, abort.
4. Call $\mathcal{L}_j^{\mathsf{SIG}}$.`Freeze`($\mathsf{pk}_{A_i}$, amt) within Δ_j rounds, where $\mathcal{L}_j^{\mathsf{SIG}} = \mathsf{UserChain}[A_i]$. If receive ("failNoFund") from $\mathcal{L}_j^{\mathsf{SIG}}$, send ("failNoFund") to $\mathcal{S}$ and abort. After $\mathsf{tlock}_{ps,j}$ rounds (timeout), if $(A_i, \cdot) \in \mathsf{Pair}$, remove it and call $\mathcal{L}_j^{\mathsf{SIG}}$.`Unfreeze`($\mathsf{pk}_{A_i}$, amt) within Δ_j rounds.
5. Append (A_i, B_i) to the list Pair, and send ("solveReq", A_i) to $\mathcal{S}$.
6. If T is honest, A_i or B_i is corrupt, $\mathcal{F}_{\text{ucc}}$ proceeds in three cases.

1) Upon receiving ("failSolve") from $\mathcal{S}$, abort.
/* ("failSolve"): the solving request provided by A_i is invalid or there is no puzzle-solving message from A_i. */
2) Upon receiving ("changePuzzle", $A_i, B_{i'}$) from $\mathcal{S}$, retrieve $(A_i, B_i) \in \mathsf{Pair}$ and update it to $(A_i, B_{i'})$.
/* ("changePuzzle"): A_i sends $B_{i'}$'s puzzle to T. */
3) Upon receiving ("changePuzzle", $A_i, \emptyset$) from $\mathcal{S}$, retrieve $(A_i, B_i) \in \mathsf{Pair}$ and update it to $(A_i, \emptyset)$.
/* $\emptyset$: A_i provides a puzzle that casually generated by users to T. */

7. If T is corrupt and receive ("closeSh") from $\mathcal{S}$ or T is honest, go to the next step. Otherwise, abort.
/* ("closeSh"): Corrupt T requests for closing the shared address. */
8. Call $\mathcal{L}_j^{\mathsf{SIG}}.\mathtt{Unfreeze}(\mathsf{pk}_T, \mathsf{amt})$ within Δ_j rounds, where $\mathcal{L}_j^{\mathsf{SIG}} = \mathsf{UserChain}[A_i]$. If receive ("failNoFrozenFunds") from $\mathcal{L}_j^{\mathsf{SIG}}$, send ("failNoFrozenFunds") to $\mathcal{S}$ and abort. If receive ("unfrozen", id, $\mathsf{pk}_T, \mathsf{amt}$) from $\mathcal{L}_j^{\mathsf{SIG}}$, remove $(A_i, B_i/B_{i'}/\emptyset) \in \mathsf{Pair}$ and update $(B_i/B_{i'}, \perp/\top, \cdot) \in \mathsf{Puzzle}$ to $(B_i/B_{i'}, \top, \cdot)$.
9. Output ("solved") to A_i, and stop.

TcRedeem

On input ("open", B_i) from B_i, $\mathcal{F}_{\text{ucc}}$ proceeds as follows.

1. Send ("open", B_i) to $\mathcal{S}$. If $(B_i, \top, \cdot) \notin \mathsf{Puzzle}$, abort.
2. If B_i is honest, upon receiving ("failOpen") from $\mathcal{S}$, abort.
/* ("failOpen"): B_i receives a false solution or does not receive a solution from its corresponding sender. */
3. If B_i is corrupt and $\mathcal{F}_{\text{ucc}}$ receives ("closeSh") from $\mathcal{S}$ or B_i is honest, go to the next step. Otherwise, abort.
4. Call $\mathcal{L}_j^{\mathsf{SIG}}.\mathtt{UnFreeze}(\mathsf{pk}_{B_i}, \mathsf{amt})$ within Δ_j rounds where $\mathcal{L}_j^{\mathsf{SIG}} = \mathsf{UserChain}[B_i]$. If receive ("failNoFrozenFunds") from $\mathcal{L}_j^{\mathsf{SIG}}$, send ("failNoFrozenFunds") to $\mathcal{S}$ and abort. If receive ("unfrozen", id, pk_{B_i}, amt) from $\mathcal{L}_j^{\mathsf{SIG}}$, remove $(B_i, \top, \cdot)$ from Puzzle.
5. Output ("opened") to B_i, and stop.

Fig. 2: Ideal functionality $\mathcal{F}_{\text{ucc}}$.

V. Detailed Description of UniCross

This section begins by providing a challenge & solution overview of UniCross. After that, we present the concrete constructions of two core components, including the puzzles for batching and a novel NIZK protocol. Ultimately, all components are combined together to provide a comprehensive universal cross-chain payment protocol, i.e., UniCross.

A. Challenge & Solution Overview

Recall that A^{2}L$^+$ lacks scalability when simply migrating to the universal cross-chain payment framework. To address this, UniCross incorporates the batching concept into A^{2}L$^+$ and removes the dependency on class group operations.

Challenge 1: Requirements for batchable puzzles. A naïve way for batching is to bind *multiple* sender-receiver payment pairs in one epoch to *one* shared puzzle. However, such a strategy fails because it is vulnerable to a Byzantine attack. In detail, if any sender-receiver pair behaves maliciously, the procedures of the remaining pair of users are forced to fail. Therefore, a more reasonable batching technique is to construct *one* aggregated puzzle Z, which encodes n independent sub-puzzles and solutions. Taking Z, each receiver can retrieve the sub-puzzle Z_i for itself to execute the following steps. The overhead of the aggregated puzzle Z should be much less expensive than n independent sub-puzzles, i.e., it does act as a compression. To address this challenge, we resort to the RLWE encryption scheme to construct such puzzles for batching. Even better, polynomial ring operations offer significantly higher efficiency than those in class groups.

Challenge 2: Appropriate NIZK to prove well-formedness. In UniCross, the tumbler should convince receivers that the aggregated puzzles are solvable (i.e., well-formed). To the best of our knowledge, there exists no NIZK tailored to prove the well-formedness of our newly designed puzzles. Moreover, existing general-purpose techniques (e.g., zkSNARKs) cannot be directly utilized, since it is still challenging to efficiently prove algebraic statements [33], [34]. To address this, we design a new NIZK protocol, named HybridProof, with transparent setup and logarithmic proof size.

By combining the countermeasures of the above two challenges to enhance the scalability of A^{2}L$^+$, we can construct a universal cross-chain payment protocol, named UniCross.

B. Concrete Construction of Puzzles for Batching

To describe puzzles satisfying the above requirements, we interpret the details of the puzzles for batching by specifying the algorithms PGen, PRetrieve, PRandom, PSolve.

Algorithm 1 $\mathsf{PGen}(\mathsf{ek}_T, \mathbf{m}, n)$ for $\mathbf{m} \leftarrow\!\$\, \mathcal{R}_t, \|\mathbf{m}\|_\infty \le t/6$

1: $\mathsf{ct} \leftarrow \mathsf{RLWE.Enc}(\mathsf{ek}, \mathbf{m})$
2: $\mathsf{chk} = d/n$
3: $\mathsf{splice} = (1, 2^{\lceil \log t \rceil}, \cdots, 2^{(\mathsf{chk}-1)\lceil \log t \rceil})^{\mathrm{T}} \in \mathbb{Z}^{\mathsf{chk}}$
4: **for** $i \in [n]$ **do**
5: $\quad y_i = \langle \mathsf{splice}, \mathsf{Coeff}(\mathbf{m})[i \cdot \mathsf{chk} : (i+1) \cdot \mathsf{chk} - 1] \rangle$
6: $\quad \mathsf{dlp}_i = f^{y_i}$
7: **end for**
8: **return** $Z := (\mathsf{ct}, \{\mathsf{dlp}_i\}_{i \in [n]})$

Algorithm 1 generates the aggregated puzzle $Z := (\mathsf{ct}, \{\mathsf{dlp}_i\}_{i \in [n]})$. Here, ct is a RLWE-based ciphertext encrypting $\mathbf{m} \in \mathcal{R}_t$. The reason why $\|\mathbf{m}\|_\infty$ is not $t/2$ but $t/6$ is explained in the details of Algorithm 3. And $\mathsf{dlp}_i = f^{y_i}$ is a modular exponentiation, where f is the generator of an elliptic curve group $\mathbb{G}$ of order p, y_i encodes the i-th chunk of $\mathsf{Coeff}(\mathbf{m})$ (line 5, Algorithm 1). Each chunk occupies a fixed chk number of coefficients, i.e., $\mathsf{chk} \cdot n = d$.

Algorithm 2 $\mathsf{PRetrieve}(Z, i \in [n], n)$

1: Parse Z as $(\mathsf{ct}, \{\mathsf{dlp}_i\}_{i \in [n]})$
2: $\mathsf{sct}_i = \{\}, \mathsf{chk} = d/n$
3: **for** $j = i \cdot \mathsf{chk}$ **to** $(i+1) \cdot \mathsf{chk} - 1$ **do**
4: $\quad \mathsf{ct}_j \leftarrow \mathsf{RLWE.Extract}(\mathsf{ct}, j)$ ▷ encrypts $\mathsf{Coeff}(\mathbf{m})[j]$
5: $\quad$ Parse ct_j as $(b_j, \vec{c}_j)$
6: $\quad \mathsf{sct}_i = \mathsf{sct}_i \cup b_j$
7: **end for**
8: $\mathsf{sct}_i = \mathsf{sct}_i \cup \vec{c}_{i \cdot \mathsf{chk}}$
9: **return** $Z_i := (\mathsf{sct}_i, \mathsf{dlp}_i)$

Algorithm 2 retrieves the i-th sub-puzzle $Z_i := (\mathsf{sct}_i, \mathsf{dlp}_i)$ from Z. Here, sct_i is composed of chk LWE-based ciphertexts encrypting messages from $\mathsf{Coeff}(\mathbf{m})[i \cdot \mathsf{chk}]$ to $\mathsf{Coeff}(\mathbf{m})[(i+1) \cdot \mathsf{chk} - 1]$ respectively. Meanwhile, $\mathsf{dlp}_i = f^{y_i}$ is a modular exponentiation where y_i also encodes the i-th chunk of $\mathsf{Coeff}(\mathbf{m})$. This algorithm exploits some tricks to significantly reduce the communication cost for transferring chk LWE-based ciphertexts in subsequent protocols. Observing the algorithm RLWE.Extract in Section III-B, the vector $\vec{c}_j$ always rotates the coefficients of $\mathsf{ct}[1]$. Therefore, transferring every $\vec{c}_j$ is redundant. It only needs to transfer

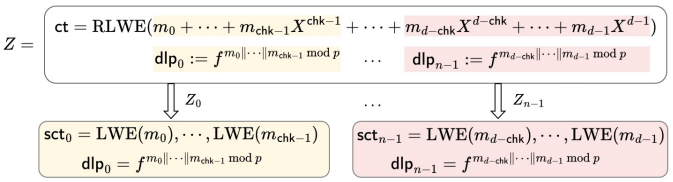


Fig. 3: The aggregated puzzle Z and sub-puzzles $\{Z_i\}_{i\in[n]}$.

$(b_{i\cdot\mathsf{chk}}, \cdots, b_{(i+1)\cdot\mathsf{chk}-1}, \vec{c}_{i\cdot\mathsf{chk}})$ and rotates $\vec{c}_{i\cdot\mathsf{chk}}$ to derive the rest of $\vec{c}_j$. Thus, the communication size for transferring sct_i can be reduced from $\mathsf{chk}\cdot(d+1)$ elements to $\mathsf{chk}+d$ elements of $\mathbb{Z}_q$. Based on the above description, we illustrate the aggregated puzzle and the sub-puzzles in Figure 3.

Algorithm 3 $\mathsf{PRandom}(\mathsf{ek}, Z_i, n)$

Parse Z_i as $(\mathsf{sct}_i, \mathsf{dlp}_i)$
Choose $\mathbf{m}' \leftarrow\$ \mathcal{R}_t, \|\mathbf{m}'\|_\infty \le t/6$
$\mathsf{ct}' \leftarrow \mathsf{RLWE.Enc}(\mathsf{ek}, \mathbf{m}')$
$\mathsf{sct}'_i \leftarrow \mathsf{PRetrieve}((\mathsf{ct}', \mathsf{dlp}_i), i, n)$ ▷ $\mathsf{dlp}_k (k \neq i)$ is dummy
$\mathsf{splice} = (1, 2^{\lceil \log t \rceil}, \cdots, 2^{(\mathsf{chk}-1)\lceil \log t \rceil})^{\mathrm{T}} \in \mathbb{Z}^{\mathsf{chk}}$
$y'_i = \langle \mathsf{splice}, \mathsf{Coeff}(\mathbf{m}')[i\cdot\mathsf{chk} : (i+1)\cdot\mathsf{chk} - 1] \rangle$
$\mathsf{dlp}'_i = \mathsf{dlp}_i \cdot f^{y'_i}$
$\mathsf{sct}'_i = \mathsf{sct}_i + \mathsf{sct}'_i$ over $\mathbb{Z}_q^{\mathsf{chk}+d}$
return $Z'_i := (\mathsf{sct}'_i, \mathsf{dlp}'_i)$

Algorithm 3 outputs a randomized sub-puzzle Z'_i. The randomization operation is homomorphically adding a fresh sub-puzzle to Z_i. Since the plaintext space of sct_i is $\mathbb{Z}_t^{\mathsf{chk}}$ and not exactly $\mathbb{Z}_p$ where p is the order of $\mathbb{G}$, to ensure simultaneous homomorphic computation of $\mathsf{sct}_i, \mathsf{dlp}_i$, we randomly choose polynomial plaintext $\mathbf{m}'$ (with $\|\mathbf{m}'\|_\infty \le t/6$) from $\mathcal{R}_t$. This restriction guarantees that no plaintext wrap-around occurs after two homomorphic additions of sct_i in our protocol, and thus the messages in sct'_i and dlp'_i are still consistent.

Algorithm 4 $\mathsf{PSolve}(\mathsf{dk}, Z_i)$

Parse Z_i as $(\mathsf{sct}_i := (b_0, \cdots, b_{\mathsf{chk}-1}, \vec{c}_0), \mathsf{dlp}_i)$
for $j = 0$ **to** $\mathsf{chk} - 1$ **do**
  $m_j \leftarrow \mathsf{LWE.Dec}(\mathsf{dk}, (b_j, \vec{c}_j))$
  $\vec{c}_{j+1} = (-\vec{c}_j[d-1], \vec{c}_j[0], \cdots, \vec{c}_j[d-2])$
end for
$\mathsf{splice} = (1, 2^{\lceil \log t \rceil}, \cdots, 2^{(\mathsf{chk}-1)\lceil \log t \rceil})^{\mathrm{T}} \in \mathbb{Z}^{\mathsf{chk}}$
$y_i = \langle \mathsf{splice}, (m_0, \cdots, m_{\mathsf{chk}-1}) \rangle$
Assert $\mathsf{dlp}_i = f^{y_i}$
return y_i

Algorithm 4 outputs the solution y_i corresponding to Z_i. In this algorithm, sct_i is extended to chk LWE-based ciphertexts by rotating $\vec{c}_0$ and is decrypted with dk.

C. Concrete Construction of HybridProof

In this section, we construct a novel NIZK, named HybridProof to prove the well-formedness of the aggregated puzzle Z.

Formally, the well-formedness of $Z = (\mathsf{ct}, \{\mathsf{dlp}_i\}_{i\in[n]})$ can be written as $\mathsf{R}_{\text{RLWE-DL}}$:

$$\left\{ \begin{array}{ll} & (\mathbf{A}, \mathsf{ct}, \{\mathsf{dlp}_i\}_{i\in[n]}, \mathsf{splice}, f; \vec{\mathbf{s}}) : \\ \hline (\mathrm{i}) & \mathbf{A}\cdot\vec{\mathbf{s}} = \mathsf{ct} \text{ over } \mathcal{R}_q \wedge \|\mathbf{m}\|_\infty \le t/6 \wedge \\ & \|(\mathbf{u}, \mathbf{e}_1, \mathbf{e}_2)\|_\infty \le \mathcal{B} \\ (\mathrm{ii}) & \mathsf{dlp}_0 = f^{\langle \mathsf{splice}, \mathsf{Coeff}(\mathbf{m})[0:\mathsf{chk}-1] \rangle} \wedge \cdots \wedge \\ & \mathsf{dlp}_{n-1} = f^{\langle \mathsf{splice}, \mathsf{Coeff}(\mathbf{m})[d-\mathsf{chk}:d-1] \rangle} \end{array} \right\}.$$

Here, $\mathbf{A} = \begin{pmatrix} \mathsf{ek}_0 & 1 & 0 & \Lambda \\ \mathsf{ek}_1 & 0 & 1 & 0 \end{pmatrix}$, $\vec{\mathbf{s}} = (\mathbf{u}, \mathbf{e}_1, \mathbf{e}_2, \mathbf{m})^{\mathrm{T}}$ express the RLWE-based encryption in Section III-B. As explained in Algorithm 3, $\mathbf{m}$ is chosen from $\mathcal{R}_t$ and the infinity norm is less than $t/6$. $\mathsf{R}_{\text{RLWE-DL}}$ is a hard relation with appropriate parameters (See Section VII-A).

The core idea of our protocol is to efficiently transform the two equations (i)&(ii) into two appropriate inner product relations when gluing equation (ii) to equation (i).

1) RLWE Part: The inner-product conversion of equation (i) can be divided into four steps.

Step 1. For the polynomial vector $\vec{\mathbf{s}} \in \mathcal{R}_q^4$ and the polynomial matrix $\mathbf{A} \in \mathcal{R}_q^{2\times 4}$, we have the following property over $\mathbb{Z}_q$: $\mathsf{Coeff}(\mathbf{A}\vec{\mathbf{s}} \bmod X^d+1) = \mathsf{rot}(\mathbf{A})\mathsf{Coeff}(\vec{\mathbf{s}})$. Therefore, proving the relation $\mathbf{A}\cdot\vec{\mathbf{s}} = \mathsf{ct}$ over $\mathcal{R}_q$ is equivalent to proving $\mathsf{rot}(\mathbf{A})\mathsf{Coeff}(\vec{\mathbf{s}}) = \mathsf{Coeff}(\mathsf{ct})$ over $\mathbb{Z}_q$. Defining $A = \mathsf{rot}(\mathbf{A}) \in \mathbb{Z}_q^{2d\times 4d}$, $\vec{s} = \mathsf{Coeff}(\vec{\mathbf{s}}) \in \mathbb{Z}_q^{4d}$ and $\vec{t} = \mathsf{Coeff}(\mathsf{ct}) \in \mathbb{Z}_q^{2d}$, the equation $\mathbf{A}\cdot\vec{\mathbf{s}} = \mathsf{ct}$ over $\mathcal{R}_q$ becomes $A\vec{s} = \vec{t}$ over $\mathbb{Z}_q$.

Step 2. We rewrite the above equation over $\mathbb{Z}$: $A\vec{s} + q\vec{r}_t = \vec{t}$. More precisely, $\vec{r}_t \in \mathbb{Z}^{2d}$ and its infinity norm is less than $(18d\mathcal{B} + dt + 6)/12$ when we use central representatives for coefficients in $\mathbb{Z}_q$. Next, for a prime $p > (18d\mathcal{B} + dt + 6)/12$, since there is no computation overflow, we have

$$A\vec{s} + q\vec{r}_t = \vec{t} \text{ over } \mathbb{Z}_p. \tag{1}$$

Step 3. We compress equation (1) by left-multiplying a randomly chosen vector $\vec{\gamma} = (\gamma_1, \ldots, \gamma_{2d})^{\mathrm{T}} \in (\mathbb{Z}_p^*)^{2d}$. This compression allows processing all the rows of (1) using a short proof for a single inner product relation. Therefore, with probability at least $1 - (2d)/p$, equation (1) is equivalent to $\vec{\gamma}^{\mathrm{T}}A\vec{s} + q\vec{\gamma}^{\mathrm{T}}\vec{r}_t = \vec{\gamma}^{\mathrm{T}}\vec{t}$ over $\mathbb{Z}_p$, which can be expressed as the following inner product equation.

$$\left\langle A^{\mathrm{T}}\vec{\gamma}, \vec{s} \right\rangle + \left\langle q\vec{\gamma}, \vec{r}_t \right\rangle = \vec{\gamma}^{\mathrm{T}}\vec{t} \text{ over } \mathbb{Z}_p. \tag{2}$$

Step 4. We replace the secret vectors in equation (2) with their binary representation. In detail, we parse A as (A_1, A_2) and $\vec{s}$ as $(\vec{s}_1, \vec{s}_2)^{\mathrm{T}}$, where $A_1 \in \mathbb{Z}_q^{2d\times 3d}, A_2 \in \mathbb{Z}_q^{2d\times d}$, $\vec{s}_1 \in \mathbb{Z}_q^{3d}$, $\vec{s}_2 \in \mathbb{Z}_q^d$. Thereafter, we get

$$\left\langle A_1^{\mathrm{T}}\vec{\gamma} \otimes \vec{2}_b, \mathsf{Bin}(\vec{s}_1) \right\rangle + \left\langle A_2^{\mathrm{T}}\vec{\gamma} \otimes \vec{2}_{b'}, \mathsf{Bin}(\vec{s}_2) \right\rangle + \left\langle q\vec{\gamma} \otimes \vec{2}_{b''}, \mathsf{Bin}(\vec{r}_t) \right\rangle = \vec{\gamma}^{\mathrm{T}}\vec{t} \text{ over } \mathbb{Z}_p,$$

where $\vec{2}_b = (1, 2, \ldots, 2^{b-2}, -2^{b-1})^{\mathrm{T}}$, $b = \lceil \log(\mathcal{B}) \rceil + 1$, $b' = \lceil \log(t/6) \rceil + 1$, $b'' = \lceil \log((18d\mathcal{B} + dt + 6)/12) \rceil$. Similarly, $\vec{2}_{b'}$ and $\vec{2}_{b''}$ are defined in the same way as $\vec{2}_b$. For the sake of clarity, we concatenate the vectors and define $\vec{v} = A_1^{\mathrm{T}}\vec{\gamma} \otimes \vec{2}_b \| A_2^{\mathrm{T}}\vec{\gamma} \otimes \vec{2}_{b'} \| q\vec{\gamma} \otimes \vec{2}_{b''}$, $\vec{a} = \mathsf{Bin}(\vec{s}_1) \| \mathsf{Bin}(\vec{s}_2) \| \mathsf{Bin}(\vec{r}_t)$. Therefore, we can derive an inner product equation:

$$\langle \vec{v}, \vec{a} \rangle = \vec{\gamma}^{\mathrm{T}}\vec{t}, \tag{3}$$

where $\vec{v}$ is a public vector over $\mathbb{Z}_p^l$ ($l = 3db + db' + 2db''$), and $\vec{a}$ is a binary and secret vector with length l.

Step 5. According to equation (3), we use additional equations to identify $\vec{a}$ is a binary vector. The fact that $\vec{a}$ is a binary vector means that there exists a complement and also secret vector $\vec{a}$, such that $\vec{a} \circ \vec{a} = \vec{0}$ and $\vec{a} + \vec{a} = \vec{1}$. The former equality is equivalent to $\langle \vec{\phi}, \vec{a} \circ \vec{a} \rangle = \langle \vec{\phi} \circ \vec{a}, \vec{a} \rangle = 0$ with probability

at least $1-l/p$, where $\vec{\phi}=(\phi_1,\cdots,\phi_l)\leftarrow\$\ (\mathbb{Z}_p^*)^l$. The latter equality is equivalent to $\langle\vec{\phi},\vec{a}+\vec{a}\rangle=\langle\vec{\phi},\vec{a}\rangle+\langle\vec{\phi}\circ\vec{a},\vec{1}\rangle=\langle\vec{\phi},\vec{1}\rangle$ with probability at least $1-l/p$. We merge both inner product equations into (3) and obtain

$$\left\langle\vec{v}+\vec{\phi}\circ\vec{a}+\psi\vec{\phi},\vec{a}+\psi\vec{1}\right\rangle=\vec{\gamma}^{\mathrm{T}}\vec{t}+\psi\left\langle\vec{v},\vec{1}\right\rangle+(\psi+\psi^2)\left\langle\vec{\phi},\vec{1}\right\rangle, \tag{4}$$

where $\psi\in\mathbb{Z}_p$ is a uniformly chosen element to separate the three inner product equations.

Taking the witnesses $\vec{a}$ and $\vec{\bar{a}}$, the prover should convince the verifier that the witnesses satisfy the inner-product equation (4). To do this, the prover generates two Pedersen vector commitments $w_1=\vec{h}^{\vec{a}}u^{o_1}$ and $w_2=\vec{g}^{\vec{\bar{a}}}u^{o_2}$ to vectors $\vec{a}$ and $\vec{\bar{a}}$ respectively and sends w_1,w_2 to the verifier. Then, the prover and verifier can construct a commitment C_{IPSS} to $\vec{v}_1=\vec{v}+\vec{\phi}\circ\vec{\bar{a}}+\psi\vec{\phi}$ and $\vec{v}_2=\vec{a}+\psi\vec{1}$ in three steps.

- First, we compute $C_{\mathsf{IPSS}}=w_1^{\eta}w_2$, where $\eta\in\mathbb{Z}_p$ is a randomly chosen element to separate the two vector commitments. With $\vec{h}'=\vec{h}^{\eta}$, C_{IPSS} commits to $\vec{a}$ and $\vec{\bar{a}}$ under $\vec{h}'$ and $\vec{g}$.
- Second, we use a standard trick to interpret C_{IPSS} commits to $\vec{\bar{a}}\circ\vec{\phi}$. Since $\vec{g}^{\vec{\bar{a}}}=(\vec{g}^{\vec{\phi}^{-1}})^{\vec{\phi}\circ\vec{\bar{a}}}=(\vec{g}')^{\vec{\phi}\circ\vec{\bar{a}}}$, the original commitment w_2 containing $\vec{\bar{a}}$ over $\vec{g}$ is reinterpreted as a commitment containing $\vec{\phi}\circ\vec{\bar{a}}$ over $\vec{g}'$. So is C_{IPSS}.
- Finally, the commitment C_{IPSS} to v_1 and v_2 can be derived by multiplying $(\vec{g}')^{\vec{v}+\psi\vec{\phi}}(\vec{h}')^{\psi}$ to C_{IPSS}, i.e., $C_{\mathsf{IPSS}}=w_1^{\eta}w_2(\vec{g}')^{\vec{v}+\psi\vec{\phi}}(\vec{h}')^{\psi}$.

With $x\overset{\text{def}}{=}\vec{\gamma}^{\mathrm{T}}\vec{t}+\psi\langle\vec{v},\vec{1}\rangle+(\psi+\psi^2)\langle\vec{\phi},\vec{1}\rangle$, $\vec{g}'\overset{\text{def}}{=}\vec{g}^{\vec{\phi}^{-1}}$, $\vec{h}'\overset{\text{def}}{=}\vec{h}^{\eta}$, $o\overset{\text{def}}{=}o_1\eta+o_2$, equation (4) can be proved by a zero-knowledge argument for the Inner Product of two Secret vectors [35], named IPSS. In detail, IPSS is a proof system for the following relation:

$$\mathsf{R}_{\mathsf{IPSS}}=\left\{\begin{array}{c}(\vec{g}',\vec{h}'\in\mathbb{G}^l,C_{\mathsf{IPSS}},u\in\mathbb{G},x\in\mathbb{Z}_p;\\ \vec{v_1},\vec{v_2}\in\mathbb{Z}_p^l,o\in\mathbb{Z}_p):\\ C_{\mathsf{IPSS}}=\vec{g}'^{\vec{v_1}}\vec{h}'^{\vec{v_2}}u^o\wedge x=\langle\vec{v_1},\vec{v_2}\rangle\end{array}\right\},$$

where l is the length of the secret vectors.

2) Discrete-logarithm Problems Part: Equation (ii) shares $\vec{\mathbf{s}}$ with equation (i), so we glue the proof of equation (ii) with that of equation (i).

First, we compress n group elements $\{\mathsf{dlp}\}_{i\in[n]}$ to one group element dlp, whose discrete-logarithm is the inner product of a public vector and a secret vector. To achieve this, we sample a random vector $\vec{\beta}\in(\mathbb{Z}_p^*)^n$ and compute the exponentiated aggregation

$$\mathsf{dlp}=\prod_{i\in[n]}\mathsf{dlp}_i^{\beta_i}=f^{\langle\vec{\beta}\otimes\mathsf{splice},\mathsf{Coeff}(\mathbf{m})\rangle}. \tag{5}$$

Besides, we should prove that the secret vector $\mathsf{Coeff}(\mathbf{m})$ in equation (5) is consistent with the witness in the RLWE part. Taking the vector commitment $w_1=\vec{h}^{\vec{a}}u^{o_1}$ generated in the RLWE part, we desire that equation (5) will share the same processed vector $\vec{a}$. Therefore, we equivalently replace $\mathsf{Coeff}(\mathbf{m})$ with $\vec{a}$ and prove:

$$w_1=\vec{h}^{\vec{a}}u^{o_1}\wedge\mathsf{dlp}=f^{\langle\vec{z},\vec{a}\rangle}, \tag{6}$$

where $\vec{z}=\mathbf{0}_{3db}\|(\vec{\beta}\otimes\mathsf{splice}\otimes\vec{2}_{b'})\|\mathbf{0}_{(l-3db-db')\times1}\in\mathbb{Z}_p^l$.

The two equations in (6) can be integrated as follows.

$$w_1\mathsf{dlp}^{\theta}=\vec{h}^{\vec{a}}f^{\theta\langle\vec{z},\vec{a}\rangle}u^{o_1}, \tag{7}$$

where $\theta\in\mathbb{Z}_p^*$ is uniformly chosen to separate w_1 and dlp. With $C_{\mathsf{IPSP}}\overset{\text{def}}{=}w_1\mathsf{dlp}^{\theta}$, $f'\overset{\text{def}}{=}f^{\theta}$, and $e\overset{\text{def}}{=}o_1$, equation (7) can be proved by a zero-knowledge argument for the Inner Product of a Secret vector and a Public vector [36], named IPSP. In detail, IPSP is a proof system for the following relation:

$$\mathsf{R}_{\mathsf{IPSP}}=\left\{\begin{array}{c}(\vec{h}\in\mathbb{G}^l,f',u,C_{\mathsf{IPSP}}\in\mathbb{G},\vec{z}\in\mathbb{Z}_p^l;\\ \vec{a}\in\mathbb{Z}_p^l,e\in\mathbb{Z}_p):\\ C_{\mathsf{IPSP}}=\vec{h}^{\vec{a}}f'^{\langle\vec{a},\vec{z}\rangle}u^e\end{array}\right\},$$

where l is the length of vectors $\vec{z}$ and $\vec{a}$.

Theorem 1. *If $p>(18d\mathcal{B}+dt+6)/12$, combined with* IPSS *and* IPSP, HybridProof *is complete, perfectly SHVZK, and knowledge sound under the discrete-log assumption.*

The detailed protocol, proof, and non-interactive transformation of HybridProof are presented in Appendix A.

D. UniCross: *Combining Everything Together*

Setup. Before the protocol execution, all entities, including the tumbler T, all senders $\{A_i\}_{i\in[n]}$ and receivers $\{B_i\}_{i\in[n]}$, can derive some public parameters.

- Each entity executes $(\mathsf{pk},\mathsf{sk})\leftarrow\mathsf{SIG}_j.\mathsf{Gen}(1^\lambda)$, where SIG_j is the digital signature scheme adopted on the entity's ledger $\mathcal{L}_j^{\mathsf{SIG}}$. We assume $\{\mathsf{SIG}_j\}_{j\in[K]}$ work on the same group $\mathbb{G}$. When the groups are not the same, we can still use this technique, assuming that there exists an efficiently computable bijection between them and utilizing the proof for discrete logarithm equality across groups as described in [37]. Correspondingly, $\Pi_{AS,j}$ is the adaptor signature with respect to SIG_j.
- Execute a trustless setup algorithm $\mathsf{NIZK.Setup}(1^\lambda)$ to generate crs for the tumbler and receivers.
- The tumbler T executes $(\mathsf{ek}_T,\mathsf{dk}_T)\leftarrow\mathsf{RLWE.Gen}(1^\lambda)$. Senders and receivers can derive the encryption key ek_T.
- (User-privacy-configuration) For users desiring privacy, amt is a value fixed by the tumbler, and a constant 1:1 exchange rate is assumed for simplicity. In practice, amt on each chain is defined to be economically equivalent to one unit of the pegged currency (e.g., 1 BTC), thereby standardizing the value of each cross-chain payment. For users without privacy requirements, the payment amount amt is arbitrary. Just to simplify the expression, we specify amt to be consistent with the privacy case. This simplification does not influence the security of our protocol.

(A) Target-chain deposit phase. The target-chain deposit phase can be divided into three steps, as described below.

A.1: Lock coins. Each B_i requests T to lock a coin with value amt. Then, for every receiver B_i, $i\in[n]$, the tumbler T learns B_i's native chain $\mathcal{L}_j^{\mathsf{SIG}}$ and deposits the coin to the shared address $\tilde{\mathsf{pk}}_{T,j}\&\tilde{\mathsf{pk}}_{B_i}$ controlled by T and B_i for a certain time tlock_{pp}. See A.1 in Figure 4.

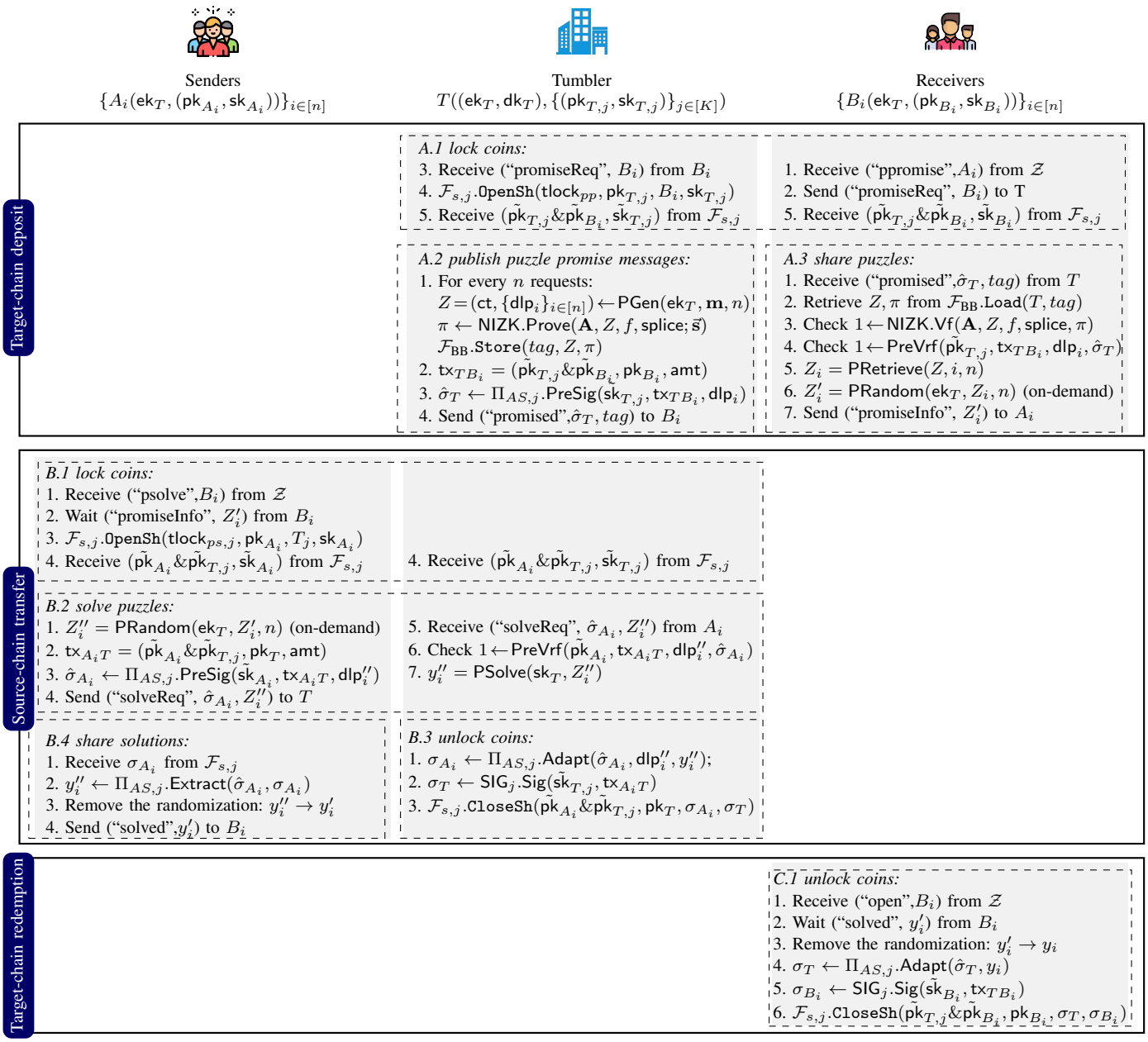


Fig. 4: The detailed construction of UniCross. The necessary parts of the $\mathcal{F}_{s,j}$'s output are explicitly written.

A.2: Publish puzzle promise messages. T generates an aggregated puzzle Z and produces a NIZK proof π proving the well-formedness of Z. Additionally, for each receiver B_i, T generates an adaptor signature $\hat{\sigma}_T$ over the agreed transaction tx_{TB_i} which transfers a coin from the shared address $\tilde{\mathsf{pk}}_{T,j}$&$\tilde{\mathsf{pk}}_{B_i}$ to B_i's address. A secure adaptor signature guarantees the pre-signature $\hat{\sigma}_T$ can be adapted to a valid signature for tx_{TB_i} if and only if the discrete log problem dlp_i is solved. This procedure is shown in A.2, Figure 4.

One-time communication via the bulletin board. T posts the puzzle-promise message Z, π to the bulletin board, and after that, each receiver B_i loads it from the bulletin board. It is worth noting that the InterPlanetary File System (IPFS) [38] always realizes the bulletin board. Excluding the inevitable peer-to-peer signature transfer, T only needs a *one-time communication* to publish the puzzle-promise message.

A.3: Share puzzles. When receiving the puzzle-promise message from T, each B_i can verify the pre-signature and the well-formedness of the puzzle. It is trivial for one receiver to learn its index i. At least it learns i by finding the received pre-signature is valid w.r.t dlp_i. Once B_i is convinced, it retrieves the i-th sub-puzzle $Z_i = (\mathsf{sct}_i, \mathsf{dlp}_i)$. If B_i desires privacy, it randomizes the sub-puzzle Z_i to derive Z'_i. Otherwise, B_i does not operate randomization and sets $Z'_i = Z_i$. Then, B_i shares the sub-puzzle Z'_i with the corresponding A_i and stops. This procedure is shown in A.3, Figure 4.

(B) Source-chain transfer phase. In the source-chain transfer phase, A_i pays a coin to T in exchange for the solution of Z'_i, which is divided into four steps below and shown in B.1-B.4, Figure 4.

B.1: Lock coins. Each sender A_i deposits a coin on its native blockchain $\mathcal{L}_j^{\mathsf{SIG}}$ to the shared address $\tilde{\mathsf{pk}}_{A_i}$&$\tilde{\mathsf{pk}}_{T,j}$ controlled

by A_i and T for a certain time $\mathsf{tlock}_{ps,j}$. Here, $\mathsf{tlock}_{pp} > \mathsf{tlock}_{ps,j}$ is required to ensure that A_i can react in time.

B.2: Solve puzzles. If A_i desires privacy, it randomizes the sub-puzzle Z_i' to derive Z_i''. Otherwise, A_i sets $Z_i'' = Z_i'$. A_i generates a pre-signature $\hat{\sigma}_{A_i}$ with respect to dlp_i'' for the transaction tx_{A_iT}, which transfers a coin from the shared address to T's address. Once convinced of the validity of $\hat{\sigma}_{A_i}$, T solves the sub-puzzle Z_i'' so as to adapt $\hat{\sigma}_{A_i}$ to a valid signature σ_{A_i}.

B.3: Unlock coins. After T locally generating the signature σ_T on tx_{A_iT}, signatures σ_{A_iT} and σ_T are used to close the shared address $\tilde{\mathsf{pk}}_{A_i}\&\tilde{\mathsf{pk}}_{T,j}$ and derive the coin.

B.4: Share solutions. After receiving σ_{A_i}, A_i can extract the solution y_i'' of puzzle Z_i'' with $\hat{\sigma}_{A_i}, \sigma_{A_i}$. Then, if privacy has been configured by A_i, it removes the randomization layer (an inverse operation of $\mathsf{PRandom}$) to obtain y_i', i.e.,

$$y_i' = y_i'' - \langle \mathsf{splice}, \mathsf{Coeff}(\mathbf{m}'')[i \cdot \mathsf{chk} : (i+1) \cdot \mathsf{chk} - 1] \rangle.$$

Otherwise, $y_i' = y_i''$. Thereafter, A_i sends y_i' to B_i. Here, A_i's successful extraction is guaranteed by the full extractability of the adaptor signature scheme.

(C) Target-chain redemption phase. Upon receiving the puzzle solution y_i' from A_i, receiver B_i proceeds as follows to redeem the coin from the tumbler.

C.1: Unlock coins. If privacy has been configured by B_i, it removes the randomization layer to obtain the solution y_i of the original puzzle Z_i. Otherwise, $y_i = y_i'$. With y_i, B_i can adapt the pre-signature $\hat{\sigma}_T$ to a valid signature σ_T. Then, same as T's operation in the source-chain transfer phase, B_i can close the shared address $\tilde{\mathsf{pk}}_{T,j}\&\tilde{\mathsf{pk}}_{B_i}$ and derive the coin.

VI. Security Analysis

Security proof in the LOE model. As demonstrated in [19], [21], to avoid general-purpose primitives, A^{2}L-like constructions are always proven secure in an idealized LOE model. In the LOE model, the key generation, encryption, decryption, and homomorphic addition operations of the encryption scheme are modeled by giving access to oracles instead of their corresponding algorithms. These oracles are formally defined in Appendix E-C.

Theorem 2. *The protocol* UniCross *in the* $(\{\mathcal{F}_{s,j}\}_{j\in[K]}, \mathcal{F}_{BB})$*-hybrid world UC realizes the ideal functionality* $\mathcal{F}_{ucc}$ *with respect to the global ideal functionalities* $\{\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j)\}_{j\in[K]}$.

Proof (Sketch). We sketch the proof by highlighting two main techniques for simulation. The complete proof is shown in Appendix C.

Technique 1 for simulation lies in the simulation of the case that the tumbler is honest and the sender or the receiver is corrupt. The major obstacle of such a simulation is that when the corrupt sender (or receiver) starts to interact with the tumbler, the simulator should call the $\mathcal{F}_{\text{ucc}}$.`TcDeposit` (or the $\mathcal{F}_{\text{ucc}}$.`ScTransfer`) interface. However, at this point, the simulator does not know which party it pays to (or is paid from). The key to solving this problem is applying the "temporary-placeholder/post-filling" technique. Concretely, our solution is to randomly choose a corrupt party as the temporary placeholder. Then, once the simulator observes the sender's output in the source-chain transfer phase, it can extract the actual pair of sender and receiver via the control of the encryption-related oracles. Then, the simulator corrects (i.e., post-filling) the payment pair by sending ("changePuzzle", $\cdot, \cdot$) to $\mathcal{F}_{\text{ucc}}$.

Technique 2 for simulation lies in the simulation of the case that the payment pair (i.e., the sender and the receiver) is honest and privacy is desired. In this case, the simulator cannot know the sender's and receiver's identities. For this issue, we let the simulator randomly appoint an honest user as the corresponding receiver of the honest sender requesting the target-chain deposit. Thus, the simulator can execute UniCross on behalf of the simulated payment pair. The effectiveness of this technique lies in a randomized sub-puzzle being indistinguishable from a fresh sub-puzzle.

VII. Implementation and Evaluation

To benchmark UniCross's performance, we develop a Go-based implementation, with the source code available at https://github.com/wwwchenke/unicross.git. All experiments are made on a 2.3GHz Laptop (Intel Core Ultra 9 185H) with 16 cores and 32GB RAM, running Ubuntu 22.04.1 LTS.

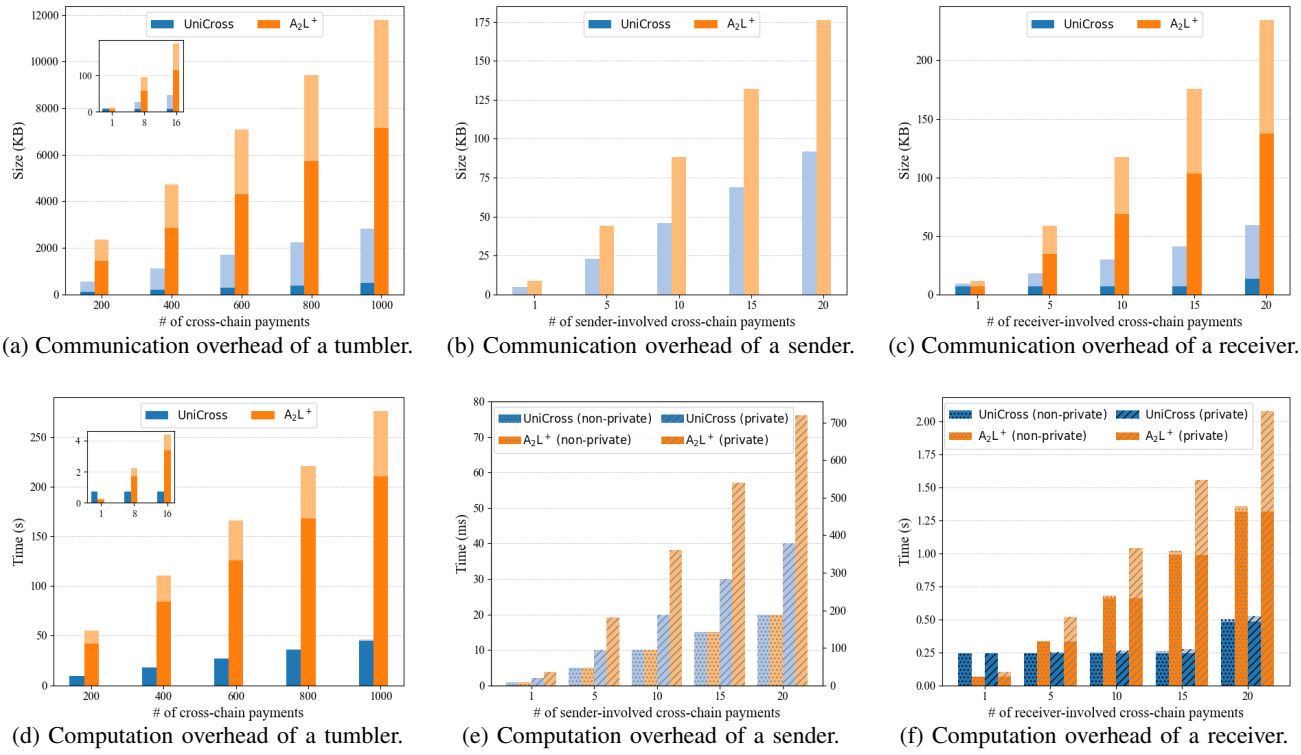
A. Implementation Details and Evaluation Methods

We provide a concrete implementation of our universal cross-chain scheme on the elliptic curve *Secp256k1* (used by typical blockchains like Bitcoin and Ethereum). Specifically, we assume the source chains employ ECDSA while the target chains adopt Schnorr signatures (and the corresponding adaptor signatures). Meanwhile, the bulletin board $\mathcal{F}_{\text{BB}}$ is implemented with InterPlanetary File System (IPFS) where tid is the content identifier (CID) [38]. For the RLWE-based encryption scheme, we set $\{\mathcal{B}, q, d, t\} = \{1, 2^{16}, 1024, 8\}$. Accordingly, for the aggregation of puzzles, we set $\mathsf{chk} = 64$ and batch size $n = 16$. Such parameter settings are for both security and best performance, which is explained in Appendix D.

In order to evaluate our scheme, we choose A^{2}L$^+$ [19] and UAS [17] for comparison. This is because A^{2}L$^+$ [19] has rigorous security guarantees and attains the highest efficiency among existing cross-chain payment protocols that provide optimal compatibility and on-demand privacy. At the same time, UAS is the only work designed for a many-to-many cross-chain payment setting, despite lacking on-demand privacy. Within such a setting, UAS exhibits the most significant throughput. Concerning comparison, it is worth noting that all three approaches incur an equivalent and minimal on-chain cost. Concretely, the on-chain cost is two transactions with standard signature verification scripts on both the source and target chains. Therefore, we focus our discussion on the additional resource consumption required by cross-chain solutions, omitting the overhead of generating on-chain transactions.

B. Universal Cross-Chain Solutions: UniCross vs. A^{2}L$^+$

1) Communication Costs: We measure the communication overhead associated with each role in UniCross and compare these results to A^{2}L$^+$, as shown in Figure 5a-5c.



(a) Communication overhead of a tumbler. (b) Communication overhead of a sender. (c) Communication overhead of a receiver.

(d) Computation overhead of a tumbler. (e) Computation overhead of a sender. (f) Computation overhead of a receiver.

Fig. 5: The performance of UniCross and comparison with A^2L^+. The number of payments involved by a sender/receiver ranges from 1 to 20. The number of tumbler-involved payments ranges from 1 to 1000. The dark-colored portion represents the operations to be batched, i.e., operations related to the aggregated puzzle and corresponding proof, while the light-colored portion indicates the rest of the operations. Note that, for Figure 5e, the right Y-axis represents the values of A^2L^+ (private).

Tumbler is the pivotal party in universal cross-chain payment protocols. To explore the scalability of the tumbler, we increase the number of cross-chain payments from 1 to 1000 and evaluate the corresponding communication overhead. The result is shown in Figure 5a. For instance, when processing a batch of 16 cross-chain payments, the tumbler in UniCross incurs only 45.11 KB of communication, which is 76% lower than that of A^2L^+. As highlighted by the dark-colored region in Figure 5a, batched puzzle-promise messages play a pivotal role in UniCross's scalability.

To illustrate how the number of cross-chain payments affects user costs, Figure 5b and Figure 5c show the communication overhead for each sender and receiver, respectively, as the number of payments increases from 1 to 20. Regardless of batching, UniCross reduces the sender's communication overhead by 48% relative to A^2L^+. This is because the LWE ciphertext extracted from an RLWE ciphertext in UniCross is shorter than the Castagnos-Laguillaumie ciphertext [39] based on class groups in A^2L^+.

On the receiver side, A^2L^+ processes each cross-chain payment using a single puzzle (and its corresponding proof), whereas UniCross employs an aggregated puzzle (and proof). Despite this, UniCross requires 15% less communication overhead per receiver compared to A^2L^+. As shown in Figure 5c, this advantage becomes more pronounced when scaling up the number of involved payments. This is because one aggregated puzzle in UniCross can support up to 16 individual payments.

2) Computation Costs: We further assess the tumbler's computational overhead for varying numbers of cross-chain payments (from 1 to 1000), as shown in Figure 5d. While UniCross requires more time to handle a single payment than A^2L^+, its batch-processing approach yields substantially better scalability as the number of payments grows. Since real-world tumblers often process a large volume of payments, UniCross ultimately outperforms A^2L^+ in practical deployments.

We evaluate the computation overhead of the sender and receiver by increasing the number of user-involved cross-chain payments. The result is shown in Figure 5e and Figure 5f, and the computation performance varies depending on whether privacy is required. On the sender side, when the sender does not desire privacy, both schemes rely on highly efficient algorithms involving only lightweight adaptor signature operations, which take under 1 ms per payment. When desiring privacy, in both schemes, an additional randomization of the ciphertext is needed. The sender in UniCross consumes 94% less computation overhead than that in A^2L^+. This is because the randomization in A^2L^+ is operated in class groups.

As for the receiver, Figure 5f illustrates that although UniCross imposes a slightly higher computational cost on the receiver for a single payment (less than 0.25 s), it becomes more efficient than A^2L^+ when processing multiple payments. This efficiency gain arises because UniCross supports batch verification of up to 16 cross-chain payments.

C. *Throughput Comparison:* UniCross *vs.* A^2L^+ & *UAS*

Although UAS [17] does not support on-demand privacy, its high performance in secure-only cross-chain scenarios makes it a suitable baseline for evaluating universal cross-chain solutions, including UniCross and A^2L^+. The corresponding results appear in Figure 6.

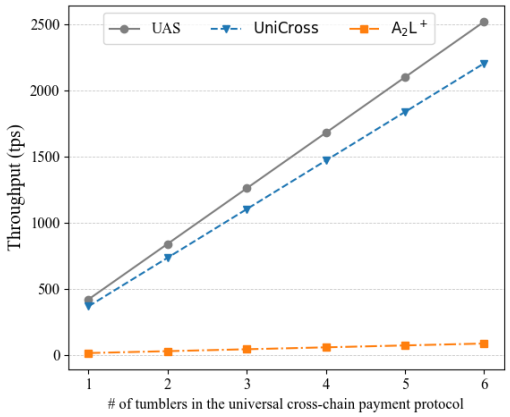


Fig. 6: The throughput comparison.

Concretely, we optimize the tumblers in UniCross and A^2L^+ by pre-generating (aggregated) puzzles and their respective proofs offline. Leveraging this optimization, UniCross achieves a throughput comparable to UAS, namely around 400 transactions per second (tps) for each tumbler. This demonstrates that the on-demand privacy offered by UniCross introduces negligible overhead to the overall protocol.

VIII. Conclusion

This paper proposed a universal cross-chain payment framework that is capable of transferring assets across a network of interconnected blockchains and providing on-demand privacy. Accordingly, this paper also proposes UniCross, a protocol to achieve all security, privacy, and usability goals for universal cross-chain. Specifically, UniCross achieves the state-of-the-art efficiency among potential universal cross-chain solutions via incorporating a batching procedure into A^2L^+ and removing the dependency on class groups. In detail, a novel puzzle construction for batching is leveraged. Further, to prove the well-formedness of such a puzzle, an efficient NIZK protocol, HybridProof, is designed. We define the security of a universal cross-chain solution and prove the security of UniCross within the UC framework. Also, we demonstrate the functionality and efficiency of UniCross in experiments.

References

[1] P2pb2b, https://p2pb2b.io, Jan. 2021, [Online].
[2] Bilaxy, https://bilaxy.com, Apr. 2018, [Online].
[3] Bkex, https://www.bkex.co, Jun. 2018, [Online].
[4] Lbank, https://www.lbank.info, Aug. 2015, [Online].
[5] G. Scaffino, L. Aumayr, Z. Avarikioti, and M. Maffei, "Glimpse:{On-Demand}{PoW} light client with {Constant-Size} storage for {DeFi}," in *USENIX Security 23*, 2023, pp. 733–750.
[6] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. Knottenbelt, "Xclaim: Trustless, interoperable, cryptocurrency-backed assets," in *SP 2019*. IEEE, 2019, pp. 193–210.
[7] G. Scaffino, L. Aumayr, M. Bastankhah, Z. Avarikioti, and M. Maffei, "Alba: The dawn of scalable bridges for blockchains," in *NDSS 2025*, 2025.
[8] M. Westerkamp and J. Eberhardt, "zkrelay: Facilitating sidechains using zksnark-based chain-relays," in *EuroS&PW*. IEEE, 2020, pp. 378–386.
[9] T. Xie, J. Zhang, Z. Cheng, F. Zhang, Y. Zhang, Y. Jia, D. Boneh, and D. Song, "zkbridge: Trustless cross-chain bridges made practical," in *ACM CCS 2022*, 2022, pp. 3003–3017.
[10] X. Jia, Z. Yu, J. Shao, R. Lu, G. Wei, and Z. Liu, "Cross-chain virtual payment channels," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 3401–3413, 2023.
[11] Y. Guo, M. Xu, X. Cheng, D. Yu, W. Qiu, G. Qu, W. Wang, and M. Song, "{zkCross}: A novel architecture for {Cross-Chain}{Privacy-Preserving} auditing," in *USENIX Security 24*, 2024, pp. 6219–6235.
[12] Z. Ge, J. Gu, C. Wang, Y. Long, X. Xu, and D. Gu, "Accio: Variable-amount, optimized-unlinkable and nizk-free off-chain payments via hubs," in *ACM CCS 2023*, 2023, pp. 1541–1555.
[13] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub," in *Network and distributed system security symposium*, 2017.
[14] "Hashed timelock contract," https://github.com/chatch/hashed-timelock-contract-ethereum, 2019.
[15] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," in *NDSS 2019*, 2019.
[16] P. Ni, A. Tian, and J. Xu, "Pipeswap: Forcing the timely release of a secret for atomic swaps across all blockchains," in *SP 2025*, 2025.
[17] S. A. Thyagarajan, G. Malavolta, and P. Moreno-Sanchez, "Universal atomic swaps: Secure exchange of coins across all blockchains," in *SP*. IEEE, 2022, pp. 1299–1316.
[18] E. Tairi, P. Moreno-Sanchez, and M. Maffei, "A^2l: Anonymous atomic locks for scalability in payment channel hubs," in *S&P*. IEEE, 2021, pp. 1834–1851.
[19] N. Glaeser, M. Maffei, G. Malavolta, P. Moreno-Sanchez, E. Tairi, and S. A. K. Thyagarajan, "Foundations of coin mixing services," in *ACM CCS 2022*, 2022, pp. 1259–1273.
[20] L. Hanzlik, J. L. Sri, A. Thyagarajan, and B. Wagner, "Sweep-uc: Swapping coins privately," in *SP 2024*. IEEE, 2024, pp. 3822–3839.
[21] P. Han, Z. Yan, L. T. Yang, and E. Bertino, "P2c2t: Preserving the privacy of cross-chain transfer," in *SP 2025*, 2025.
[22] Etairi, https://github.com/etairi/A2L, 2022.
[23] D. Schwartz, N. Youngs, A. Britto *et al.*, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, vol. 5, no. 8, p. 151, 2014.
[24] D. Mazieres, "The stellar consensus protocol: A federated model for internet-level consensus," *Stellar Development Foundation*, vol. 32, pp. 1–45, 2015.
[25] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.
[26] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient homomorphic conversion between (ring) lwe ciphertexts," in *ACNS*. Cham: Springer International Publishing, 2021, pp. 460–479.
[27] 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*. Springer International Publishing, 2021, pp. 635–664.
[28] W. Dai, T. Okamoto, and G. Yamamoto, "Stronger security and generic constructions for adaptor signatures," in *International Conference on Cryptology in India*. Springer, 2022, pp. 52–77.
[29] P. Gerhart, D. Schröder, P. Soni, and S. A. Thyagarajan, "Foundations of adaptor signatures," in *EUROCRYPT 2024*. Springer, 2024, pp. 161–189.
[30] X. Liu, I. Tzannetos, and V. Zikas, "Adaptor signatures: New security definition and a generic construction for np relations," in *Advances in Cryptology – ASIACRYPT 2024*, Singapore, 2025, pp. 168–193.
[31] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *FCS*. IEEE, 2001, pp. 136–145.
[32] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *TCC 2007*. Springer, 2007, pp. 61–85.
[33] S. Agrawal, C. Ganesh, and P. Mohassel, "Non-interactive zero-knowledge proofs for composite statements," in *CRYPTO 2018*. Springer, 2018, pp. 643–673.
[34] Z. Wu, S. Qi, X. Zhang, and Y. Deng, "Generic, fast and short proofs for composite statements," *Cryptology ePrint Archive*, 2024.
[35] R. del Pino, V. Lyubashevsky, and G. Seiler, "Short discrete log proofs for fhe and ring-lwe ciphertexts," in *PKC 2019*. Cham: Springer International Publishing, 2019, pp. 344–373.
[36] K. Gjøsteen, M. Raikwar, and S. Wu, "Pribank: confidential blockchain scaling using short commit-and-proof nizk argument," in *CT-RSA 2022*. Springer, 2022, pp. 589–619.
[37] S. Noether, "Discrete logarithm equality across groups," *MRL-0010. pdf*, 2018.
[38] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.
[39] G. Castagnos and F. Laguillaumie, "Linearly homomorphic encryption from," in *CT-RSA*. Springer, 2015, pp. 487–505.
[40] M. Abdalla, J. H. An, M. Bellare, and C. Namprempre, "From identification to signatures via the fiat-shamir transform: Minimizing assumptions

for security and forward-security," in *EUROCRYPT*. Springer, 2002, pp. 418–433.
[41] M. Rivinius, P. Reisert, D. Rausch, and R. Küsters, "Publicly accountable robust multi-party computation," in *SP 2022*. IEEE, 2022, pp. 2430–2449.
[42] X. Qin, S. Pan, A. Mirzaei, Z. Sui, O. Ersoy, A. Sakzad, M. F. Esgin, J. K. Liu, J. Yu, and T. H. Yuen, "Blindhub: Bitcoin-compatible privacy-preserving payment channel hubs supporting variable amounts," in *SP*. IEEE, 2023, pp. 2462–2480.
[43] J. Groth, "Rerandomizable and replayable adaptive chosen ciphertext attack secure cryptosystems," in *TCC*. Springer, 2004, pp. 152–170.
[44] M. F. Esgin, "Practice-oriented techniques in lattice-based cryptography," Ph.D. dissertation, Monash University, 2020.
[45] M. F. Esgin, R. Steinfeld, J. K. Liu, and D. Liu, "Lattice-based zero-knowledge proofs: new techniques for shorter and faster constructions and applications," in *CRYPTO*. Springer, 2019, pp. 115–146.
[46] M. F. Esgin, R. K. Zhao, R. Steinfeld, J. K. Liu, and D. Liu, "Matrict: efficient, scalable and post-quantum blockchain confidential transactions protocol," in *ACM CCS 2019*, 2019, pp. 567–584.
[47] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 315–334.

APPENDIX A
DESCRIPTION AND PROOF OF HYBRIDPROOF

Here we give a more detailed description of HybridProof.

Prover and Verifier's public inputs: $\mathbf{A} \in \mathcal{R}_q^{2\times 4}$, $\mathsf{ct} \in \mathcal{R}_q^2$, $\mathsf{splice} \in \mathbb{Z}^{\mathsf{chk}}$, $\{\mathsf{dlp}_i\}_{i\in[n]} \in \mathbb{G}$, $f, u \in \mathbb{G}$, $\vec{g}, \vec{h} \in \mathbb{G}^l$, $b, b', b'', l \in \mathbb{Z}$
Prover's secret inputs: $\vec{\mathbf{s}} = (\mathbf{u}, \mathbf{e_1}, \mathbf{e_2}, \mathbf{m})^{\mathrm{T}} \in \mathcal{R}_q^4$, where $\|\mathbf{m}\|_\infty \leq t/4$, $\|(\mathbf{u}, \mathbf{e_1}, \mathbf{e_2})\|_\infty \leq \mathcal{B}$

- Prover
  - computes
    - $A = \mathsf{rot}(\mathbf{A}) \in \mathbb{Z}^{2d\times 4d}$
    - $\vec{s} = \mathsf{Coeff}(\vec{\mathbf{s}}) \in \mathbb{Z}_t^{4d}$, $\vec{t} = \mathsf{Coeff}(\mathsf{ct}) \in \mathbb{Z}_q^{2d}$
    - $\vec{r}_t = \vec{t} - A\vec{s} \in \mathbb{Z}^{2d}$
  - integrates $\vec{a} = \mathsf{Bin}(\vec{s}) \| \mathsf{Bin}(\vec{r}_t)$
  - computes $\vec{a} = \vec{a} + \vec{1} \in \mathbb{Z}_2^l$ (XOR operation)
  - choose $o_1, o_2 \leftarrow\$ \mathbb{Z}_p$
  - computes and sends $w_1 = \vec{h}^{\vec{a}} u^{o_1}, w_2 = \vec{g}^{\vec{a}} u^{o_2}$ to Verifier
- Verifier
  - chooses and sends $\vec{\beta} \leftarrow\$ (\mathbb{Z}_p^*)^n$, $\vec{\gamma} \leftarrow\$ (\mathbb{Z}_p^*)^{2d}$, $\theta \leftarrow\$ \mathbb{Z}_p^*$, $\eta \leftarrow\$ \mathbb{Z}_p^*$, $\psi \leftarrow\$ \mathbb{Z}_p^*$, $\vec{\phi} \leftarrow\$ (\mathbb{Z}_p^*)^l$ to the Prover.
- Prover and Verifier compute
  - $\vec{g}' = \vec{g}^{\vec{\phi}^{-1}}, \vec{h}' = \vec{h}^{\eta}, f' = f^{\theta}$
  - parse A as (A_1, A_2), where $A_1 \in \mathbb{Z}_q^{2d\times 3d}$, $A_2 \in \mathbb{Z}_q^{2d\times d}$
  - $\vec{v} = A_1^{\mathrm{T}}\vec{\gamma} \otimes \vec{2}_b \| A_2^{\mathrm{T}}\vec{\gamma} \otimes \vec{2}_{b'} \| q\vec{\gamma} \otimes \vec{2}_{b''} \in \mathbb{Z}_p^l$
  - $\vec{z} = \vec{z} = \mathbf{0}_{3db} \| (\vec{\beta} \otimes \mathsf{splice} \otimes \vec{2}_{b'}) \| \mathbf{0}_{(l-3db-db')\times 1} \in \mathbb{Z}_p^l$
  - $C_{\mathsf{IPSS}} = w_1^{\eta} w_2 (\vec{g}')^{\vec{v}+\psi\vec{\phi}} \vec{h}'^{\psi}$
  - $C_{\mathsf{IPSP}} = w_1 (\prod_{i\in[n]} \mathsf{dlp}_i^{\beta_i})^{\theta}$
- Prover computes
  - $\vec{v}_1 = \vec{v} + \vec{\phi} \circ \vec{a} + \psi\vec{\phi}$, $\vec{v}_2 = \vec{a} + \psi\vec{1}$
  - $o = o_1\eta + o_2$, $e = o_1$
  - $x = \langle \vec{v}_1, \vec{v}_2 \rangle$
- Verifier computes
  - $x = \vec{\gamma}^{\mathrm{T}}\vec{t} + \psi \left\langle \vec{v}, \vec{1} \right\rangle + (\psi + \psi^2) \left\langle \vec{\phi}, \vec{1} \right\rangle$

The parties run $\mathsf{IPSS}(\vec{g}', \vec{h}', C_{\mathsf{IPSS}}, u, x; \vec{v}_1, \vec{v}_2, o)$ and $\mathsf{IPSP}(\vec{h}, f', u, C_{\mathsf{IPSP}}, \vec{z}; \vec{a}, e)$ and the verifier accepts if he accepts in IPSS and IPSP.

Fig. 7: The description of HybridProof.

Theorem 3. *If $p > (18d\mathcal{B} + dt + 6)/12$, combined with* IPSS *and* IPSP*, our protocol in Figure 7 is complete, perfectly SHVZK, and knowledge sound under the discrete-log assumption.*

Proof of Theorem 1. The completeness is direct from the discussion in Section V-C of our main manuscript. SHVZK follows immediately from the perfectly hiding property of vector Pedersen commitments and perfectly SHVZK property of IPSS and IPSP.

In terms of the knowledge soundness, we construct an extractor $\mathcal{X}$, which can either output witnesses $\vec{\mathbf{s}}^* \in \mathcal{R}_q^4$ satisfying relation $\mathsf{R}_{\text{RLWE-DL}}$, or a non-trivial discrete-log relation between generators of the group $\mathbb{G}$. Specifically, the extractor $\mathcal{X}$ runs $\mathcal{P}^*$ to derive w_1, w_2 and sends uniformly random challenges in the second move. Then $\mathcal{X}$ exploits the extractors $\mathcal{X}_1$ and $\mathcal{X}_2$ for the inner product proofs IPSS and IPSP respectively.

On the one hand, the extractor $\mathcal{X}_1$ can derive an opening $\vec{v}_1^*, \vec{v}_2^*, o^*$ of C_{IPSS}, i.e., $C_{\mathsf{IPSS}} = (\vec{g}')^{\vec{v}_1^*} (\vec{h}')^{\vec{v}_2^*} u^{o^*}$ such that $\langle \vec{v}_1^*, \vec{v}_2^* \rangle = x$. Since $C_{\mathsf{IPSS}} = w_1^{\eta} w_2 (\vec{g}')^{\vec{v}+\psi\vec{\phi}} \vec{h}'^{\psi}$, we get the opening $\vec{\phi} \circ \vec{a}^* = \vec{v}_1^* - \vec{v} - \psi\vec{\phi}$, $\vec{a}^* = \vec{v}_2^* - \psi\vec{1}$, o^* for $w_1^{\eta} w_2$ with generators $\vec{g}, \vec{h}', u$, such that $\langle \vec{v}_1^*, \vec{v}_2^* \rangle$ can be written as

$$\begin{aligned}
&\langle \vec{v}, \vec{a}^* \rangle + \left\langle \vec{v}, \psi\vec{1} \right\rangle + \left\langle \vec{\phi} \circ \vec{a}^*, \vec{a}^* \right\rangle \\
&+ \left\langle \vec{\phi} \circ \vec{a}^*, \psi\vec{1} \right\rangle + \left\langle \psi\vec{\phi}, \vec{a}^* \right\rangle + \left\langle \psi\vec{\phi}, \psi\vec{1} \right\rangle \\
=&\vec{\gamma}^{\mathrm{T}}\vec{t} + \psi \left\langle \vec{v}, \vec{1} \right\rangle + (\psi + \psi^2) \left\langle \vec{\phi}, \vec{1} \right\rangle .
\end{aligned}$$

The above equation is equivalent to

$$\langle \vec{v}, \vec{a}^* \rangle + \left\langle \vec{\phi}, \vec{a}^* \circ \vec{a}^* \right\rangle + \psi \left\langle \vec{\phi}, \vec{a}^* + \vec{a}^* - \vec{1} \right\rangle = \vec{\gamma}^{\mathrm{T}}\vec{t},$$

which can be interpreted as a multivariate polynomial P over $\mathbb{Z}_p$ in $2d + l + 1$ variables that evaluates to zero at $(\vec{\gamma}, \vec{\phi}, \psi)$. If the polynomial is the zero polynomial, it follows that $\vec{a}^* \circ \vec{a}^* = 0, \vec{a}^* + \vec{a}^* = \vec{1}, \langle \vec{v}, \vec{a}^* \rangle = \vec{\gamma}^{\mathrm{T}}\vec{t}$. So $\vec{a}^*$ is a binary vector. With $\vec{a}^*$, we can reconstruct the vector $\vec{v}^*$ and $\vec{r}_t^*$ which have components that fit in b, b', and b'' bits, respectively. Then, since $\langle \vec{v}, \vec{a}^* \rangle = \vec{\gamma}^{\mathrm{T}}\vec{t}$, it follows by inspection $\vec{\gamma}^{\mathrm{T}}(A\vec{s}^* + q\vec{r}^* - \vec{t}) = 0$ over $\mathbb{Z}_p$. The i_{th} component of $A\vec{s}^* + q\vec{r}^* - \vec{t}$ corresponds to the coefficient of γ_i of our multivariate polynomial P that we assume to be the zero polynomial. So $A\vec{s}^* + q\vec{r}^* - \vec{t} = 0$ over $\mathbb{Z}_p$. From our restriction on p, this equation is even true over $\mathbb{Z}$ and we can derive $A\vec{s}^* = \vec{t}$ over $\mathbb{Z}_q$ and then $\mathbf{A}\vec{\mathbf{s}}^* = \mathsf{ct}$.

It remains to consider the case where $P \neq 0$. In this case, the polynomial is of a total degree of 2. Consequently, it can be evaluated to zero at no more than $2p^{2d+l}$ points in $\mathbb{Z}_p^{2d+l+1}$ (this is just a counting version of the Schwartz-Zippel lemma). Now the extractor $\mathcal{X}$ reruns $\mathcal{P}^*$ but sends a uniform challenge $(\vec{\gamma}, \vec{\phi}, \vec{\psi}) \in \mathbb{Z}_p^{2d+l+1}$ from the set of non-roots of P. At least for a fraction of $1/2 - 2/p$ of the non-roots, $\mathcal{P}^*$ executes IPSS and outputs a valid transcript and $\mathcal{X}$ either get a non-trivial discrete-log relation or a new multivariate polynomial P' that is zero outside of the small set of roots of original P. So P' must be different from P. But then, since P and P' are in one-to-one correspondence to openings of the commitment $w_1^{\eta} w_2$, we derive two different openings and compute a non-trivial discrete-log relation.

On the other hand, the extractor $\mathcal{X}_2$ can derive an opening $\vec{a}^*, e^*$ of C_{IPSP} such that, $C_{\mathsf{IPSP}} = \vec{h}^{\vec{a}^*} f'^{\langle \vec{z}, \vec{a}^* \rangle} u^{e^*}$. Considering $C_{\mathsf{IPSP}} = w_1 (\prod_{i\in[n]} \mathsf{dlp}_i^{\beta_i})^{\theta}$, we below describe how

to extract the witnesses used by w_1 and $\{\mathsf{dlp}_i\}_{i\in[n]}$. At the beginning of the protocol, the extractor runs $\mathcal{P}^*$. Upon receiving w_1, w_2 from $\mathcal{P}^*$, the extractor runs $\mathcal{P}^*$ with challenges $\vec{\beta}^{(i)}$ $(i \in \{1,\cdots,n\})$ and $\theta^{(i,1)}$ and uses $\mathcal{X}_2$ to obtain a witness $\vec{a}^{(i,1)}, e^{(i,1)}$ such that $w_1(\prod_{j\in[n]} \mathsf{dlp}_j^{\beta_j^{(i)}})^{\theta^{(i,1)}} = \vec{h}^{\vec{a}^{(i,1)}} f^{\theta^{(i,1)}\langle \vec{z}^{(i)}, \vec{a}^{(i,1)}\rangle} u^{e^{(i,1)}}$.

Rewinding $\mathcal{P}^*$ with different challenges $\theta^{(i,2)}$, $\mathcal{X}$ derives $\vec{a}^{(i,2)}, e^{(i,2)}$ such that

$$w_1(\prod_{j\in[n]} \mathsf{dlp}_j^{\beta_j^{(i)}})^{\theta^{(i,2)}} = \vec{h}^{\vec{a}^{(i,2)}} f^{\theta^{(i,2)}\langle \vec{z}^{(i)}, \vec{a}^{(i,2)}\rangle} u^{e^{(i,2)}}.$$

Then, combining the above equations, we derive

$$(\prod_{j\in[n]} \mathsf{dlp}_j^{\beta_j^{(i)}})^{(\theta^{(i,1)}-\theta^{(i,2)})} = \vec{h}^{(\vec{a}^{(i,1)}-\vec{a}^{(i,2)})} f^{\theta^{(i,1)}\langle \vec{z}^{(i)}, \vec{a}^{(i,1)}\rangle - \theta^{(i,2)}\langle \vec{z}^{(i)}, \vec{a}^{(i,2)}\rangle} u^{e^{(i,1)}-e^{(i,2)}}.$$

It implies that $\vec{a}^{(i,1)} = \vec{a}^{(i,2)} \stackrel{\text{def}}{=} \vec{a}^{(i)}$, $e^{(i,1)} = e^{(i,2)} \stackrel{\text{def}}{=} e^{(i)}$. Otherwise, we get a non-trivial discrete-log relation between $\vec{h}, f, u$. Therefore, it holds that $w_1 = \vec{h}^{\vec{a}^{(i)}} u^{o_1^*}$, $\prod_{j\in[n]} \mathsf{dlp}_j^{\beta_j^{(i)}} = f^{\langle \vec{z}^{(i)}, \vec{a}^{(i)}\rangle}$, where $o_1^* = e^{(i)}$. Since we fix w_1, $\vec{a}^{(i)}, e^{(i)}$ with different i are the same. Otherwise, $\mathcal{X}$ derives a non-trivial discrete-log relation between $\vec{h}$ and u. So we define $\vec{a}^* = \vec{a}^{(i)}$ and $e^* = e^{(i)}$. Defining $\vec{\beta}^{(j)}$ as a vector with 1 in the j-th position and 0 elsewhere, we can reconstruct $\mathsf{dlp}_j = f^{\langle \mathsf{splice},\mathsf{Coeff}(\mathbf{m})[\mathsf{chk}\cdot j:\mathsf{chk}\cdot(j+1)-1]\rangle}$, where $\mathsf{Coeff}(\mathbf{m})$ is reconstructed from $\vec{a}^*$.

Non-interactive proof through Fiat-Shamir heuristic. So far, the presented zero-knowledge proof works as an interactive protocol with a logarithmic number of rounds. This protocol can be converted into a non-interactive protocol (with logarithmic proof length) using the Fiat-Shamir heuristic [40], i.e., by replacing all random challenges with hashes of the transcript up to a point.

APPENDIX B
DEFINITIONS OF IDEAL FUNCTIONALITIES

The functionality interacts with parties $P_1,\cdots,P_n$, the adversary $\mathcal{A}$. The functionality maintains a list Storage to record information.

Store: On input ("store", tag, x) from P_i, $\mathcal{F}_{\text{BB}}$ proceeds as follows.

1. If $(P_i, tag, \cdot) \in$ Storage, send ("fail") and abort.
2. Send ("store", P_i, x) to adversary $\mathcal{A}$
3. Upon receiving ("ok") from $\mathcal{A}$, store (P_i, tag, x) to Storage and reply ("ok") to P_i.

Load: On input ("load", P_j, tag) from P_i, $\mathcal{F}_{\text{BB}}$ proceeds as follows.

1. Wait until $(P_j, tag, x) \in$ Storage.
2. Send x to P_i.

Fig. 8: Bulletin board functionality $\mathcal{F}_{\text{BB}}$ [41].

The global functionality $\mathcal{L}_j^{\mathsf{SIG}}$ interacts with parties $P_1,\cdots,P_n$, the environment $\mathcal{Z}$, and other functionalities. It is parameterized by a digital signature scheme $\mathsf{SIG}_j = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Vf})$. The functionality maintains a list FrozenCoins, and a key value table bal.

Update: On input ("update", pk, c) from $\mathcal{Z}$, $\mathcal{L}_j^{\mathsf{SIG}}$ proceeds as follows.

1. Set $\mathsf{bal}[\mathsf{pk}] = c$.
2. Send ("updatedFund", pk, c) to every entity.

Pay: On input ("pay", $\mathsf{pk}_s, \mathsf{pk}_r, c, \mathsf{sk}_s$) from P_i, $\mathcal{L}_j^{\mathsf{SIG}}$ proceeds as follows.

1. If $c > \mathsf{bal}[\mathsf{pk}_s]$ or $c \notin \mathbb{R}_{\geq 0}$, send ("failNoFund") to P_i and abort.
2. If $(\mathsf{pk}_s, \mathsf{sk}_s)$ is not a valid key pair, send ("failInvalidKey") to P_i and abort.
3. Set $\mathsf{bal}[\mathsf{pk}_s] := \mathsf{bal}[\mathsf{pk}_s] - c, \mathsf{bal}[\mathsf{pk}_r] := \mathsf{bal}[\mathsf{pk}_r] + c$.
4. Send ("payed", $\mathsf{pk}_s, \mathsf{pk}_r, c$) to every entity.

Freeze: On input ("freeze", pk, c) from an ideal functionality with identifier id, $\mathcal{L}_j^{\mathsf{SIG}}$ proceeds as follows.

1. If $c > \mathsf{bal}[\mathsf{pk}]$ or $c \notin \mathbb{R}_{\geq 0}$, send ("failNoFund") to the ideal functionality with id and abort.
2. Set $\mathsf{bal}[\mathsf{pk}] := \mathsf{bal}[\mathsf{pk}] - c$ and append (id, c) to a list FrozenCoins.
3. Send ("frozen", id, pk, c) to every entity.

Unfreeze: On input ("unfreeze", pk, c) from an ideal functionality with identifier id, $\mathcal{L}_j^{\mathsf{SIG}}$ proceeds as follows.

1. If there is no entry (id, c') such that $c' \geq c$ in FrozenCoins, then send ("failNoFrozenFunds") to the ideal functionality with id and abort.
2. Replace (id, c') in FrozenCoins with $(\mathsf{id}, c' - c)$.
3. If $c' = c$, remove (id, c') from FrozenCoins.
4. Set $\mathsf{bal}[\mathsf{pk}] := \mathsf{bal}[\mathsf{pk}] + c$.
5. Send ("unfrozen", id, pk, c) to every entity.

Fig. 9: Ledger functionality $\mathcal{L}_j^{\mathsf{SIG}}$ [20].

The functionality $\mathcal{F}_{s,j}$ interacts with the functionality $\mathcal{L}_j^{\mathsf{SIG}}(\delta_j)$, parties $P_1,\cdots,P_n$. It holds a list OpenShared to record opened addresses.

OpenSh: On input ("openSh", tlock, $\mathsf{pk}_a, \mathsf{pk}_b, c, \mathsf{sk}_a$) from P_i, $\mathcal{F}_{s,j}$ proceeds as follows.

1. If $(\mathsf{pk}_a, \mathsf{sk}_a)$ is not a valid key pair, reply with ("failInvalidKey") and abort.
2. Generate $(\tilde{\mathsf{pk}}_a, \tilde{\mathsf{sk}}_a) \leftarrow \mathsf{SIG}_j.\mathsf{Gen}(1^\lambda)$, $(\tilde{\mathsf{pk}}_b, \tilde{\mathsf{sk}}_b) \leftarrow \mathsf{SIG}_j.\mathsf{Gen}(1^\lambda)$. Define $\tilde{\mathsf{pk}}_a\&\tilde{\mathsf{pk}}_b$ to be the shared address.
3. Call $\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j).\mathtt{Freeze}(\mathsf{pk}_a, c)$ within Δ_b rounds. Upon receiving ("failNoFunds") from $\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j)$, reply with ("failNoFunds") and abort. Otherwise, append $(\tilde{\mathsf{pk}}_a\&\tilde{\mathsf{pk}}_b, \mathsf{tlock}, c)$ to the list OpenShared.
4. Send ("openedSh", $\tilde{\mathsf{pk}}_a\&\tilde{\mathsf{pk}}_b, \tilde{\mathsf{sk}}_a, \mathsf{pk}_a, c$) to P_a who corresponds to pk_a, ("openedSh", $\tilde{\mathsf{pk}}_a\&\tilde{\mathsf{pk}}_b, \tilde{\mathsf{sk}}_b, \mathsf{pk}_a, c$) to P_b who corresponds to pk_b and ("openedSh", $\tilde{\mathsf{pk}}_a\&\tilde{\mathsf{pk}}_b, \mathsf{pk}_a, c$) to other entities.
5. After tlock rounds, if the entry $(\tilde{\mathsf{pk}}_a\&\tilde{\mathsf{pk}}_b, \mathsf{tlock}, c)$ is still in OpenShared, call $\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j).\mathtt{Unfreeze}(\mathsf{pk}_a, c)$ and remove it from OpenShared.

CloseSh: On input ("closeSh", $\tilde{\mathsf{pk}}_a\&\tilde{\mathsf{pk}}_b, \mathsf{pk}_{out}, c, \sigma_a, \sigma_b$) from P_i, $\mathcal{F}_{s,j}$ proceeds as follows.

1. If there is no entry $(\tilde{\mathsf{pk}}_a\&\tilde{\mathsf{pk}}_b, \cdot, c)$ in list OpenShared, reply with ("failNoSh") and abort.
2. Set $\mathsf{tx} = (\tilde{\mathsf{pk}}_a\&\tilde{\mathsf{pk}}_b, \mathsf{pk}_{out}, c)$, $v_a = \mathsf{SIG}_j.\mathsf{Ver}(\tilde{\mathsf{pk}}_a, \mathsf{tx}, \sigma_a)$, $v_b = \mathsf{SIG}_j.\mathsf{Ver}(\tilde{\mathsf{pk}}_b, \mathsf{tx}, \sigma_b)$. If $v_a \vee v_b = 0$, reply with ("failInvalidSig") and abort.
3. Call $\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j).\mathtt{Unfreeze}(\mathsf{pk}_{out}, c)$ within Δ_j rounds and remove the entry $(\tilde{\mathsf{pk}}_a\&\tilde{\mathsf{pk}}_b, \cdot, c)$ from OpenShared.
4. Send ("closedSharedAddress", $\tilde{\mathsf{pk}}_a\&\tilde{\mathsf{pk}}_b, \mathsf{pk}_{out}, c, \sigma_a, \sigma_b$) to every entity.

Fig. 10: Shared address functionality $\mathcal{F}_{s,j}$ [20].

APPENDIX C
SECURITY PROOF OF UniCross

Theorem 4. *The protocol* UniCross *in the* $(\{\mathcal{F}_{s,j}\}_{j\in[K]}, \mathcal{F}_{BB})$*-hybrid world UC realizes the ideal functionality* $\mathcal{F}_{ucc}$ *with respect to the global ideal functionalities* $\{\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j)\}_{j\in[K]}$.

Before proving Theorem 2, we define a property that will be useful for our proof.

Definition 2 (OM-CCA-UniCross)**.** *An encryption scheme* Π_E *is one-more CCA-UniCross-secure (OM-CCA-UniCross) if for any PPT adversary* $\mathcal{A}$ *and for any polynomial* $\mathsf{pl} = \mathsf{poly}(\lambda)$*, there is a negligible polynomial* negl *such that*

$$\Pr[\textit{OM-CCA-UniCross}^{\mathcal{A}}_{\Pi_E,\mathsf{pl}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

where $\textit{OM-CCA-UniCross}^{\mathcal{A}}_{\Pi_E,\mathsf{pl}}$ *is defined in Figure 11.*

$\mathsf{OM\text{-}CCA\text{-}UniCross}^{\mathcal{A}}_{\Pi_E,\mathsf{pl}}$

1 : $Q := 0$
2 : $(\mathsf{ek},\mathsf{dk}) \leftarrow \Pi_E.\mathsf{Gen}(1^\lambda)$
3 : $m_0,\cdots,m_{\mathsf{pl}} \leftarrow\$\ \{0,1\}^\lambda$
4 : **for** $i \in [\mathsf{pl}+1], \mathsf{sct}_i \leftarrow \Pi_E.\mathsf{Enc}_{\mathsf{ek}}(m_i)$
5 : $(m_0^*,\cdots,m_{\mathsf{pl}}^*) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{UniCross}}_{\mathsf{dk},\Pi_E,\Pi_{AS}}}(\mathsf{ek},\{\mathsf{sct}_i, f^{m_i}\}_{\in[\mathsf{pl}+1]})$
6 : **if** $\forall i \in [\mathsf{pl}+1], m_i^* = m_i \wedge Q \le \mathsf{pl}$ **then return** 1
7 : **else return** 0

$\mathcal{O}^{\mathsf{UniCross}}_{\mathsf{dk},\Pi_E,\Pi_{AS}}(\mathsf{pk},\mathsf{tx},\mathsf{dlp},\mathsf{sct},\hat{\sigma})$

1 : $m^* \leftarrow \Pi_E.\mathsf{Dec}(\mathsf{dk},\mathsf{sct})$
2 : **if** $\Pi_{AS}.\mathsf{PreVrf}(\mathsf{pk},\mathsf{tx},\mathsf{dlp},\hat{\sigma}) = 1 \wedge f^{m^*} = \mathsf{dlp}$
3 : $\quad Q := Q+1$
4 : $\quad$ **return** $\sigma \leftarrow \Pi_{AS}.\mathsf{Adapt}(\hat{\sigma}, m^*)$
5 : **else return** $\perp$

Fig. 11: OM-CCA-UniCross game.

The following lemma shows that an LOE scheme satisfies OM-CCA-UniCross, assuming the hardness of OMDL problem (see Section E-A).

Lemma 1. *Let Π_E be an LOE scheme. Assuming the hardness of OMDL, Π_E is OM-CCA-UniCross.*

Proof. In a nutshell, if $\mathcal{A}$ is a PPT adversary with a non-negligible advantage in the OM-CCA-UniCross, we can construct a PPT adversary $\mathcal{D}$ exploiting $\mathcal{A}$ to solve the OMDL problem with a non-negligible advantage.

Concerete ly, $\mathcal{D}$ is given $(h_0,\cdots,h_{\mathsf{pl}}) = (f^{m_0},\cdots,f^{m_{\mathsf{pl}}})$ by the challenger in the OMDL game. Then $\mathcal{D}$ proceeds as follows. First, it queries $\mathcal{O}^{\mathsf{Gen}}$ to derive $(\mathsf{ek},\mathsf{dk})$. Second, it samples $\mathsf{pl}+1$ uniform λ-bit strings $\{\mathsf{sct}_i^*\}_{i\in[\mathsf{pl}+1]}$, which are identically distributed to the outputs of $\mathcal{O}^{\mathsf{Enc}}$. It adds $(X_0,\mathsf{sct}_0^*),\cdots,(X_{\mathsf{pl}},\mathsf{sct}_{\mathsf{pl}}^*)$ into an initially empty table M, where X_i are random variables since $\mathcal{D}$ does not handle the corresponding plaintext of sct_i^*. Now it sends $\mathsf{ek},\{\mathsf{sct}_i,h_i\}_{i\in[\mathsf{pl}+1]}$ to the adversary $\mathcal{A}$.

Thereafter, $\mathcal{D}$ must simulate $\mathcal{A}$'s oracles. Any queries from $\mathcal{A}$ to the encryption related oracles, including $\mathcal{O}^{\mathsf{Enc}}$ and $\mathcal{O}^{\mathsf{Add}}$, and their corresponding responses are controlled by $\mathcal{D}$. Especially, the adversary $\mathcal{D}$ also adds the queried message and the answered ciphertext pairs into the table M. Upon receiving queries $\mathcal{O}^{\mathsf{ucc}}_{\mathsf{dk},\Pi_E,\Pi_{AS}}(\mathsf{pk}_i,\mathsf{tx}_i,\mathsf{dlp}_i,\mathsf{sct}_i,\hat{\sigma}_i)$ from $\mathcal{A}$, $\mathcal{D}$ acts in one of the following four ways.

- If these exists $j \in [\mathsf{pl}+1]$, $\mathsf{sct}_i = \mathsf{sct}_j^*$ and $\mathsf{dlp}_i = h_j$, it checks whether $\Pi_{AS}.\mathsf{PreVrf}(\mathsf{pk}_i,\mathsf{tx}_i,\mathsf{dlp}_i,\hat{\sigma}_i) = 1$. If not, it returns $\perp$; otherwise it queries $\mathsf{DL}(h_j)$ to get m_j and returns $\Pi_{AS}.\mathsf{Adapt}(\hat{\sigma}_i, m_j)$.
- If $\mathsf{sct}_i = \mathsf{sct}_j^*$ but $\mathsf{dlp}_i \neq h_j$, $\mathcal{D}$ sends $\perp$ to $\mathcal{A}$. This case means the adversary $\mathcal{A}$ queries this oracle with an unsolvable puzzle.
- If $(\cdot,\mathsf{sct}_i) \notin M$, $\mathcal{D}$ sends $\perp$ to $\mathcal{A}$. This case means the adversary $\mathcal{A}$ queries this oracle with a randomly sampled string sct_i.
- Otherwise, retrieve $(p_i,\mathsf{sct}_i) \in M$ where p_i is the plaintext corresponding to sct_i. It is worth noting that, by the linear-only property of the encryption scheme, p_i is a polynomial in $X_0,\cdots,X_{\mathsf{pl}}$ with $\deg(p_i) \le 1$.
  1) If $\deg(p_i) = 0$, p_i is a constant. It checks whether $\mathsf{dlp}_i = f^{p_i}$ and $\Pi_{AS}.\mathsf{PreVrf}(\mathsf{pk}_i,\mathsf{tx}_i,\mathsf{dlp}_i,\hat{\sigma}_i) = 1$. If not, it returns $\perp$; otherwise, it returns $\Pi_{AS}.\mathsf{Adapt}(\hat{\sigma}_i,p_i)$ to $\mathcal{A}$.
  2) If $\deg(p_i) = 1$, assume $p_i := \alpha_0 X_0 + \cdots + \alpha_{\mathsf{pl}} X_{\mathsf{pl}} + \alpha_{\mathsf{pl}+1}$. It checks whether $\Pi_{AS}.\mathsf{PreVrf}(\mathsf{pk}_i,\mathsf{tx}_i,\mathsf{dlp}_i,\hat{\sigma}_i) = 1$ and $\mathsf{dlp}_i = f^{\alpha_{\mathsf{pl}+1}} \cdot \Pi_{k=0}^{\mathsf{pl}} h_k^{\alpha_k}$. If not, it returns $\perp$; otherwise, it queries $\mathsf{DL}(\mathsf{dlp}_i)$ to get x_i and returns $\Pi_{AS}.\mathsf{Adapt}(\hat{\sigma}_i,x_i)$ to $\mathcal{A}$.

Observe that $\mathcal{D}$ makes at most pl queries to $\mathsf{DL}(\cdot)$ for answering pl successful $\mathcal{O}^{\mathsf{ucc}}_{\mathsf{dk},\Pi_E,\Pi_{AS}}$ queries. Here, pl is the upper bound of the discrete log queries in the OMDL game. If $\mathcal{A}$ outputs $(m_0^*,\cdots,m_{\mathsf{pl}}^*)$ and wins the OM-CCA-UniCross game, $\mathcal{D}$ outputs the same values to the challenger and wins in the OMDL game. Since $\mathcal{A}$ can win the OM-CCA-UniCross game with a non-negligible advantage, $\mathcal{D}$ wins the OMDL game with the non-negligible advantage which violates the OMDL assumption. Therefore, there exists no such $\mathcal{A}$. □

To prove Theorem 2, we should construct a simulator $\mathcal{S}$ for any adversary $\mathcal{A}$, such that the environment $\mathcal{Z}$ cannot distinguish the real and ideal worlds. First, we describe what the simulator $\mathcal{S}$ should simulate in the ideal world. Then, we present a detailed description of $\mathcal{S}$. Finally, we prove the indistinguishability between the ideal and real worlds. As [19], [21], [42], we prove the security by exploiting the LOE model, which is described in Section E-C. Specifically, we abstract the LWE-based encryption scheme as the LOE scheme Π_E as described in [19], [21], [42], [43], in which the key generation, encryption, decryption, and homomorphic addition operations are modeled by giving access to oracles instead of their corresponding algorithms. Accordingly, the RLWE-based encryption scheme shares the same key generation oracle. Encryption, decryption, and homomorphic addition operations are abstracted as d queries to the LOE scheme. That is because one RLWE-based ciphertext packages d LWE-based ciphertexts.

Simulation ideas. On a macro level, the key to the proof of security in the UC model is to prove the indistinguishability of two worlds, i.e., the ideal world and the real world. Specifically, in the ideal world, we specify an ideal functionality $\mathcal{F}_{\text{ucc}}$ to state what a protocol is supposed to do, which can be viewed as a trusted third party. The ideal functionality $\mathcal{F}_{\text{ucc}}$ receives inputs from or sends outputs to parties in $\mathcal{P}$ and a special party $\mathcal{S}$ who plays the role of the adversary in the ideal world. In the real world, parties in $\mathcal{P}$ and the real adversary $\mathcal{A}$ execute the protocol Π. The environment $\mathcal{Z}$ is defined to provide inputs to the parties in $\mathcal{P}$ and $\mathcal{A}$ and collects their outputs in both worlds. We say that a protocol Π securely implements the ideal functionality if the outputs of $\mathcal{Z}$ running either in the real world or in the ideal world are computationally indistinguishable.

Simulation settings. On the one hand, in the ideal world, $\mathcal{S}$ takes the roles of the honest parties, the ideal functionality

$\mathcal{F}_{s,j}$ and $\mathcal{F}_{\text{BB}}$ to interact with the corrupt parties. Furthermore, the simulator simulates the oracles of the encryption scheme in the LOE model. Since corrupt parties cannot observe the secure communication between honest parties, the simulator $\mathcal{S}$ does not have to simulate this part. On the other hand, the simulator $\mathcal{S}$ extracts the actual operations of the corrupt parties to guarantee that the environment $\mathcal{Z}$ cannot distinguish between the ideal and the real world with what it obtains.

Simulation procedure. The detailed construction of the simulator $\mathcal{S}$ is defined in Figure 12.

Detailed indistinguishability proof. We prove that the real-world execution is indistinguishable from the ideal world by the environment through a sequence of hybrid executions.

$\mathcal{H}_0$: This hybrid is the real world execution with environment $\mathcal{Z}$. It keeps table M, list Puzzle, and Hpair as the simulator $\mathcal{S}$ in Figure 12, but does not use them yet.

$\mathcal{H}_1$: The execution is identical to $\mathcal{H}_0$ except that the honestly generated NIZK proof π is replaced with a simulated proof $\pi \leftarrow \mathsf{NIZK.Sim}(\tau, \mathbf{A}, Z, f, \mathsf{splice})$ when T is honest.

$\mathcal{H}_2$: The execution is identical to $\mathcal{H}_1$ except that $\mathcal{H}_2$ controls the encryption-related oracles, adds the quires into a table M, and has one more check as follows. In the case that T is corrupt and A_{i^*}, B_{i^*} are honest, if the puzzle-promise message from T is valid, decrypt $\{\mathsf{sct}_i\}_{i\in[n]}$ by looking up the table M to derive $\{msg_i\}_{i\in[n]}$ and check, for $i \in [n]$, if $f^{msg_i} = \mathsf{dlp}_i$. If not, abort.

$\mathcal{H}_3$: The execution is identical to $\mathcal{H}_2$ except that $\mathcal{H}_3$ has one more check as follows. In the cases that T is corrupt and A_{i^*} is honest, after the $\Pi_{\text{AS},j}$.Extract algorithm, check if $\mathsf{dlp}'_{i^*} = f^{y'_{i^*}}$. If not, abort.

$\mathcal{H}_4$: The execution is identical to $\mathcal{H}_3$ except that $\mathcal{H}_4$ has one more check as follows. In the cases that B_{i^*} is honest, after adapting the pre-signature to a signature in the open phase, check if the signature is valid. If not, abort.

$\mathcal{H}_5$: The execution is identical to $\mathcal{H}_4$ except that $\mathcal{H}_5$ maintains a list Puzzle which records the state (either solved or unsolved) of puzzles and has one more check as follows. If B_{i^*} redeems a coin from the shared address but $(B_{i^*}, \top) \notin$ Puzzle, abort.

Lemma 2. *$\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable for all PPT distinguisher $\mathcal{Z}$, if the NIZK is zero-knowledge.*

Proof. Suppose there exists a PPT distinguisher $\mathcal{Z}$ that can distinguish $\mathcal{H}_0$ and $\mathcal{H}_1$ with a non-negligible probability. In that case, we can construct an adversary $\mathcal{B}_1$ to break the zero-knowledge property of NIZK. Specifically, in the cases that the tumbler is honest, instead of generating the zero-knowledge proof, $\mathcal{B}_1$ sends $(\mathbf{A}, Z, f, \mathsf{splice}, \{y_i\}_{i\in[n]})$ to its challenger and receive π as the reply. If π is generated by NIZK.Prove, $\mathcal{B}_1$ simulates $\mathcal{H}_0$ for $\mathcal{Z}$. If π is generated by NIZK.Sim, $\mathcal{B}_1$ simulates $\mathcal{H}_1$ for $\mathcal{Z}$. If $\mathcal{Z}$ wins the game with a non-negligible advantage, then $\mathcal{B}_1$ can break the zero-knowledge property of NIZK with the same advantage, which violates the zero-knowledge property of NIZK. Therefore, there exists no such distinguisher $\mathcal{Z}$. □

Lemma 3. *$\mathcal{H}_1$ and $\mathcal{H}_2$ are indistinguishable for all PPT distinguisher $\mathcal{Z}$, if the NIZK is knowledge sound.*

Proof. The difference between $\mathcal{H}_1$ and $\mathcal{H}_2$ is whether the simulation aborts or not after the check. Therefore, it suffices to prove that the probability that the simulation aborts after the check is negligible. If the simulation aborts in $\mathcal{H}_2$ with a non-negligible advantage, we can construct an adversary $\mathcal{B}_2$ to break the soundness property of NIZK. Specifically, $\mathcal{B}_2$ proceeds the hybrid world $\mathcal{H}_2$ for $\mathcal{Z}$. When the corrupt T outputs a valid puzzle-promise message, $\mathcal{B}_2$ sends the statement and the corresponding proof π to its challenger. If the simulation aborts in $\mathcal{H}_2$ with a non-negligible advantage, $\mathcal{B}_2$ breaks the knowledge soundness property of NIZK with a non-negligible advantage. Therefore, the simulation aborts in $\mathcal{H}_2$ with negligible advantage. □

Lemma 4. *$\mathcal{H}_2$ and $\mathcal{H}_3$ are indistinguishable for all PPT distinguisher $\mathcal{Z}$, if the adaptor signature Π_{AS} is full extractable.*

Proof. Proving this lemma amounts to proving that the probability that the simulation aborts in $\mathcal{H}_3$ is negligible. If the simulation aborts in $\mathcal{H}_3$ with a non-negligible advantage, we can construct an adversary $\mathcal{B}_3$ to break the full extractability property of $\Pi_{AS,j}$. Specifically, receiving pk from its challenger, $\mathcal{B}_3$ proceeds as $\mathcal{H}_3$ for $\mathcal{Z}$ and replaces a shared address $\tilde{\mathsf{pk}}_{A_k}$ with pk where A_k is honest. When the simulation needs to execute $\Pi_{AS,j}.\mathsf{PreSig}(\tilde{\mathsf{sk}}_{A_k}, \mathsf{tx}_{A_k,T}, \mathsf{dlp}_k)$, $\mathcal{B}_3$ sends $(\mathsf{tx}_{A_k,T}, \mathsf{dlp}_k)$ to its challenger and derives $\hat{\sigma}_{A_k}$. When the event " after the $\Pi_{\text{AS},j}$.Extract algorithm, $\mathsf{dlp}'_{i^*} \neq f^{y'_{i^*}}$" occurs, and $i^* = k$, $\mathcal{B}_3$ sends $(\mathsf{tx}_{A_k,T}, \sigma_{A_{i^*}})$ that is used to its challenger. If the simulation aborts in $\mathcal{H}_3$ with a non-negligible advantage, $\mathcal{B}_3$ breaks the full extractability property of Π_{AS} with a non-negligible advantage. Therefore, the simulation aborts in $\mathcal{H}_3$ with negligible advantage. □

Lemma 5. *$\mathcal{H}_3$ and $\mathcal{H}_4$ are indistinguishable for all PPT distinguisher $\mathcal{Z}$, if the adaptor signature Π_{AS} is pre-signature adaptable and pre-verify soundness.*

Proof. It directly follows from the pre-signature adaptability and pre-verify soundness properties of Π_{AS}, which guarantees that the simulation cannot abort in the case described in $\mathcal{H}_4$. □

Lemma 6. *$\mathcal{H}_4$ and $\mathcal{H}_5$ are indistinguishable for all PPT distinguisher $\mathcal{Z}$, if the adaptor signature Π_{AS} is full extractable and Π_E is OM-CCA-UniCross secure.*

Proof. For simplicity, we consider the case that all senders and receivers are corrupt while the tumbler is honest. The other cases are similar. If the simulation aborts in $\mathcal{H}_5$ with a non-negligible advantage, we can construct an adversary $\mathcal{B}_3$ to break either the OM-CCA-UniCross property of Π_E or the full extractability property of Π_{AS}. We now specify $\mathcal{B}_3$.

Receiving ek, $\{\mathsf{sct}_i, f^{y_i}\}_{i\in[n-1]}$ from the challenger of the OM-CCA-UniCross game, $\mathcal{B}_4$ uses $\{\mathsf{sct}_i, f^{y_i}\}_{i\in[n-1]}$ as $n-1$ puzzles and generates $n-1$ pre-signatures using $\Pi_{AS,j}.\mathsf{PreSig}(\tilde{\mathsf{sk}}_{T,j}, \mathsf{tx}_{TB_i}, f^{y_i})$ where $i \in [n-1]$. Receiving pk from the challenger of the unforgeability game, $\mathcal{B}_4$ replaces a shared address $\tilde{\mathsf{pk}}_{T,j}$ of T with pk. When the simulation needs to execute $\Pi_{AS,j}.\mathsf{PreSig}(\tilde{\mathsf{sk}}_{T,j}, \mathsf{tx}_{TB_{n-1}}, \mathsf{dlp}_{n-1})$, $\mathcal{B}_4$ sends $\mathsf{tx}_{TB_{n-1}}$ to its challenger, derives $\hat{\sigma}_T$ and dlp_{n-1}, and chooses $\mathsf{sct}_{n-1} \leftarrow\!\$\ \{0,1\}^\lambda$.

If the simulation aborts in $\mathcal{H}_5$ and $i^* \in [n-1]$, $\mathcal{B}_4$ retrieves the adapted signature σ_T and executes $y_i \leftarrow \Pi_{AS,j}.\mathsf{Extract}(\tilde{\sigma}_T, \sigma_T)$ for $i \in [n-1]$. Then, $\mathcal{B}_4$ outputs $\{y_i\}_{i\in[n-1]}$ to its challenger of the OM-CCA-UniCross game.

If the simulation aborts in $\mathcal{H}_5$ and $i^* = n-1$, $\mathcal{B}_4$ retrieves the adapted signature σ_T and sends $(\mathsf{tx}_{TB_{n-1}}, \sigma_T)$ to its challenger of the full extractability game.

If the simulation aborts in $\mathcal{H}_5$ with a non-negligible advantage, $\mathcal{B}_4$ either breaks the OM-CCA-UniCross property of Π_E or the unforgeability property of Π_{AS}. Therefore, the simulation aborts in $\mathcal{H}_5$ with negligible advantage. □

Lemma 7. *For all PPT distinguisher $\mathcal{Z}$, $\mathcal{H}_5$ and the ideal world are statistically indistinguishable.*

Proof. There are two differences between the $\mathcal{H}_5$ and the ideal world. First, in the ideal world, the simulator $\mathcal{S}$ simulates the ideal functionality $\mathcal{F}_s$. Second, the simulator $\mathcal{S}$ in the ideal world maintains a list Hpair recording the receiver identity B_{i^*} in the case that A_{i^*} and B_{i^*} are honest and T is corrupt, and change the execution for such case. Namely, in the puzzle-solving phase, instead of randomizing the puzzle from its corresponding receiver B_{i^*}, A_{i^*} randomizes the puzzle of the first identity B_j in Hpair and removes B_j from Hpair. Since the randomized puzzle of B_{i^*} and that of B_j have the same distribution, $\mathcal{H}_5$ is identical to the ideal world. □

Therefore, we conclude that for any PPT adversary $\mathcal{A}$, we can construct a simulator, such that $\mathcal{Z}$ cannot distinguish the real world and the ideal world.

(A) **Target-chain deposit**: Distinguish the following cases,

1. **Honest T, Corrupt B_{i^*}, Honest or Corrupt A_{i^*}.** Upon receiving ("promiseReq", B_{i^*}) from B_{i^*} on behalf of T:
   - Send ("ppromise", A'_{i^*}, $\mathsf{IsPrivate}_r$) to $\mathcal{F}_{\text{ucc}}$ on behalf of B_{i^*}, where A'_{i^*} is randomly chosen from corrupt parties and $\mathsf{IsPrivate}_r = 1$.
   - Upon receiving ("ppromise", A'_{i^*}, B_{i^*}, $\mathsf{IsPrivate}_s$) from $\mathcal{F}_{\text{ucc}}$, simulate $\mathcal{F}_{s,j}$ and execute operations on behalf of T except that the honestly generated NIZK proof π is replaced with a simulated proof $\pi \leftarrow \mathsf{NIZK.Sim}(\tau, \mathbf{A}, Z, f, \mathsf{splice})$.
   - Append $(B_{i^*}, \perp, \mathsf{IsPrivate}_r)$ to the list Puzzle, and stop.
2. **Corrupt T, Honest B_{i^*}.** Distinguish the following cases.
   Case A_{i^*} is honest. Upon receiving ("ppromise", $\diamond/A_{i^*}$, B_{i^*}, $\mathsf{IsPrivate}_r$) from $\mathcal{F}_{\text{ucc}}$:
   - Send ("promiseReq", B_{i^*}) to T on behalf of B_{i^*}.
   - Simulate the ideal functionality $\mathcal{F}_{s,j}$ for T.
   - Upon receiving ("promiseRes") from $\mathcal{F}_{\text{ucc}}$, if no promise message from T within 1 round, send ("invalidPromise") to $\mathcal{F}_{\text{ucc}}$ and abort.
   - Execute the steps on behalf of B_{i^*} except that the simulator does not send ("promisedInfo", Z'_{i^*}). If abort during the execution, send ("invalidPromise") to $\mathcal{F}_{\text{ucc}}$.
   - When the promise message from T is valid, decrypt $\{\mathsf{sct}_i\}_{i\in[n]}$ by looking up the table M to derive $\{msg_i\}_{i\in[n]}$ and check, for $i \in [n]$, whether $\mathsf{dlp}_i = f^{msg_i}$. If not, abort.
   - If $\mathsf{IsPrivate}_r = 1$, append $(B_{i^*}, \perp, 1)$ to the list Puzzle and append B_{i^*} to the list Hpair. If $\mathsf{IsPrivate}_r = 0$, append $(B_{i^*}, \perp, 0)$ to the list Puzzle.

   Case A_{i^*} is corrupt. Upon receiving ("ppromise", A_{i^*}, B_{i^*}, $\mathsf{IsPrivate}_r$) from $\mathcal{F}_{\text{ucc}}$: Proceed as the case where A_{i^*} is honest except that the simulator sends ("promiseInfo", Z'_{i^*}) to A_{i^*} on behalf of B_{i^*} but doesn't update the list Hpair.
3. **Corrupt T, B_{i^*}, Honest A_{i^*}.** Upon receiving $\mathcal{F}_{s,j}.\mathtt{OpenSh}(\mathsf{tlock}_{pp}, \mathsf{pk}_T, B_{i^*}, \mathsf{amt}, \mathsf{sk}_T)$ from T and $(\mathsf{pk}_T, \mathsf{sk}_T)$ is a valid key pair:
   - Send ("ppromise", A'_{i^*}, $\mathsf{IsPrivate}_r$) to $\mathcal{F}_{\text{ucc}}$ on behalf of B_{i^*}, where A'_{i^*} is randomly chosen from corrupt parties and $\mathsf{IsPrivate}_r = 1$.
   - Upon receiving ("ppromise", A'_{i^*}, B_{i^*}) from $\mathcal{F}_{\text{ucc}}$, simulate the ideal functionality $\mathcal{F}_{s,j}$ for T.
   - Append $(B_{i^*}, \perp, \mathsf{IsPrivate}_r)$ to the list Puzzle.
4. T, B_{i^*} **Honest,** A_{i^*} **Corrupt.** Upon receiving ("ppromise", A_{i^*}, B_{i^*}, $\mathsf{IsPrivate}_r$) from $\mathcal{F}_{\text{ucc}}$:
   - Follow the protocol on behalf of T and B_{i^*} and simulate the ideal functionality $\mathcal{F}_{s,j}$.
   - Append $(B_{i^*}, \perp, \mathsf{IsPrivate}_r)$ to the list Puzzle.

(B) **Source-chain transfer**: Distinguish the following cases,

1. **Honest T, A_{i^*}, Corrupt B_{i^*}.** Upon receiving ("psolve", A_{i^*}, B_{i^*}, $\mathsf{IsPrivate}_s$) from $\mathcal{F}_{\text{ucc}}$:
   - Wait for ("promiseInfo", Z'_{i^*}) from B_{i^*} within 1 round. If there is no message from B_{i^*}, send ("noInfo") to $\mathcal{F}_{\text{ucc}}$ and abort.
   - Execute the protocol on behalf of A_{i^*}, T and simulate $\mathcal{F}_{s,j}$. If abort during execution, send ("failSolve") to $\mathcal{F}_{\text{ucc}}$.
   - If sct'_{i^*} is rerandomized from $\mathsf{sct}_{i'}$ which is extracted from ct and the corresponding receiver is $B_{i'}$, send ("changePuzzle", A_{i^*}, $B_{i'}$) to $\mathcal{F}_{\text{ucc}}$ and update $(B_{i'}, \perp, \cdot) \in \mathsf{Puzzle}$ to $(B_{i'}, \top, \cdot)$.
   - If sct'_{i^*} is not rerandomized from any ciphertext which is extracted from ct, send ("changePuzzle", A_{i^*}, $\emptyset$) to $\mathcal{F}_{\text{ucc}}$.
2. **Corrupt A_{i^*}, Honest T, Honest or Corrupt B_{i^*}.** Upon receiving $\mathcal{F}_{s,j}.\mathtt{OpenSh}(\mathsf{tlock}_{ps,j}, \mathsf{pk}_{A_{i^*}}, T, \mathsf{amt}, \mathsf{sk}_{A_{i^*}})$ from A_{i^*} and $(\mathsf{pk}_{A_{i^*}}, \mathsf{sk}_{A_{i^*}})$ is a valid key pair:
   - Send ("psolve", $B_{i'}$, $\mathsf{IsPrivate}_s$) on behalf of A_{i^*}, where $B_{i'}$ is randomly chosen from corrupt parties, and $\mathsf{IsPrivate}_s$=1.
   - Upon receiving ("psolve", A_{i^*}, $B_{i'}$, $\mathsf{IsPrivate}_s$) from $\mathcal{F}_{\text{ucc}}$, simulate the ideal functionality $\mathcal{F}_{s,j}$ for A_{i^*}.
   - Upon receiving ("solveReq", A_{i^*}) from $\mathcal{F}_{\text{ucc}}$, if receive ("solveReq", $\hat{\sigma}_{A_{i^*}}, Z'_{i^*}$) from A_{i^*} on behalf of T, go to the next step. Otherwise, send ("failSolve") to $\mathcal{F}_{\text{ucc}}$.
   - Execute the protocol on behalf of T and simulate the ideal functionality $\mathcal{F}_{s,j}$. If abort during the execution, send ("failSolve") to $\mathcal{F}_{\text{ucc}}$.
   - If sct'_{i^*} is rerandomized from $\mathsf{sct}_{i'}$ which is extracted from ct and the corresponding receiver is $B_{i'}$, send ("changePuzzle", A_{i^*}, $B_{i'}$) to $\mathcal{F}_{\text{ucc}}$ and update $(B_{i'}, \perp, \cdot) \in \mathsf{Puzzle}$ to $(B_{i'}, \top, \cdot)$.
   - If sct'_{i^*} is not rerandomized from any ciphertext which is extracted from ct, send ("changePuzzle", A_{i^*}, $\emptyset$) to $\mathcal{F}_{\text{ucc}}$.
3. **Honest A_{i^*}, Corrupt T.** Distinguish the following cases.
   Case B_{i^*} is honest. Upon receiving ("psolve", $A_{i^*}, \diamond/B_{i^*}$, $\mathsf{IsPrivate}_s$) from $\mathcal{F}_{\text{ucc}}$:
   - If receiving $\diamond$, retrieve the first $B_{i'} \in \mathsf{Hpair}$, remove $B_{i'}$ from Hpair, and derive $Z'_{i'}$ from the simulated $B_{i'}$. If receiving B_{i^*}, derive the corresponding Z'_{i^*} from the simulated B_{i^*}.
   - Execute the protocol on behalf of A_{i^*}.
   - Simulate the ideal functionality $\mathcal{F}_{s,j}$.
   - Update $(B_{i'}, \perp, \cdot) \in \mathsf{Puzzle}$ to $(B_{i'}, \top, \cdot)$.
   - After executing the $\Pi_{\text{AS},j}.\mathsf{Extract}$ algorithm, check if $\mathsf{dlp}'_{i^*} = f^{y'_{i^*}}$. If not, abort.

   Case B_{i^*} is corrupt. Upon receiving ("psolve", A_{i^*}, B_{i^*}, $\mathsf{IsPrivate}_s$) from $\mathcal{F}_{\text{ucc}}$:
   - Wait for ("promiseInfo", Z'_{i^*}) from B_{i^*} within 1 round. If there is no message from B_{i^*}, send ("noInfo") to $\mathcal{F}_{\text{ucc}}$ and abort.
   - Execute the protocol on behalf of A_{i^*}.
   - Simulate the ideal functionality $\mathcal{F}_{s,j}$.
   - Upon receiving ("solveReq", A_{i^*}) from $\mathcal{F}_{\text{ucc}}$, if sct'_{i^*} is rerandomized from $\mathsf{sct}_{i'}$ which is extracted from ct and the corresponding receiver is $B_{i'}$, send ("changePuzzle", A_{i^*}, $B_{i'}$) to $\mathcal{F}_{\text{ucc}}$ and update $(B_{i'}, \perp, \cdot) \in \mathsf{Puzzle}$ to $(B_{i'}, \top, \cdot)$.
   - If sct'_{i^*} is not rerandomized from any ciphertext which is extracted from ct, send ("changePuzzle", A_{i^*}, $\emptyset$) to $\mathcal{F}_{\text{ucc}}$.
   - After executing the $\Pi_{\text{AS},j}.\mathsf{Extract}$ algorithm, check if $\mathsf{dlp}'_{i^*} = f^{y'_{i^*}}$. If not, abort.
4. **Corrupt** A_{i^*}, T, **Honest** B_{i^*}. Upon receiving $\mathcal{F}_{s,j}.\mathtt{OpenSh}(\mathsf{tlock}_{ps,j}, \mathsf{pk}_{A_{i^*}}, T, \mathsf{amt}, \mathsf{sk}_{A_{i^*}})$ from A_{i^*} and $(\mathsf{pk}_{A_{i^*}}, \mathsf{sk}_{A_{i^*}})$ is a valid key pair:
   - Send ("psolve", B_{i^*}, $\mathsf{IsPrivate}_s$) on behalf of A_{i^*}, where B_{i^*} is the party that has sent ("promiseInfo", Z'_{i^*}) to A_{i^*} in the puzzle-promise phase and $\mathsf{IsPrivate}_s = 1$.
   - Simulate the ideal functionality $\mathcal{F}_{s,j}$ for A_{i^*} and T.
   - Update $(B_{i^*}, \perp, \cdot) \in \mathsf{Puzzle}$ to $(B_{i^*}, \top, \cdot)$.

(C) **Open**:
Distinguish the following cases.

1. B_{i^*} **Honest.** Upon receiving ("open", B_{i^*}) from $\mathcal{F}_{\text{ucc}}$:
- Wait for ("solved", $y^*_{i^*}$) from A_{i^*} within 1 round, where A_{i^*} is either the honest party that is assigned to be the corresponding sender of B_{i^*} or a corrupt party to whom B_{i^*} sends promise information. If there is no message from A_{i^*}, send ("failOpen") to $\mathcal{F}_{\text{ucc}}$ and abort.
- If there is no message from A_{i^*}, send ("failOpen") to $\mathcal{F}_{\text{ucc}}$ and abort.
- Execute the open phase on behalf of B_{i^*}.
- If abort during the open phase, send ("failOpen") to $\mathcal{F}_{\text{ucc}}$.
- Check if the signature adapted from the pre-signature is valid. If not, abort.
- Simulate the ideal functionality $\mathcal{F}_{s,j}$ for B_{i^*}.
- If $(B_{i^*}, \top, \cdot) \notin \mathsf{Puzzle}$, abort. Otherwise, remove $(B_{i^*}, \top, \cdot)$ from Puzzle.

2. B_{i^*} **Corrupt.** Upon receiving $\mathcal{F}_{s,j}.\mathtt{CloseSh}(\tilde{\mathsf{pk}}_{T,j}, \tilde{\mathsf{pk}}_{B_{i^*}}, \mathsf{pk}_{B_{i^*}}, \mathsf{amt}, \sigma_T, \sigma_{B_{i^*}})$:
- Send ("open", B_i) to $\mathcal{F}_{\text{ucc}}$ on behalf of B_{i^*}.
- Simulate the ideal functionality $\mathcal{F}_{s,j}$ for B_{i^*}.
- If $(B_{i^*}, \top, \cdot) \notin \mathsf{Puzzle}$, abort. Otherwise, remove $(B_{i^*}, \top, \cdot)$ from Puzzle.

Simulate the ideal functionality $\mathcal{F}_{s,j}$ for P_i

When the ideal functionality $\mathcal{F}_{s,j}$ is called by P_i, the simulator $\mathcal{S}$ distinguishes the following cases,

1. Case P_i is honest. when $\mathcal{F}_{s,j}.\mathtt{OpenSh}$ or $\mathcal{F}_{s,j}.\mathtt{CloseSh}$ is called, the simulator proceeds as $\mathcal{F}_{s,j}$ except that $\mathcal{S}$ does not directly call $\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j)$ but derives the output from $\mathcal{F}_{\text{ucc}}$ within Δ_j rounds.

2. Case P_i is corrupt.
- When $\mathcal{F}_{s,j}.\mathtt{OpenSh}$ is called, the simulator proceeds as $\mathcal{F}_{s,j}$ except that $\mathcal{S}$ does not directly call $\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j)$ but sends ("openSh") to $\mathcal{F}_{\text{ucc}}$ and derives the output from $\mathcal{F}_{\text{ucc}}$ within Δ_j rounds.
- When $\mathcal{F}_{s,j}.\mathtt{CloseSh}$ is called, the simulator proceeds as $\mathcal{F}_{s,j}$ except that $\mathcal{S}$ does not directly call $\mathcal{L}_j^{\mathsf{SIG}}(\Delta_j)$ but sends ("closeSh") to $\mathcal{F}_{\text{ucc}}$ and derives the output from $\mathcal{F}_{\text{ucc}}$ within Δ_j rounds.

Fig. 12: The simulation procedure.

APPENDIX D
PARAMETERS SETS

The parameters in UniCross are set based on a Nonlinear Goal Programming (NGP) problem. The objective function is the communication size of the tumbler amortized across each user, which should be as short as possible. Constraints are defined to guarantee the security of UniCross. A set of optimal parameters is given below to solve such an NGP problem.

For the RLWE-based encryption scheme, we set $\mathcal{B} = 1$, $q = 2^{16}$ and $d = 1024$. Then, we set $t = 8$ to ensure the correct decryption. We follow the methodology in Section 3.2.4 of [44] and measure the practical hardness of decision-RLWE in terms of the "root Hermite factor" δ. Our parameter setting yields $\delta < 1.0045$ for decision-RLWE, and $\delta \approx 1.0045$ has been used in recent works [45], [46] for 128-bit post-quantum security.

For the aggregation of puzzles, we set $\mathsf{chk} = 64$ and $n = 16$, i.e., an aggregated puzzle $Z := (\mathsf{ct}, \{\mathsf{dlp}_i\}_{i\in[n]})$ can package 16 sub-puzzles $Z_i := (\mathsf{sct}_i, \mathsf{dlp}_i)$ in which sct_i consists of 64 LWE-based ciphertexts. In such a setting, the relation $\mathsf{R}_{\text{RLWE-DL}}$ proved by HybridProof in UniCross is a hard relation. Otherwise, there would exist an adversary breaking the CPA security of the RLWE-based encryption scheme or solving the discrete logarithm of any $\mathsf{dlp}_i = f^{y_i}$, where y_i is randomly chosen from some 2^{128}-sized subset of $\mathbb{Z}_p$.

Aggregated puzzles for arbitrary anonymity size. To extend the anonymity size N, the tumbler can generate multiple (i.e., $\lceil \frac{N}{16} \rceil$) aggregated puzzles and prove that each of them is well-formed without changing the optimal parameters. For example, suppose $N = 17$. In this case, the tumbler generates 2 aggregated puzzles in the puzzle-promise phase to ensure that 17 sub-puzzles can be extracted and solved in the puzzle-solving phase. Subsequently, each receiver only needs to take the aggregated puzzle containing the sub-puzzle for him/her.

APPENDIX E
ADDITIONAL PRELIMINARIES

A. Assumptions

Definition 3 (Decision-RLWE)**.** *The Decision-RLWE$_{d,q,\mathcal{B}}$ problem is to distinguish between the following two cases: i)* $(\mathbf{a}, \mathbf{b}) \leftarrow\$ \mathcal{R}_q^2$*, and ii)* $(\mathbf{a}, \mathbf{a}\cdot\mathbf{s} + \mathbf{e})$ *for* $\mathbf{a} \leftarrow\$ \mathcal{R}_q$*, a secret* $\mathbf{s} \leftarrow\$ \mathbb{S}_{\mathcal{B}}$ *and an error* $\mathbf{e} \leftarrow\$ \mathbb{S}_{\mathcal{B}}$*.*

Decision-RLWE$_{d,q,\mathcal{B}}$ hardness assumption implies that $\mathbf{a}\cdot\mathbf{s} + \mathbf{e}$ is (computationally) indistinguishable from a uniformly random element of $\mathcal{R}_q$ when $\mathbf{s}, \mathbf{e} \leftarrow\$ \mathbb{S}_{\mathcal{B}}$ for any $\mathcal{B} \geq 1$.

Definition 4 (One-More Discrete Logarithm (OMDL) Assumption)**.** *Let $\mathbb{G}$ be a cyclic group of prime order p and f a random generator of $\mathbb{G}$. The one-more discrete logarithm (OMDL) assumption states that for all $\lambda \in \mathbb{N}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for any PPT adversary $\mathcal{A}$ making at most* $\mathsf{pl} = \mathsf{poly}(\lambda)$ *queries to* $\mathsf{DL}(\cdot)$*, it holds that*

$$\Pr\left[\forall i, x_i = r_i \;\middle|\; \begin{array}{c} r_0, \cdots, r_{\mathsf{pl}} \leftarrow\$ \mathbb{Z}_p \\ \forall i \in [\mathsf{pl}+1], h_i = f^{r_i} \\ (x_0, \cdots, x_{\mathsf{pl}}) \leftarrow \mathcal{A}^{\mathsf{DL}(\cdot)}(h_0, \cdots, h_{\mathsf{pl}}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the $\mathsf{DL}(\cdot)$ *oracle takes an element $h \in \mathbb{G}$ as input and returns x such that $h = f^x$.*

B. Adaptor Signatures

Definition 5 (Secure Adaptor Signature)**.** *An adaptor signature Π_{AS} is secure if it satisfies the following definitions.*

Definition 6 (Pre-signature Correctness)**.** *An adaptor signature scheme Π_{AS} satisfies pre-signature correctness if for every message $m \in \{0,1\}^*$, and every statement/witness pair $(Y, y) \in \mathsf{R}$, the following holds:*

$$\Pr\left[\begin{array}{c} \mathsf{PreVrf}(\mathsf{pk}, m, Y, \hat{\sigma}) = 1 \wedge \\ \mathsf{Vf}(\mathsf{pk}, m, \sigma) = 1 \wedge \\ (Y, y') \in \mathsf{RS} \end{array} \;\middle|\; \begin{array}{c} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{SIG.Gen}(1^\lambda); \\ \hat{\sigma} \leftarrow \mathsf{PreSig}(\mathsf{sk}, m, Y); \\ \sigma = \mathsf{Adapt}(\hat{\sigma}, y); \\ y' = \mathsf{Extract}(\sigma, \hat{\sigma}, Y) \end{array}\right] = 1.$$

Definition 7 (Full Extractability)**.** *An adaptor signature Π_{AS} satisfies full extractability if for every PPT adversary $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr[\mathit{FullExt}^{\mathcal{A}}_{\Pi_{AS}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

where $\mathit{FullExt}^{\mathcal{A}}_{\Pi_{AS}}$ is defined in Figure 13.

Definition 8 (Pre-signature Adaptability)**.** *An adaptor signature scheme Π_{AS} satisfies pre-signature adaptability if for every message $m \in \{0,1\}^*$, and every statement/witness pair $(Y, y) \in \mathsf{R}$, any key pair* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ *and any pre-signature $\hat{\sigma} \leftarrow \{0,1\}^*$ with* $\mathsf{PreVrf}(\mathsf{pk}, m, Y, \hat{\sigma}) = 1$*, the following holds:*

$$\Pr[\mathsf{Vf}(\mathsf{pk}, m, \mathsf{Adapt}(\hat{\sigma}, y)) = 1] = 1.$$

$\mathrm{FullExt}^{\mathcal{A}}_{\Pi_E}(1^\lambda)$

$\mathcal{Q}_{\mathrm{fext}} = \emptyset, \mathcal{Q}_{\mathrm{st}} = \emptyset, \mathcal{Q}_{\mathrm{pS}} = []$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SIG.Gen}(1^\lambda)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{sign}}(\cdot), \mathcal{O}_{\mathrm{psign}}(\cdot), \mathcal{O}_{\mathrm{newY}}(\cdot)}(\mathsf{pk})$
assert $m^* \notin \mathcal{Q}_{\mathrm{fext}}$
$b_0 := \mathsf{Vf}(\mathsf{pk}, m^*, \sigma^*)$
$b_1 := \forall (Y, \hat{\sigma}) \in \mathcal{Q}_{\mathrm{pS}}[m^*]$ s.t. $Y \notin \mathcal{Q}_{\mathrm{st}}$: $(Y, \mathsf{Extract}(\sigma^*, \hat{\sigma}, Y)) \notin \mathsf{R}$
return $b_0 \wedge b_1$

$\mathcal{O}_{\mathrm{sign}}(m)$

$\sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, m)$
$\mathcal{Q}_{\mathrm{fext}} = \mathcal{Q}_{\mathrm{fext}} \cup \{m\}$
return σ

$\mathcal{O}_{\mathrm{psign}}(m, Y)$

$\hat{\sigma} \leftarrow \mathsf{PreSig}(\mathsf{sk}, m, Y)$
$\mathcal{Q}_{\mathrm{pS}}[m] = \mathcal{Q}_{\mathrm{pS}}[m] \cup \{Y, \hat{\sigma}\}$
return $\hat{\sigma}$

$\mathcal{O}_{\mathrm{newY}}()$

$(Y, y) \leftarrow \mathsf{creatR}(1^\lambda)$
$\mathcal{Q}_{\mathrm{st}} = \mathcal{Q}_{\mathrm{st}} \cup \{Y\}$
return Y

Fig. 13: Full extractability experiment of adaptor signatures.

Definition 9 (Computational Pre-verify Soundness)**.** *An adaptor signature scheme Π_{AS} satisfies computational pre-verify soundness if for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for polynomially-bounded $Y \notin L_{\mathsf{R}}$,*

$$\Pr\left[\begin{array}{c}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SIG.Gen}(1^\lambda), (m, \hat{\sigma}) \leftarrow \mathcal{A}(\mathsf{sk}) : \\ \mathsf{PreVrf}(\mathsf{pk}, m, \hat{\sigma}, Y) = 1\end{array}\right] \leq \mathsf{negl}(\lambda).$$

C. Linear-only Encryption Oracles

A public-key encryption scheme $\Pi_E := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ allows one to generate a key pair $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Anyone can encrypt message $m \in \mathbb{M}$ as $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ek}, m)$. The owner of the decryption key dk can decrypt the ciphertext $\mathsf{ct} \in \mathbb{C}$ as $m \leftarrow \mathsf{Dec}(\mathsf{dk}, \mathsf{ct})$. We require that Π_E satisfies perfect correctness and the standard notion of CPA-security. We say that an encryption scheme is *linearly homomorphic* if there exists some efficiently computable operation $+_C$ such that $\mathsf{Enc}(\mathsf{ek}, m_0) +_C \mathsf{Enc}(\mathsf{ek}, m_1) \in \mathsf{Enc}(\mathsf{ek}, m_0 +_M m_1)$, where $+_C$ (or $+_M$) is defined over the ciphertext (or message) space. The α-fold application of $+_C$ is denoted by $\mathsf{Enc}(\mathsf{ek}, m)^\alpha$.

Linear-only encryption (LOE) is an idealized model introduced by [43] as a "generic homomorphic cryptosystem" and used by [19], [42]. In the LOE model, homomorphic encryption is modeled by giving access to oracles instead of their corresponding algorithms. Figure 14 gives a formal description of these oracles.

$\mathcal{O}_{\mathsf{Gen}}(i)$

$\mathsf{ek}_i \leftarrow\!\!\$\ \{0,1\}^n$
Enter (i, ek_i) into table K
return ek_i

$\mathcal{O}_{\mathsf{Enc}}(\mathsf{ek}_i, m)$

$\mathsf{ct}_j \leftarrow\!\!\$\ \{0,1\}^n$
Enter (m, ct_j) into table M_i
return ct_j

$\mathcal{O}_{\mathsf{Dec}}(\mathsf{ek}_i, \mathsf{ct})$

if $(\cdot, \mathsf{ct}) \notin M_i$ **then return** $\perp$
else
  Look up m corresponding to ct_j in table M_i
  return m

$\mathcal{O}_{\mathsf{Add}}(\mathsf{ek}_i, \mathsf{ct}_0, \mathsf{ct}_1)$

Look up m_0, m_1 corresponding to $\mathsf{ct}_0, \mathsf{ct}_1$ in table M_i
$\mathsf{ct}' \leftarrow\!\!\$\ \{0,1\}^n$
Enter $(m_0 + m_1, \mathsf{ct}')$ into table M_i
return ct'

Fig. 14: Linear-only encryption oracles.

D. Non-interactive Zero-knowledge Proof

Definition 10 (NIZK)**.** *The triple $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$ is a NIZK for relation R if it satisfies the following three definitions.*

Definition 11 (Perfect Completeness)**.** *A non-interactive zero-knowledge argument protocol NIZK is perfectly complete if the prover who knows a witness of a statement could always convince the verifier to accept. Formally,*

$$\Pr\left[\mathsf{Vf}(\sigma, x, \pi) = 1 \vee (x, w) \notin \mathsf{R}_L \;\middle|\; \begin{array}{c}(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda); \\ \pi \leftarrow \mathsf{Prove}(\sigma, x, w)\end{array}\right] = 1.$$

Definition 12 (Knowledge Soundness)**.** *For a non-interactive zero-knowledge argument protocol NIZK, knowledge soundness holds if the prover "knows" a witness for the given accepted instance. Namely, for all non-uniform polynomial time prover $\mathcal{P}^*$, there exists a PPT extractor Ext such that,*

$$\Pr\left[\begin{array}{c}(x, w) \notin \mathsf{R}_L \wedge \\ \mathsf{Vf}(\sigma, x, \pi) = 1\end{array}\;\middle|\; \begin{array}{c}(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda); \\ (x, \pi; w) \leftarrow \mathsf{Ext}^{\mathcal{P}^*}(\sigma, x);\end{array}\right] \leq \mathsf{negl}(\lambda).$$

where $\mathsf{Ext}^{\mathcal{P}^*}$ means the extractor gets full access to the prover's state, including any random coins.

Definition 13 (Zero-knowledge)**.** *A non-interactive zero-knowledge argument protocol NIZK is zero-knowledge if the verifier cannot get additional information except that the statement is true. Formally, for $x \in L$, there exists a PPT algorithm Sim such that for all adversaries $\mathcal{A}$,*

$$\Pr\left[\mathcal{A}(\sigma, \tau, x, \pi) = 1 \;\middle|\; \begin{array}{c}(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda); \\ \pi \leftarrow \mathsf{Prove}(\sigma, x, w)\end{array}\right] \approx$$
$$\Pr\left[\mathcal{A}(\sigma, \tau, x, \pi) = 1 \;\middle|\; \begin{array}{c}(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda); \\ \pi \leftarrow \mathsf{Sim}(\tau, x)\end{array}\right].$$

A zero-knowledge argument of knowledge for $(x, w) \in \mathsf{R}_L$ consists of three interactive algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$ running in probabilistic polynomial time. On input the security parameter 1^λ, Setup outputs a common reference string σ. The transcripts generated by the interaction between Prover and Verifier are denoted by $tr \leftarrow \langle \mathsf{Prove}(\sigma, x, w), \mathsf{Vf}(\sigma, x) \rangle$. We write the output of interactive protocol as $b = \langle \mathsf{Prove}(\sigma, x, w), \mathsf{Vf}(\sigma, x) \rangle$. If the verifier accepts $b = 1$, otherwise $b = 0$.

We require that the zero-knowledge arguments of knowledge for R_L have perfect completeness, knowledge soundness, and perfect special honest verifier zero-knowledge (SHVZK). We refer readers to [47] for the detailed definitions of these three properties.