Cybersecurity

RESEARCH **Open Access**

Atomic cross-chain swap based on private key exchange

Zeshuo Zhu[1,2], Rui Zhang[1,2] and Yang Tao[1*]

Abstract

Atomic Cross-Chain Swap (ACCS) is one important topic in cryptocurrency, where users can securely and trustlessly exchange assets between two different blockchains. However, most known ACCS schemes assume specific scripting functionalities of the underlying blockchains, such as Hash Time Locked Contracts (HTLC). In addition, these schemes are typically only applicable to certain digital signature schemes, like Schnorr or Elliptic Curve Digital Signature Algorithm (ECDSA) signatures. In this paper, we propose a generic ACCS scheme, independent from the underlying blockchains. To the best of our knowledge, this is the first solution of this kind. Our results are as follows. First, we define a formal system model of ACCS. Next, we present a generic ACCS scheme meets our model. This scheme admits atomicity in cross-chain swaps without the need for a Trusted Third Party (TTP) and protects users' privacy. Finally, by using the Non-Interactive Zero-Knowledge (NIZK) proof protocol as a tool, we instantiate our generic scheme for Elliptic Curve Discrete Logarithm Problem-based (ECDLP-based) signatures. In addition, we implement our scheme, and the experimental results show that our protocol outperforms the existing ACCS schemes, such as the HTLC-based schemes.

Keywords Fair exchanges, Atomic swaps, Non-interactive zero-knowledge proofs

Introduction

Atomic cross-chain swap (Herlihy 2018) is used to directly exchange assets between two different blockchains without additional trust assumptions, such as trusted hardware or TTP. In ACCS, user A holds asset x on blockchain $\mathbb{B}_1$, while user B holds asset y on blockchain $\mathbb{B}_2$, and both intend to exchange their assets. The main goal of ACCS is to ensure that if the swap is successful, user A receives asset y and user B receives asset x. Otherwise, all assets are refunded to both users.

In recent years, with the maturity of blockchain technology, ACCS has gained much attention. The main reason is that in the early stages of blockchain development, the necessity of interoperability between different blockchains was not considered. Therefore, each blockchain operated in isolation, lacking efficient communication mechanisms, which led to what is commonly known as the "blockchain islands" phenomenon. This makes it difficult to exchange data and assets between different blockchains, thereby hindering the further development of blockchain technology. To address this limitation, ACCS emerged as a solution for achieving interoperability between various blockchains. Its primary goal is to enable the transfer and exchange of data and value across different chains. By implementing ACCS, the "blockchain islands" phenomenon has been effectively broken, promoting the continued progress of blockchain technology.

However, with the application of blockchain technology in various industries, such as financial services (Guo and Liang 2016), healthcare (Agbo et al. 2019), and Internet of Things (IoT) (Reyna et al. 2018), the mere circulation of data and value is no longer sufficient to meet the

*Correspondence:
Yang Tao
taoyang@iie.ac.cn
[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, No.19 Shucun Road, Beijing 100084, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

practical requirements for cross-chain interactions. Real-world application scenarios present additional requirements for ACCS. For example, different enterprises often choose different blockchain platforms based on their industry characteristics. Therefore, they may require that the ACCS scheme can be used on any blockchain platform. Moreover, the cross-chain interaction process often involves sensitive private data. Thus, users or companies typically want to prevent the leakage of sensitive information in ACCS. Finally, for some enterprises with high security requirements, they usually hope to implement ACCS without the need for TTP. These emerging requirements make it meaningful to conduct further research on ACCS.

Related work

Up to now, there were two popular ways to achieve trustless cross-chain swaps. The first approach uses on-chain scripts to simulate a TTP. It admits atomicity in cross-chain swaps through the automatic execution of on-chain scripts, such as smart contracts. A seminal early work in this field is the HTLC proposed by TierNolan (2013). Subsequently, in 2016, Poon used HTLC to execute atomic swaps between payment channels in the Lightning Network (Poon and Dryja 2016). Later, in 2018, Herlihy formalized the theory of such ACCS (Herlihy 2018). However, ACCS schemes that rely on on-chain scripts may not be compatible with all blockchain platforms, as they depend on specific functionalities provided by each blockchain script. For example, a HTLC-based ACCS scheme requires that both blockchain scripts must support the same hash function. In addition, due to the fact that on-chain scripts are public, using them may compromise users' privacy (Deshpande and Herlihy 2020; Thyagarajan et al. 2022).

In contrast, the second approach uses cryptographic techniques to simulate a TTP. It is more compatible than the first one and can protect users' privacy. However, it typically relies on more complex cryptographic primitives, such as adaptor signatures (Fournier 2019) or lockable signatures (Thyagarajan and Malavolta 2021). In recent years, the ACCS schemes based on cryptographic techniques have been thoroughly studied, and many practical schemes have been proposed. For example, in 2020, Shlomovits designed a scriptless ACCS scheme (Shlomovits and Leiba 2020), where both players gradually released their private key shares. His scheme protected the users' privacy, but was limited to blockchains utilizing ECDLP-based signatures. In addition, it required a TTP called Provider to enhance system security. Subsequently, in 2021, Thyagarajan proposed lockable signatures (Thyagarajan and Malavolta 2021), and in 2022, he further presented a generic scriptless ACCS based on lockable signatures (Thyagarajan 2022). His protocol enables fair exchange of coins among any currencies, while only requiring the minimal script from the underlying blockchain to verify payments, i.e. the verification of digital signatures. Unfortunately, his protocol does not provide an efficient solution for blockchains that do not support adaptor signatures (Erwig et al. 2021; Hanzlik et al. 2022).

Tumblebit (Heilman et al. 2017), A^2L (Tairi et al. 2021), and BlindHub (Qin et al. 2023) are atomic swap protocols that use an alternate route. In these protocols, a participant exchanges his coins through an untrusted intermediate party. When multiple swaps occur simultaneously, these protocols guarantee that neither intermediary nor any other participant can link the specific coins being exchanged, thereby protecting user privacy. However, Tumblebit relies on the HTLC scripts of the underlying blockchains, which results in poor compatibility and privacy. A^2L improved this issue, but a later work (Glaeser et al. 2022) found that A^2L's security model had a gap that allowed key recovery attacks on specific instantiations. Subsequently, Glaeser improved A^2L and proposed A^2L^+ and A^2L^{UC} (Glaeser et al. 2022). However, neither of them is compatible with systems that lack adaptor signature support. In addition, since BlindHub also relies on adaptor signatures, it faces a similar compatibility issue (Hanzlik et al. 2022).

Challenges of the the previous work

To summarize, the known ACCS solutions have three limitations thus are desirable to overcome the following challenges:

- *Generality.* It means that the ACCS scheme is independent of the underlying blockchain platforms. The main reason for the formation of blockchain islands is that the underlying architecture and data structure of each blockchain are different. For example, Bitcoin uses the Unspent Transaction Output (UTXO) model (Nakamoto 2008), while Ethereum uses the account-based transaction model (Buterin 2013). The heterogeneity between blockchains leads to the complexity and incompatibility of cross-chain interactions.
- *Privacy.* In ACCS, users' privacy includes: (i) On-chain data privacy (Deshpande and Herlihy 2020). The process of cross-chain swap may leak users' on-chain data. For instance, the utilization of public smart contracts could disclose sensitive information such as transaction amounts or participant identities. (ii) Transaction privacy, which includes transaction unlinkability and transaction indistinguishability. The former means that adversaries who do not par-

ticipate in ACCS cannot associate two cross-chain transactions. For example, HTLC-based ACCS cannot achieve transaction unlinkability, because two cross-chain transactions use the same hash value, which makes it easy for adversaries to infer that they may come from the same user. The latter means that observers who do not participate in ACCS cannot distinguish the cross-chain transactions from standard transactions (Thyagarajan et al. 2022).

– *Atomicity without TTP.* It means that the scheme admits atomicity in cross-chain swaps without relying on a TTP. ACCS can be easily implemented using a TTP. However, introducing TTP contradicts the decentralized nature of blockchain. In addition, TTP may bring additional security threats, such as single point of failure or trust problems. Unfortunately, Zamyatin et al. (2021) pointed out that in the deterministic system model of distributed ledgers, achieving Correct Cross-Chain Communication (CCC) is impossible in asynchronous settings without a TTP. Therefore, one of the technical challenges in this field is to design an ACCS scheme in which the distributed ledger platform, on-chain scripts or cryptographic primitives within the system serve as an implicit TTP, thereby guaranteeing atomicity in cross-chain swaps.

Hence, we raise the following question: *"Is it possible to design an ACCS scheme without the above three limitations, namely, to meet generality, privacy and atomicity without TTP?"*

Our contribution

In this paper, we focus on designing a generic ACCS scheme without TTP, and give an affirmative answer to the above question. Our contributions are summarized as follows:

Model for ACCS. We give a formal system model of ACCS and the corresponding security definitions. Our model is applicable to one-to-one ACCS scenarios on any blockchain. To the best of our knowledge, this is the first work in this field.

Generic Atomic Cross-Chain Swaps Scheme. We present a generic ACCS scheme that solves all above-mentioned challenges, namely, generality, privacy and atomicity without TTP. In addition, this scheme has been proven to be secure in our model. We outline the comparison between our scheme and other existing schemes in Table 1. Notably, our scheme is not only applicable to existing blockchain platforms, but can also be adapted to potential future platforms.

New Tool: NIZK for the Correct Commitment of Discrete Logarithm (DL). To show the practicability of our generic scheme, we instantiate it in ECDLP-based blockchains. To achieve this instantiation, we present a NIZK protocol Π_{CCNIZK} for the correct commitment of DL, which is an effective tool to instantiate our scheme. It is used to verify the correctness of the secret segments released by each player, thereby achieving fair exchange of private key shares. We regard this zero-knowledge proof as an independent property, which may bring other interesting applications.

Table 1 Comparison with existing schemes[1]

Schemes	Security	Generality	Privacy	Atomicity without TTP
Li et al. (2022)	✓	×	✓	×
Mazumdar (2022)	✓	×	×	✓
Bentov et al. (2019)	✓	×	✓	×
Hei et al. (2022)	✓	✓	×	×
Hoenisch et al. (2022)	✓	×	✓	✓
Shlomovits and Leiba (2020)	✓	×	✓	✓
Wang and Nixon (2021)	✓	✓	×	×
Thyagarajan et al. (2022)	✓	×	✓	✓
Heilman et al. (2017)	✓	×	×	✓
Tairi et al. (2021)	✓	×	✓	✓
Glaeser et al. (2022) (A^2L^+)	✓	×	✓	✓
Glaeser et al. (2022) (A^2L^{UC})	✓	×	✓	✓
Qin et al. (2023)	✓	×	✓	✓
Herlihy et al. (2019)	✓	×	×	✓
Manevich and Akavia (2022)	✓	×	×	✓
Gugger (2020)	×	×	✓	✓
Hoenisch and Pino (2021)	×	×	✓	✓
Deshpande and Herlihy (2020)	×	×	✓	✓
Ours	✓	✓	✓	✓

The "Security" indicator is used to evaluate whether the scheme provides a security model, security analysis, or security proof. The specific evaluation contents of "Generality", "Privacy" and "Atomicity without TTP" indicators are shown in this section. In addition, unless the scheme indicates that the introduced third party does not need to be trusted, we will treat it as a TTP. Including but not limited to third-party blockchain, trusted execution environment, trusted execution hardware, etc

Technical overview

We briefly review our technical treatments below.

Model for ACCS. In our system model, there are two entities, player P_1 and player P_2, and we focus on a one-to-one ACCS between these two entities. To ensure the generality of this model, we represent each blockchain as a distributed ledger, independent of the specific blockchain implementation. To better describe the ACCS in the real world, we divide it into five phases: setup, freeze, exchange, complete and timeout. In terms of security, we consider possible security threats from a malicious adversary, and formally define the security of ACCS. Therefore, once our scheme is proven in this model, it means that the scheme is sufficiently secure.

Generic Construction of Atomic Cross-Chain Swaps. The technical difficulty in building our generic ACCS scheme lies in solving the three above-mentioned challenges, namely generality, privacy and atomicity without TTP.

To solve the first and second challenges, we move private key exchange off-chain. Our scheme uses cryptographic techniques instead of on-chain scripts, making it independent of the specific blockchain architecture. Specifically, we use a Fair Exchange (FE) protocol to achieve off-chain private key exchange, and utilize Verifiable Timed Signatures (VTS) (Thyagarajan et al. 2020) for asset refund. As a result, the cross-chain transactions generated by our scheme are indistinguishable from the standard one-to-one transactions.

Then, to solve the third challenge, we use the idea of gradual release to achieve fair exchange of private keys. Zamyatin et al. (2021) pointed out the similarity between the ACCS problem and the FE problem (Asokan 1998). Specifically, their properties of effectiveness (Asokan 1998) and timeliness (Zamyatin et al. 2021) are similar, and the fairness (Asokan 1998) in FE is similar to the atomicity (Zamyatin et al. 2021) in ACCS. Therefore, we can regard the atomicity property in ACCS as the fairness property in private key fair exchange. However, unlike the fair exchange of items, in ACCS, the sender does not lose the knowledge of the private key after the swap. Due to the fact that private key represents ownership of asset in blockchain, this will result in both players eventually owning the asset. To solve this problem, we let the players exchange the private key shares rather than the complete private keys. Specifically, we introduce a joint address (Thyagarajan et al. 2022) that is jointly controlled by both players, and each player holds a private key share. Then, they exchange their private key shares fairly to gain ownership of the joint address.

Unfortunately, it has been shown by Cleve (1986) that without a majority of honest parties, it is generally impossible to achieve complete fairness. However, achieving partial fairness (Gordon and Katz 2012) is possible, and gradual release is one of the widely used ideas. It allows both parties to exchange their secrets little by little, where one party can obtain an advantage over the other party, but this advantage is polynomially bounded. Our generic scheme is based on this idea. In our scheme, both players first divide the private key shares into multiple segments, then exchange the commitments of these segments and their corresponding proofs, and finally alternately release each segment. In this way, a player can only have at most one segment advantage over the other player.

Next, we give a high-level overview of our generic scheme. First, both players transfer their assets to joint addresses. To prevent indefinite locking of assets, we introduce a timeout mechanism for refund. Next, using the FE protocol with gradual release, players exchange their private key shares. Finally, they can reconstruct the complete private keys of the joint addresses to transfer assets. In addition, this scheme is proven to be secure in our system model.

New Tool: NIZK for the Correct Commitment of DL. We borrow the idea from Camacho (Camacho 2013) to build a new zero-knowledge proof protocol. Camacho (2013) proposed a short signature exchange without TTP by gradually releasing the blinding factor of the signature. However, his scheme can only achieve bit-by-bit private key exchange. We replace the bit commitment proof in the original protocol with Bulletproofs range proof, enabling it to support segment-by-segment private key share exchange.

Preliminaries

In this section, we review some useful notations and notions.

Notations. For $m, n \in \mathbb{N}$, where $m < n$, $[n]$ represents the set of integers $\{1, ..., n\}$, and $[m..n]$ represents the set of integers $\{m, m+1, ..., n-1, n\}$. Let $\lambda \in \mathbb{N}$ be a security parameter, 1^{λ} denotes a unary string with λ ones, and p represents a prime number with λ bits. We say that the function $\mathsf{negl} : \mathbb{N} \rightarrow [0, 1]$ is negligible in λ, if for every polynomial $q(\cdot)$ there exists λ_0 such that $\forall \lambda > \lambda_0 : \mathsf{negl}(\lambda) < 1/q(\lambda)$. We use $x \xleftarrow{R} X$ to represent an element x randomly and uniformly chosen from the set X, and use $x \leftarrow v$ to represent the variable x assigned the value v. We represent a protocol as $(\mathsf{output}_1, \mathsf{output}_2) \leftarrow \mathsf{Protocol}\langle P_1(\mathsf{input}_1), P_2(\mathsf{input}_2)\rangle$, where P_1 inputs input_1 and obtains output output_1, and P_2 inputs input_2 and obtains output output_2.

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two groups with the same prime order p, i.e., $p = |\mathbb{G}| = |\mathbb{G}_T|$, where $\mathbb{G}$ is an additive group and $\mathbb{G}_T$ is a multiplication group. Let G represents a random generator in $\mathbb{G}$, and $\mathbb{Z}_p$ represents a ring of integer module p. The notation $\mathbb{Z}_p \backslash \{0\}$ is defined as $\mathbb{Z}_p^*$.

The vector $\vec{v}$ is defined as $\vec{v} = (v_i)_{i \in (n]}$, where each component v_i represents a value associated with the corresponding index i. If the vector contains elements in $\mathbb{Z}_p^n$, it can also be written as $B(\cdot) = (B[1], B[2], ..., B[n])$. We define l and κ as symbols that satisfy $\kappa = \lceil \frac{\lambda}{l} \rceil$. Let $\theta \in \mathbb{Z}_p^*$, and the κ-Segmentation of θ refers to dividing θ into κ segments. Each segment has a length of l bits, and if the length is less than l, leading zeros are added. We denote the vector of κ-Segmentation of θ as $\theta[\cdot] = (\theta[1], ..., \theta[\kappa])$, where θ can be expressed as $\theta = \sum_{i\in[\kappa]} \theta[i]2^{(i-1)l}$. Additionally, let $P(\cdot)$ represent a formal polynomial with coefficients in $\mathbb{Z}_p$, and $P[\cdot]$ represent the vector of its coefficients. If the degree of polynomial $P(\cdot)$ is denoted as $d = deg(P)$, then $P(X) = \sum_{i\in[d+1]} P[i]X^{i-1}$.

Number theoretic assumptions

Bilinear Maps. Consider the $\mathbb{G}$, $\mathbb{G}_\mathsf{T}$, p and G defined above, assuming that ECDLP in $\mathbb{G}$ and $\mathbb{G}_\mathsf{T}$ is difficult. We define $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_\mathsf{T}$ as a bilinear map, which satisfies the following properties:

- Bilinearity: For $\forall P, Q \in \mathbb{G}, a, b \in \mathbb{Z}_p^*$, there is $e(aP, bQ) = e(P, Q)^{ab}$.
- Non-degenerate: Let P be the generator of $\mathbb{G}$. Then, $e(P, P)$ is the generator of $\mathbb{G}_\mathsf{T}$, which means $e(P, P) \neq 1$.
- Efficiently computable: Let G be the generator of $\mathbb{G}$, $\hat{\mathbb{G}}$ and $\hat{\mathbb{G}}_\mathsf{T}$ be groups of size p (p is a prime number of λ bits), with $\hat{e}$ being an efficient algorithm to compute the map. There exists an algorithm BMGen that takes as input 1^λ, and outputs $(p, \hat{\mathbb{G}}, \hat{\mathbb{G}}_\mathsf{T}, \hat{e}, G)$. For simplicity, we will not distinguish between $\hat{\mathbb{G}}, \hat{\mathbb{G}}_\mathsf{T}, \hat{e}$ and $\mathbb{G}, \mathbb{G}_\mathsf{T}, e$.

Let $N \in \mathbb{N}$. The common public parameters of the following assumptions are denoted by $\mathsf{pp} = \langle (p, \mathbb{G}, \mathbb{G}_\mathsf{T}, e, G), (G_0, G_1, G_2, ..., G_N)\rangle$. For $i \in [0..N]$, we have $G_i = s^iG$, where s is randomly chosen from $\mathbb{Z}_p^*$.

Definition 1 (*N-Diffie-Hellman Inversion (N-DHI)* Assumption (Mitsunari et al. 2002)). The N-DHI problem is to compute $\frac{1}{s}G$ given pp. For any Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$, we say that the N-DHI assumption holds, if we have:

$$\mathrm{Adv}^{\mathrm{N-DHI}}(\mathcal{A}, \kappa, \mathrm{N}) = \Pr[\frac{1}{\mathrm{s}}\mathrm{G} \leftarrow \mathcal{A}(1^\kappa, \mathsf{pp})] = \mathsf{negl}(\kappa)$$

Definition 2 (*N-Bilinear Diffie-Hellman Inversion (N-BDHI) Assumption*). The N-BDHI problem is to compute $e(G, G)^{\frac{1}{s}}$ given pp. For any PPT adversary $\mathcal{A}$, we say that the N-BDHI assumption holds, if we have:

$$\mathrm{Adv}^{\mathrm{N-BDHI}}(\mathcal{A}, \kappa, N) = \Pr[e(G, G)^{\frac{1}{s}} \leftarrow \mathcal{A}(1^\kappa, \mathsf{pp})] = \mathsf{negl}(\kappa)$$

Definition 3 (*N-Strong Diffie-Hellman (N-SDH)* Assumption (Boneh and Boyen 2008)). The N-SDH problem is to compute $(c, \frac{1}{s+c}G)$ given pp. For any PPT adversary $\mathcal{A}$, we say that the N-SDH assumption holds, if we have:

$$\mathrm{Adv}^{\mathrm{N-SDH}}(\mathcal{A}, \kappa, \mathrm{N}) = \Pr[(\mathrm{c}, \frac{1}{\mathrm{s}+\mathrm{c}}\mathrm{G}) \leftarrow \mathcal{A}(1^\kappa, \mathsf{pp})] = \mathsf{negl}(\kappa)$$

Digital signatures schemes

Digital Signatures Scheme. A digital signature scheme Π_{DS} consists of three algorithms. A key generation algorithm $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$ takes as input a security parameter 1^λ, and outputs the public-private key pair $(\mathsf{pk}, \mathsf{sk})$. A signature algorithm $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$ takes as input the private key sk and message $m \in \{0, 1\}^*$, and outputs the signature σ. A verification algorithm $0/1 \leftarrow \mathsf{Vf}(\mathsf{pk}, m, \sigma)$, outputs 1 if σ is a valid signature for the message m under the public key pk, otherwise it outputs 0.

Threshold Secret Sharing. A (t, n)-threshold secret sharing scheme Π_{TSS} of a secret x consists of n shares $x_1, ..., x_n$. For any set $|\mathsf{S}| \geq t + 1$, there exists an efficient algorithm $x \leftarrow \mathsf{Recon}(\{x_i\}_{i\in[\mathsf{S}]})$ that takes as input $t + 1$ of these shares and outputs the secret x. However, if only t or fewer shares are provided, no information about the secret x will be revealed.

Threshold Signatures. The (t, n)-threshold signature scheme Π_{Th} is used to distribute signatures to a group of n players $P_1, ..., P_n$, such that any at least $t + 1$ players can co-generate a signature, while t or fewer players cannot. More formally, a (t, n)-threshold signature scheme Π_{Th} for a signature scheme $\Pi_{\mathsf{DS}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ consists of two parts (Gennaro and Goldfeder 2018):

- $(\mathsf{pk}, \mathsf{sk}_1, ..., \mathsf{sk}_n) \leftarrow \mathsf{ThKGen}(1^\lambda, t, n)$. This distributed key generation protocol takes as input a security parameter 1^λ, and outputs a public key pk and P_i's private key share sk_i to each player P_i. The private key shares $\mathsf{sk}_1, ..., \mathsf{sk}_n$ form a (t, n)-threshold secret sharing of the private key sk.
- $\sigma \leftarrow \mathsf{ThSign}\langle\{P_i(\mathsf{sk}_i, m)\}_{i\in[\mathsf{S}]}\rangle$. For $|\mathsf{S}| \geq t + 1$, this distributed signing protocol takes as public input a message $m \in \{0, 1\}^*$ to be signed as well as a private input sk_i from each player P_i, and outputs a signature $\sigma \in \{\Pi_{\mathsf{DS}}.\mathsf{Sign}(\mathsf{sk}, m)\}$.

Notice that $\Pi_{\mathsf{Th}}.\mathsf{ThSign}$ will output a valid signature under the centralized signing protocol $\Pi_{\mathsf{DS}}.\mathsf{Sign}$. Therefore, the verification algorithm of both Π_{Th} and Π_{DS} remains the same, denoted as $\Pi_{\mathsf{DS}}.\mathsf{Vf}$.

Verifiable Timed Signatures. The VTS algorithm (Thyagarajan et al. 2020) is a variant of verifiably encrypted signatures (Boneh et al. 2003; Hanser et al. 2015), designed to decrypt and reveal the signature without TTP. In VTS, the committer generates a timed commitment C for a signature σ of a message m under the public key pk. This commitment C can only be opened after a predefined time T (chosen arbitrarily by the committer). In addition, the committer generates a proof π to prove that C contains a valid signature σ. Its formal definition is as follows:

Definition 4 (*Verifiable Timed Signatures* (Thyagarajan et al. 2020)). A VTS scheme Π_{VTS} for a signature scheme $\Pi_{\mathsf{DS}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ consists of the following four operations:

- $(C, \pi) \leftarrow \mathsf{Commit}(\sigma, \mathsf{T})$: This randomized commit algorithm takes as input a signature σ (generated using $\Pi_{\mathsf{DS}}.\mathsf{Sign}(\mathsf{sk}, m)$) and a hiding time T, and outputs a commitment C and a proof π.
- $0/1 \leftarrow \mathsf{Verify}(\mathsf{pk}, m, C, \pi)$: This verify algorithm takes as input a public key pk, a message m, a commitment C of hardness T and a proof π. It outputs 1 if and only if the value σ embedded in C satisfies $\Pi_{\mathsf{DS}}.\mathsf{Vf}(\mathsf{pk}, m, \sigma) = 1$, otherwise it outputs 0.
- $(\sigma, r) \leftarrow \mathsf{Open}(C)$: This open algorithm is run by the committer, takes as input a commitment C, and outputs the committed signature σ and the randomness r used in generating C.
- $\sigma \leftarrow \mathsf{ForceOp}(C)$: This force open algorithm takes as input a commitment C, and outputs the committed signature σ.

Commitment schemes

A commitment scheme Π_{Com} consists of three algorithms. A generation algorithm $h \leftarrow \mathsf{Gen}(1^{\lambda})$ takes as input a security parameter 1^{λ}, and outputs a public value h. A commit algorithm $(c, d) \leftarrow \mathsf{Com}(h, m)$ takes as input a public value h and a message $m \in \{0,1\}^*$, and outputs a commitment c and an opening value d. A verification algorithm $0/1 \leftarrow \mathsf{Vf}(h, m, c, d)$, outputs 1 if the verification succeeds, otherwise, it outputs 0.

The model of ACCS

In this section, we give the model and the security definitions of ACCS.

System model

Consider two distributed ledgers represented as $\mathbb{B}_1$ and $\mathbb{B}_2$. We assume that there are two players, P_1 and P_2, where P_1 owns asset v_1 on $\mathbb{B}_1$, and P_2 owns asset v_2 on $\mathbb{B}_2$, and they want to exchange assets. The players communicate using secure and authenticated channels, where messages are public and sent sequentially. We assume the network to be synchronous, which is widely adopted by blockchain protocols (Garay et al. 2015; Zamani et al. 2018; Kiayias et al. 2017; Luu et al. 2016).

In this model, we use the public key to represent the wallet address and the private key to represent asset ownership. Moreover, a transaction is used to represent the process of transferring assets between addresses. For example, $\mathsf{tx}(\mathsf{pk}, \mathsf{pk}', v)$ denotes the transfer of asset v from the address corresponding to pk to the address corresponding to pk'. A valid transaction must be signed by a private key corresponding to the source address of the asset.

Our system model consists of 8 algorithms and 2 protocols:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda})$. On input a security parameter 1^{λ}, the algorithm outputs public parameters pp.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^{\lambda})$. On input a security parameter 1^{λ}, the algorithm outputs a public-private key pair $(\mathsf{pk}, \mathsf{sk})$.
- $(\mathsf{pk}', \mathsf{sk}_1', \mathsf{sk}_2') \leftarrow \mathsf{JointKeyGen}(1^{\lambda})$. On input a security parameter 1^{λ}, the algorithm outputs a public key pk' to each player, and a private output sk_i' to P_i as his private key share.
- $\mathsf{tx} \leftarrow \mathsf{GenTx}(\mathsf{pk}, \mathsf{pk}', v)$. On input public keys $\mathsf{pk}, \mathsf{pk}'$ and asset v, the algorithm outputs a transaction tx.
- $((\mathsf{com}, \pi), (\sigma)) \leftarrow \mathsf{CommitRfnd}\langle P_1(\mathsf{T}, \mathsf{sk}_1', \mathsf{pk}', \mathsf{tx}), P_2(\mathsf{T}, \mathsf{sk}_2', \mathsf{pk}', \mathsf{tx})\rangle$. When P_1 inputs a timed parameter T, a private key share sk_1', a public key pk' and a transaction tx, and P_2 inputs a timed parameter T, a private key share sk_2', a public key pk' and a transaction tx, the protocol outputs a timed commitment com, which commits to the signature of the transaction, and its corresponding proof π to P_1. It also outputs a signature σ of the transaction to P_2.
- $0/1 \leftarrow \mathsf{Verify}(\mathsf{pk}', \mathsf{tx}, \mathsf{com}, \pi)$. On input a public key pk', a transaction tx, a timed commitment com and its corresponding proof π, the algorithm outputs 0 to indicate validation failure, and outputs 1 to indicate validation success.
- $\sigma' \leftarrow \mathsf{Freeze}(\mathsf{sk}, \mathsf{tx}')$. On input a private key sk and a transaction tx', the algorithm outputs a signature σ' of the transaction.
- $((\mathsf{sk}_2''), (\mathsf{sk}_1')) \leftarrow \mathsf{Exchange}\langle P_1(\mathsf{sk}_1'), P_2(\mathsf{sk}_2'')\rangle$. When P_1 and P_2 respectively input their private key shares sk_1' and sk_2'', the protocol outputs sk_2'' to P_1 and sk_1' to P_2.
- $\sigma'' \leftarrow \mathsf{Complete}(\mathsf{sk}_1', \mathsf{sk}_2', \mathsf{tx}'')$. On input private key shares sk_1' and sk_2', and a transaction tx'', the algorithm outputs a signature σ'' of the transaction.
- $\sigma \leftarrow \mathsf{UnFreeze}(\mathsf{com})$. On input a timed commitment com, the algorithm outputs a signature σ embedded in the commitment.

Security definition

In this section, we discuss the security threats and give the formal security definitions of ACCS. We consider a PPT adversary who corrupts one of the players. The corrupt player can deviate from the protocol in any arbitrary manner, e.g., by sending invalid or inconsistent messages, or aborting interactions. We assume that once a player obtains the private key share of joint address from the other player, he will immediately generate a signature of the swap transaction and publish it on distributed ledger to transfer assets in the joint address. We denote the player's state using a state set $\mathsf{state} = \{0, 1\}$, where $\mathsf{state}_\mathsf{A} = 0$ represents a failed swap for player A, meaning that player A does not obtain player B's asset and successfully refund, while $\mathsf{state}_\mathsf{A} = 1$ represents a successful swap for player A, meaning that player A obtains player B's asset and loses his own asset. We now give the formal security definitions as follows.

Definition 5 (*Completeness*). The completeness of ACCS guarantees that there exists a negligible function $\mathsf{negl}(\lambda)$, such that when all players execute honestly, the execution result of ACCS is:

$$\Pr\left[(\mathsf{state}_\mathsf{A} = 0 \wedge \mathsf{state}_\mathsf{B} = 1) \vee (\mathsf{state}_\mathsf{A} = 1 \wedge \mathsf{state}_\mathsf{B} = 0)\right] \leq \mathsf{negl}(\lambda)$$

Intuitively, it means that if both players are honest, then ACCS must end with both players' states being either all 0 or all 1. To better understand this definition, we discuss two situations: timeout and no timeout. Timeout means that a player fails to complete the asset swap within the predefined time. In this case, the completeness requires the states of both players are 0, i.e., $\mathsf{state}_\mathsf{A} = 0 \wedge \mathsf{state}_\mathsf{B} = 0$. On the contrary, no timeout means that the players complete the asset swap within the predefined time. In this case, the completeness requires the states of both players are 1, i.e., $\mathsf{state}_\mathsf{A} = 1 \wedge \mathsf{state}_\mathsf{B} = 1$.

Definition 6 (*Soundness*). The soundness of ACCS guarantees that there exists a negligible function $\mathsf{negl}(\lambda)$, such that for all PPT adversaries $\mathcal{A}$ and all $\lambda \in \mathbb{N}$, given the correct public key pk, transaction tx and timed parameter T generated during protocol execution, we have:

$$\Pr\left[\begin{matrix}(\mathsf{com}^*, \pi^*) \leftarrow \mathcal{A}(1^\lambda), \sigma^* \leftarrow \mathsf{UnFreeze}(\mathsf{com}^*) : \\ 1 \leftarrow \mathsf{Verify}(\mathsf{pk}, \mathsf{tx}, \mathsf{com}^*, \pi^*) \wedge 0 \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Vf}(\mathsf{pk}, \mathsf{tx}, \sigma^*)\end{matrix}\right] \leq \mathsf{negl}(\lambda)$$

Intuitively, it means that a PPT adversary $\mathcal{A}$ cannot generate a commitment com^* and its corresponding proof π^* such that they can be successfully verified by the Verify algorithm. However, when using the $\mathsf{UnFreeze}$ algorithm to open the commitment after the predefined time, the obtained result is not the correct signature.

Definition 7 (*Timed Privacy*). We use the symbols $\ll$ to represent "much less than", and $\gg$ to represent "much greater than", where $b \ll a$ implies that the result of $a + b$ is approximately equal to a. In addition, the symbol const_1 represents a constant, specified by the system in the setup phase. We say that an ACCS satisfies timed privacy if both of the following conditions hold:

- There exists a negligible function $\mathsf{negl}(\lambda)$ such that for all PPT adversaries $\mathcal{A}$ with running time $t \ll \mathsf{T} - \mathsf{const}_1$, all transaction messages $m \in \{0, 1\}^*$ and all $\lambda \in \mathbb{N}$, given the correct public key pk and transaction tx generated during protocol execution, we have:

$$\Pr\left[\sigma^* \leftarrow \mathcal{A}(1^\lambda) : 1 \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Vf}(\mathsf{pk}, \mathsf{tx}, \sigma^*)\right] \leq \mathsf{negl}(\lambda)$$

- There exists a negligible function $\mathsf{negl}(\lambda)$ such that for all honest players A with running time $t \gg \mathsf{T} + \mathsf{const}_1$, all transaction messages $m \in \{0, 1\}^*$ and all $\lambda \in \mathbb{N}$, given the correct public key pk, transaction tx and commitment com generated by the protocol, we have:

$$\Pr\left[\sigma \leftarrow \mathsf{UnFreeze}(\mathsf{com}) : 0 \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Vf}(\mathsf{pk}, \mathsf{tx}, \sigma)\right] \leq \mathsf{negl}(\lambda)$$

Intuitively, the former condition implies that a PPT adversary $\mathcal{A}$ cannot obtain a valid signature when the running time t is much less than $\mathsf{T} - \mathsf{const}_1$. The latter

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$[2]. Given a security parameter 1^λ, the algorithm does:
  1. Runs $h \leftarrow \Pi_{\mathsf{Com}}.\mathsf{Gen}(1^\lambda)$.
  2. Sets v_1 and v_2 as assets of P_1 and P_2.
  3. Sets constants const_1 and const_2, let $l = \mathsf{const}_2$ and $\kappa = \lceil \frac{\lambda}{l} \rceil$.
  4. Sets timed parameters T_1 and $\triangle$, where $\mathsf{T}_1, \triangle \in \mathbb{N}$, and let $\mathsf{T}_2 = \mathsf{T}_1 - \triangle$.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. Given a security parameter 1^λ, the algorithm runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Pi_{\mathsf{DS}}.\mathsf{KGen}(1^\lambda)$, and takes $(\mathsf{pk}, \mathsf{sk})$ as output.
- $(\mathsf{pk}', \mathsf{sk}'_1, \mathsf{sk}'_2) \leftarrow \mathsf{JointKeyGen}(1^\lambda)$. Given a security parameter 1^λ, the algorithm runs $(\mathsf{pk}', \mathsf{sk}'_1, \mathsf{sk}'_2) \leftarrow \Pi_{\mathsf{Th}}.\mathsf{ThKGen}(1^\lambda, 1, 2)$, then outputs pk' and sk'_1 as the output for P_1, and pk' and sk'_2 as the output for P_2.
- $\mathsf{tx} \leftarrow \mathsf{GenTx}(\mathsf{pk}, \mathsf{pk}', v)$. Given public keys pk, pk' and asset v, the algorithm generates a valid transaction $\mathsf{tx} := \mathsf{tx}(\mathsf{pk}, \mathsf{pk}', v)$ as output.
- $((\mathsf{com}, \pi), (\sigma)) \leftarrow \mathsf{CommitRfnd}\langle P_1(\mathsf{T}, \mathsf{sk}'_1, \mathsf{pk}', \mathsf{tx}), P_2(\mathsf{T}, \mathsf{sk}'_2, \mathsf{pk}', \mathsf{tx})\rangle$. The protocol does:
  1. P_1 and P_2 execute the protocol $(\perp, \sigma) \leftarrow \Pi_{\mathsf{Th}}.\mathsf{ThSign}\langle P_1(\mathsf{sk}'_1, \mathsf{tx}), P_2(\mathsf{sk}'_2, \mathsf{tx})\rangle$. As a result of the protocol, P_2 obtains σ as his output.
  2. P_2 runs $(\mathsf{com}, \pi) \leftarrow \Pi_{\mathsf{VTS}}.\mathsf{Commit}(\sigma, \mathsf{T})$, and sends (com, π) to P_1.
- $0/1 \leftarrow \mathsf{Verify}(\mathsf{pk}', \mathsf{tx}, \mathsf{com}, \pi)$. Given a public key pk', a transaction tx, a timed commitment com and its corresponding proof π, the algorithm runs $0/1 \leftarrow \Pi_{\mathsf{VTS}}.\mathsf{Verify}(\mathsf{pk}', \mathsf{tx}, \mathsf{com}, \pi)$ and takes the result as the output.
- $\sigma' \leftarrow \mathsf{Freeze}(\mathsf{sk}, \mathsf{tx}')$. Given a private key sk and a transaction tx', the algorithm runs $\sigma' \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}(\mathsf{sk}, \mathsf{tx}')$ and takes σ' as the output.

Fig. 1 Generic atomic cross-chain swaps scheme - Part I [2]Note: In the practical application of the system model, variables are usually defined to represent various parameters and are initially set to default values, such as 0, and then assigned to the actual values of specific parameters at runtime. To simplify the representation, the variables are directly assigned values in the Setup algorithm

condition implies that an honest player A will not obtain an invalid signature after running the $\mathsf{UnFreeze}$ algorithm when the running time t is much greater than $\mathsf{T} + \mathsf{const}_1$.

Definition 8 (*Partial Fairness*). We define the partial fairness of ACCS through the following experiment. Assuming that the adversary $\mathcal{A}$ corrupts player B. Therefore, player A is honest. If $\mathcal{A}$ aborts before the end of ACCS, let $\mathsf{sk}^*_{\mathsf{B}}[1 \ldots i]$ represents the partial private key share obtained by player A. At this point, we assume that player A will randomly select some elements in the remaining space of size $2^{\lambda - i}$ as B's tentative private key share, denoted as $\mathsf{sk}'_{\mathsf{B}}$. Similarly, $\mathcal{A}$ will also output the tentative private key share of A, denoted as $\mathsf{sk}'_{\mathsf{A}}$. We use const_2 to represent a constant, specified by the system in the setup phase.

The partial fairness of ACCS guarantees that there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all PPT adversaries $\mathcal{A}$, we have:

$$|\Pr[\mathsf{sk}_{\mathsf{A}}^{(\mathrm{AB})} = sk'_A] - \Pr[\mathsf{sk}_{\mathsf{B}}^{(\mathrm{BA})} = \mathsf{sk}'_B]| \leq \frac{2^{\mathsf{const}_2}}{2^{\lambda - i}} + \mathsf{negl}(\lambda)$$

Our generic atomic cross-chain swaps scheme

In this section, first we present a generic scheme for one-to-one atomic swap, which shows the feasibility and be seen as a framework for further optimizations. Second, we explain the entire process of an atomic swap between user A and B as a concrete example. Finally, we prove the security of the scheme in our model.

- $((\mathsf{sk}_2''), (\mathsf{sk}_1')) \leftarrow \mathsf{Exchange}\langle P_1(\mathsf{sk}_1'), P_2(\mathsf{sk}_2'')\rangle$. The protocol does the following:
  1. P_1 runs $\mathsf{sk}_1'[\cdot] \leftarrow \mathsf{DivideSegment}(\mathsf{sk}_1', l, \kappa)$ to get κ-Segmentation of sk_1'.
  2. P_2 runs $\mathsf{sk}_2''[\cdot] \leftarrow \mathsf{DivideSegment}(\mathsf{sk}_2'', l, \kappa)$ to get κ-Segmentation of sk_2''.
  3. P_1 runs $(\vec{r'}, \vec{\mathsf{com}}', (\pi'_{rpi})_{i\in[\kappa]}, \pi'_{cc}, \pi'_{\mathsf{pk}}) \leftarrow \mathsf{ComProofGen}(\mathsf{sk}_1'[\cdot])$, and sends $(\vec{\mathsf{com}}', (\pi'_{rpi})_{i\in[\kappa]}, \pi'_{cc}, \pi'_{\mathsf{pk}})$ to P_2.
  4. P_2 runs $(\vec{r''}, \vec{\mathsf{com}}'', (\pi''_{rpi})_{i\in[\kappa]}, \pi''_{cc}, \pi''_{\mathsf{pk}}) \leftarrow \mathsf{ComProofGen}(\mathsf{sk}_2''[\cdot])$, and sends $(\vec{\mathsf{com}}'', (\pi''_{rpi})_{i\in[\kappa]}, \pi''_{cc}, \pi''_{\mathsf{pk}})$ to P_1.
  5. P_1 runs $0/1 \leftarrow \mathsf{ComProofCheck}((\pi'_{rpi})_{i\in[\kappa]}, \pi'_{cc}, \pi'_{\mathsf{pk}})$, if outputs 0, aborts.
  6. P_2 runs $0/1 \leftarrow \mathsf{ComProofCheck}((\pi''_{rpi})_{i\in[\kappa]}, \pi''_{cc}, \pi''_{\mathsf{pk}})$, if outputs 0, aborts.
  7. For each $i \in [\kappa]$, P_1 and P_2 alternately open the commitment of segment and verify it. Specifically, they do the following:
     - P_1 runs $(\mathsf{sk}_1'[i], r_i') \leftarrow \mathsf{SegComOpen}(\mathsf{sk}_1', \vec{r'}, i)$ to get the opening value $(\mathsf{sk}_1'[i], r_i')$ of the i-th commitment, and sends it to P_2.
     - P_2 runs $0/1 \leftarrow \mathsf{SegComCheck}(h, \mathsf{sk}_1'[i], \mathsf{com}', (\mathsf{sk}_1'[i], r_i'))$ to verify, if outputs 0, aborts.
     - Similarly, P_2 runs $(\mathsf{sk}_2''[i], r_i'') \leftarrow \mathsf{SegComOpen}(\mathsf{sk}_2'', \vec{r''}, i)$ to get the opening value $(\mathsf{sk}_2''[i], r_i'')$ of the i-th commitment, and sends it to P_1.
     - P_1 runs $0/1 \leftarrow \mathsf{SegComCheck}(h, \mathsf{sk}_2''[i], \mathsf{com}'', (\mathsf{sk}_2''[i], r_i''))$ to verify, if outputs 0, aborts.
  8. P_1 runs $\mathsf{sk}_2'' \leftarrow \mathsf{ConnectSegment}(\mathsf{sk}_2''[\cdot])$ to get sk_2'' as output.
  9. P_2 runs $\mathsf{sk}_1' \leftarrow \mathsf{ConnectSegment}(\mathsf{sk}_1'[\cdot])$ to get sk_1' as output.
- $\sigma'' \leftarrow \mathsf{Complete}(\mathsf{sk}_1', \mathsf{sk}_2', \mathsf{tx}'')$. Given private key shares sk_1' and sk_2', and a transaction tx'', the algorithm does the following:
  1. Runs $\mathsf{sk}' \leftarrow \Pi_{\mathsf{TSS}}.\mathsf{Recon}(\mathsf{sk}_1', \mathsf{sk}_2')$.
  2. Runs $\sigma'' \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}(\mathsf{sk}', \mathsf{tx}'')$ and takes σ'' as output.
- $\sigma \leftarrow \mathsf{UnFreeze}(\mathsf{com})$. Given a timed commitment com, the algorithm runs $\sigma \leftarrow \Pi_{\mathsf{VTS}}.\mathsf{ForceOp}(\mathsf{com})$ and takes σ as output.

Fig. 2 Generic atomic cross-chain swaps scheme - Part II

Given the digital signature scheme Π_{DS}, commitment scheme Π_{Com}, threshold signatures scheme Π_{Th}, threshold secret sharing scheme Π_{TSS}, VTS scheme Π_{VTS}, range proofs protocol Π_{RP} and NIZK protocol Π_{NIZK}, we constructed our generic scheme based on the system model, which consists of 8 algorithms and 2 protocols. The complete generic scheme is shown in Figs. 1 and 2, where the complete $\mathsf{Exchange}$ protocol is postponed to Appendix A.

Here, we provide an example of the generic construction that considers an atomic swap between two users, namely user A and user B. In this scenario, A owns asset v_1 on $\mathbb{B}_1$, while B owns asset v_2 on $\mathbb{B}_2$. To simplify the description, we define that the payment from A to B involves the keys, transactions and signatures with (AB), while the payment from B to A involves the keys, transactions and signatures with (BA). The complete process is postponed to Appendix B.

Theorem 1 *Assume the underlying digital signature scheme* Π_{DS}, *commitment scheme* Π_{Com}, *threshold signatures scheme* Π_{Th}, *threshold secret sharing scheme* Π_{TSS}, *VTS scheme* Π_{VTS}, *range proofs protocol* Π_{RP} *and NIZK protocol* Π_{NIZK} *are secure, the atomic swap protocol described in Appendix B is secure, and has the properties of completeness, soundness, timed privacy and partial fairness.*

The proof is postponed to Appendix C.

```
**Common reference string:** Input $(1^\lambda, \kappa)$
  1. $(p, \mathbb{G}, \mathbb{G}_\mathsf{T}, e, G) \leftarrow \mathsf{BMGen}(1^\lambda)$.
  2. $s \xleftarrow{R} \mathbb{Z}_p^*$. Here, $s$ is required to contain a factor $2^l$.
  3. Return $\mathsf{CRS} = \langle (p, \mathbb{G}, \mathbb{G}_\mathsf{T}, e), (G_0, G_1, ..., G_\kappa) \rangle$, where $\forall i \in [0, \kappa] : G_i = s^i G$.
**Proof:** Input $(\mathsf{CRS}, \theta, r_1, ..., r_\kappa)$
  1. Check that $D = \theta G$. If not, return $\perp$.
  2. For each $i \in [\kappa]$, compute $C_i = r_i G + \theta[i] G_i$.
  3. Compute $r = \sum_{i \in [\kappa]} r_i$.
  4. Compute $U = \frac{1}{s} D' = \frac{1}{s}(\sum_{i \in [\kappa]} C_i - rG)$ using the segment vector $\theta[\cdot]$ and the common reference string $\mathsf{CRS}$.
  5. Compute the formal polynomial $W(\cdot)$ such that $P(X) - P(2^l) = W(X)(X - 2^l)$, where $P(X) = \sum_{i \in [\kappa]} \theta[i] X^{i-1}$, and $P(2^l) = \sum_{i \in [\kappa]} \theta[i] 2^{l(i-1)} = \theta$.
  6. Compute $V = W(s)G$ using the common reference string $\mathsf{CRS}$ and the coefficients of the formal polynomial $W(\cdot)$.
  7. Return $\pi = (r, U, V)$.
**Verification:** Input $(\mathsf{CRS}, C, \pi)$
  1. Parse $C$ as $(D, (C_i)_{i \in [\kappa]})$.
  2. Parse $\pi$ as $(r, U, V)$.
  3. Check that $r \in \mathbb{Z}_p^*$.
  4. Check that $(U, V, D, C_1, ..., C_\kappa) \in \mathbb{G}^{\kappa+3}$.
  5. Compute $D' = \sum_{i \in [\kappa]} C_i - rG$.
  6. Check that $e(D', G) = e(U, G_1)$.
  7. Check that $e(U - D, G) = e(V, G_1 - 2^l G)$.
  8. Return $\perp$ if any test fails, otherwise return $valid$.
```

Fig. 3 The NIZK protocol Π_{CCNIZK}

NIZK for the correct commitment of DL

In this section, we propose a tool, the NIZK protocol for the correct commitment of DL, to instantiate our generic scheme for ECDLP-based signatures. It is designed to prove that the commitment vector $(C_i)_{i\in[\kappa]}$ encrypts the κ-Segmentation of θ without revealing any additional knowledge. Here, θ represents the DL of a group element D in the group $\mathbb{G}$. This proof consists of two parts: the segmented range proof, denoted as Π_{BulletRP}, and the NIZK proof of correct commitment, denoted as Π_{CCNIZK}. Moreover, since our scheme only involves cross-chain swap between two parties, the zero-knowledge proof protocol does not need the property of non-malleability (Sahai 1999).

Segmentation range proofs

Our construction is based on a slight variation of the Pedersen commitment scheme. Consider a common reference string $\mathsf{CRS} = (G, sG, s^2G, ..., s^N G) = (G_0, G_1, ..., G_N)$, where $s \xleftarrow{R} \mathbb{Z}_p^*$ is a trapdoor. Here, we require that s contains a factor 2^l. To commit the segment $\theta[i]$ at position i using the randomness $r_i \in \mathbb{Z}_p$, we compute $\mathsf{Commit}(\theta[i], r_i, i) = C_i = r_i G + \theta[i] G_i$. The commitment of randomness $\vec{r} = (r_i)_{i\in[N]}$ to the vector $\theta[\cdot] = (\theta[1], ..., \theta[N])$ is realized through a vector formed by the commitment to each segment in position i. This vector is denoted as $\vec{C} = (C_i)_{i\in[N]}$. Alternatively, we can express this relationship as $\vec{C} = \mathsf{Commit}(\theta[\cdot], \vec{r})$. For each commitment C_i, we use the Bulletproofs range proof Π_{BulletRP} to prove that it encrypts a segment with a binary length of l (if the length is less than l, we fill in the leading zeros), indicating that the value of the segment falls within the interval $[0, ..., 2^l - 1]$.

Zero-knowledge proof protocol of correct commitment

In this section, we present a NIZK protocol Π_{CCNIZK}. Let $\theta \xleftarrow{R} \mathbb{Z}_p^*$. Consider the commitment to the κ-Segmentation of θ. This commitment is denoted as $\vec{C} = (C_i)_{i\in[\kappa]} = (r_i G + \theta[i]G_i)_{i\in[\kappa]}$, where $r_i \in \mathbb{Z}_p^*$ for each $i \in [\kappa]$. Additionally, $D = \theta G$. Π_{CCNIZK} is used to prove that each commitment item C_i at position i indeed encrypts the i-th item in the κ-Segmentation of θ. Here, θ represents the DL of D, which is denoted as $D = \theta G$. This proof enables us to create segmented commitments to the private key share of the joint address, and gradually release each segment of the private key share without revealing any additional knowledge. The complete NIZK protocol Π_{CCNIZK} is shown in Fig. 3.

For the given $\theta \in \mathbb{Z}_p^*$ and $\vec{C} = (r_i G + \theta[i]G_i)_{i\in[\kappa]}$, our proof protocol does as follows:

(1) The prover computes $D' = \sum_{i\in[\kappa]} C_i - rG = \sum_{i\in[\kappa]} r_i G + \sum_{i\in[\kappa]} \theta[i]G_i - rG$, where $r = \sum_{i\in[\kappa]} r_i$. In this step, the prover obtains a compressed representation of the segmented vector commitment, and removing the randomness.

(2) Using the common reference string CRS and the segmented vector $\theta[\cdot]$, the prover computes $U = \frac{1}{s}D' = \frac{1}{s}(\sum_{i\in[\kappa]} \theta[i]G_i) = \sum_{i\in[\kappa]} \theta[i]G_{i-1}$, where $G_0 = G$. This step ensures that once the equation $e(D', G) = e(U, G_1)$ holds, it can be proven that r is indeed the accumulation of the randomness in the segmented vector commitments. If $r \neq \sum_{i\in[\kappa]} r_i$, it would imply a break in certain assumptions, which is impossible.

(3) The prover lets $U = \sum_{i\in[\kappa]} \theta[i]G_{i-1} = \sum_{i\in[\kappa]} \theta[i]s^{i-1}G = P(s)G$, where the symbol $P(\cdot)$ represents the polynomial $P(X) = \sum_{i\in[\kappa]} \theta[i]X^{i-1}$, and so $P(2^l) = \sum_{i\in[\kappa]} \theta[i]2^{l(i-1)} = \theta$. To prove that the compressed segmented vector commitment $U = \sum_{i\in[\kappa]} \theta[i]G_{i-1}$ is "equivalent" a single commitment θG, the prover needs to prove that $P(s) - P(2^l) = P(s) - \theta$ can be divisible by $s - 2^l$. Specifically, the prover does:

- Computes the coefficients of the formal polynomial $W(\cdot)$ such that $P(X) - P(2^l) = W(X)(X - 2^l)$.
- Computes $V = W(s)G$ using the common reference string CRS.

Therefore, once the equation $e(U - D, G) = e(V, G_1 - 2^l G) = e(V, (s - 2^l)G)$ holds, it can be ensured that the coefficients of the polynomial $P(\cdot)$ correspond to the κ-Segmentation of θ (that is, $\theta = P(2^l)$).

Theorem 2 *The protocol in Fig. 3 is a NIZK proof that proves the κ-Segments of D's elliptic curve DL correspond to the committed segment vectors in $(C_i)_{i\in[\kappa]}$. This NIZK proof has perfect completeness, computational soundness and perfect zero-knowledge under the assumption of κ-SDH.*

The proof is postponed to Appendix D.

Instantiation: ACCS for ECDLP-based signatures

Instantiation of the generic construction

To show the practicability of our generic construction, we instantiate it in ECDLP-based blockchains. Specifically, we use the ECDSA signature scheme to instantiate it, where we instantiate Π_{DS} as Π_{ECDSA}. In addition, we instantiate each algorithm in the $\mathsf{Exchange}$ protocol as follows: First, we employ the Bulletproofs range proof protocol (Bünz et al. 2018) to implement the range proof protocol, resulting in Π_{RP} being instantiated as Π_{BulletRP}. Second, we utilize the Pedersen commitment scheme to implement the commitment scheme, leading to Π_{Com} being instantiated as Π_{PedCom}. Third, we use the NIZK protocol Π_{DLNIZK} proposed in (Camenisch and Stadler 1997) to generat the proof π_{pk}. Finally, we use the NIZK protocol Π_{CCNIZK} described in Sect. to generate the proof π_{cc}.

Extensions

In this section, we propose some possible extensions of our scheme.

Replace the VTS Algorithm with the Verifiable Timed Discrete Logarithm (VTD) Algorithm. To enhance the efficiency of our scheme, we can replace the VTS algorithm with the VTD algorithm (Thyagarajan et al. 2020). In VTD, the committer no longer needs to prove that the signature of the commitment is valid, but only needs to prove that the commitment value is the DL of a known group element in the group, which is a simpler algebraic statement (Thyagarajan et al. 2022). Therefore, the VTD algorithm is more efficient in commitment generation and verification than VTS, which can significantly improve the efficiency in freeze phase.

In our ECDLP-based ACCS scheme, using the VTD algorithm, players can make a commitment to their own private key share associated with the joint address. Specifically, user B generates a commitment $\mathsf{com}^{(A)}$ for his private key share $\mathsf{sk}_B^{(AB)}$ and sends it to user A, while user A generates a commitment $\mathsf{com}^{(\mathsf{B})}$ for his private key share $\mathsf{sk}_\mathsf{A}^{(\mathsf{BA})}$ and sends it to user B. After timeout, they can forcibly open the commitment to obtain the other user's private key share. By combining their own private key share with the obtained one, each user can reconstruct the complete private key of the joint address, allowing them to refund their respective assets.

In terms of security, replacing VTS with the VTD algorithm will not compromise the security of our protocol, as both VTS and VTD have the same properties of soundness and privacy (Thyagarajan et al. 2020; Thyagarajan 2022).

Multi-Asset Atomic Swaps. The existing HTLC has functional limitations as it only supports one-to-one asset swaps. However, the current cryptocurrency prices vary greatly, making it more difficult for two exchanges on different blockchains to match. In addition, there are also many users who have a need for multi-asset swaps, such as the exchanges. They hold assets in different blockchains and have different combinations of swap needs. This makes further research on multi-asset ACCS meaningful, where A can exchange his Nakamoto (2008), Buterin (2013) and Litecoin (2011) for B's Noether (2014) and Schwartz et al. (2014) through a single ACCS. Therefore, we provide two ways to achieve multi-asset ACCS for our scheme, one is to swap assets on the same curve, and the other is to swap assets cross different curves.

- *The same curve swaps.* First, we consider the case where all coins to be swapped between A and B are on blockchains that use the same elliptic curve. That is, all coins of A are on blockchains that use the elliptic curve cyclic group $\mathbb{G}_1$, with generator G_1 and prime order value p_1. And all coins of B are on blockchains that use the elliptic curve cyclic group $\mathbb{G}_2$, with generator G_2 and prime order value p_2. In addition, all coins participating in the swap are on ECDLP-based blockchains. Therefore, if both players use the same private key share to generate the joint addresses on different blockchains, then it can be ensured that multiple coins can be swapped with only once private key share exchange. This greatly improves the efficiency of multi-asset ACCS. Moreover, using the same private key share will not compromise the privacy of the players. Because the other player can generate a random private key share, resulting in different final joint addresses and ensuring the unlinkability of cross-chain transactions (Deshpande and Herlihy 2020).
- *Cross-curve swaps.* Next, we consider the case where all coins to be swapped between user A and user B are on blockchains that use different elliptic curves. Specifically, all coins of A are on blockchains that use the elliptic curve cyclic groups $(\mathbb{G}_{1i})_{i\in[n]}$, with generators $(G_{1i})_{i\in[n]}$ and prime order values $(p_{1i})_{i\in[n]}$. All coins of B are on blockchains that use the elliptic curve cyclic groups $(\mathbb{G}_{2i})_{i\in[\widetilde{n}]}$, with generators $(G_{2i})_{i\in[\widetilde{n}]}$ and prime order values $(p_{2i})_{i\in[\widetilde{n}]}$. In this case, we can still use the solution employed in the same curve swaps. However, it's important to note that the parameters of the elliptic curves used for each coin are different, resulting in different generated public key shares even when the same private key share are used. Therefore, it is difficult to verify whether two joint addresses are generated by the same private key share. To deal with this challenge, we introduce an additional zero-knowledge proof to prove that different public key shares on different curves have the same DL (i.e., private key share), thereby ensuring the correctness of the generated joint address. This proof can be implemented using NIZK proof mechanism proposed in Noether (2018) or Chase et al. (2022).

Experimental analysis

In this section, we evaluate and analyze our scheme through experiments. First, we evaluate the performance of our NIZK protocol. Next, we instantiate our ACCS scheme for ECDLP-based signatures, and compare it with other ACCS schemes. All experiments are run on the Windows 10 Enterprise LTSC operating system, utilizing an Intel Core i5-6200U CPU @ 2.30GHz, with 8GB of memory and a 465GB hard disk capacity.

Performance analysis of NIZK protocol

Implementation details. To evaluate the practical performance of our NIZK protocol Π_{CCNIZK}, we give a reference implementation of python language. It relies on the ecpy library (Midorikawa 2019) for elliptic curve related operations, and the sympy library (Smith 2023) for scientific calculations. The code runs on the elliptic curve secp256k1, which is also used in Bitcoin. The experimental results show that segment lengths longer than 50 bits have no significant impact on the experimental results, so we set the segment length in the range of 1 bit to 50 bits. We measure the average time of over 1000 runs and report our results in milliseconds.

Time costs and communication costs. We evaluate the time costs of the proof phase and verification phase of the NIZK protocol under different segment lengths, considering both cases of 256-bit and 128-bit private keys. Our experimental results are shown in Fig. 4. The results show that in the case of a 128-bit private key, the running

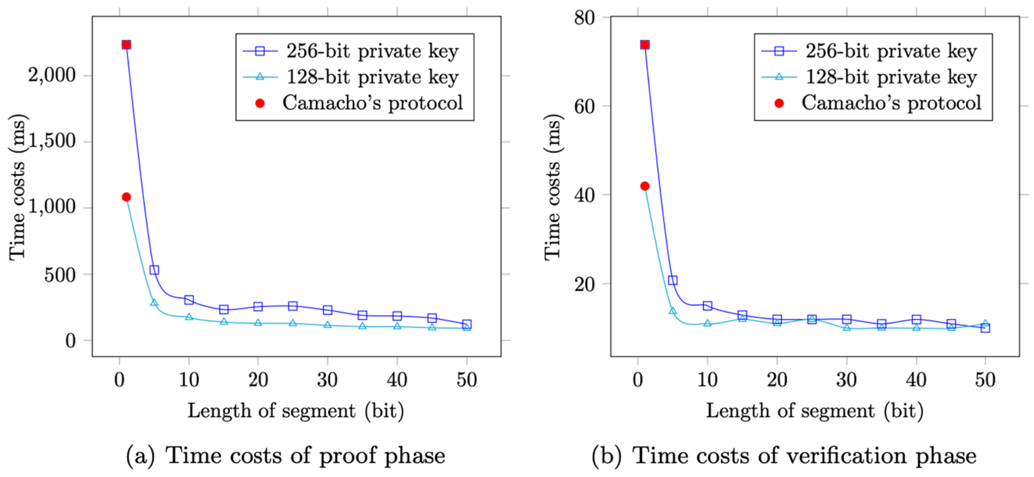

Fig. 4 Time costs of NIZK protocol

time of the proof phase does not exceed 1100 ms, and the verification phase can be completed within 50 ms. In the case of a 256-bit private key, the running time of the proof phase does not exceed 2300 ms, and the verification phase can be completed within 80 ms.

In addition, we also calculate the communication cost of our protocol. We measure the communication cost as the amount of information that each party needs to exchange during protocol execution, which in our protocol is the size of the proof π_{cc}. Specifically, it includes an element on $\mathbb{Z}_p^*$ and two elements on $\mathbb{G}$. When we set the elliptic curve to the secp256k1 curve with a 256-bit group order, the experimental results show that the communication cost of our protocol is approximately 160 bytes.

Comparison. Compared with the original protocol proposed by Camacho (Camacho 2013), for the proof phase, our protocol can reduce the time cost by up to about 91.62% (set 128-bit private key and 50-bit segment length) and 94.56% (set 256-bit private key and 50-bit segment length). For the verification phase, our protocol can reduce the time cost by up to about 76.28% (set 128-bit private key and 45-bit segment length) and 86.44% (set 256-bit private key and 50-bit segment length). In addition, the communication costs of the two protocols are similar. It can be seen that our NIZK protocol greatly reduces the time cost without increasing the communication cost of the original protocol.

Comparison with other schemes

(1) On-chain costs

Implementation details. According to Hanzlik et al. (2022), a typical metric to measure on-chain costs is to evaluate the transaction fees associated with all transactions that appear on-chain in the protocol. Therefore, we define on-chain execution costs measured in USD[1] as the amount of Ethereum transaction fees required to execute each operation: create, redeem and refund. The create operation represents the process of creating an HTLC contract or generating a joint address and depositing assets into it. The redeem operation represents the transfer process of assets in the case of successful swap, and the refund operation represents the transfer process of assets in the case of timeout. We implement and execute our ACCS scheme designed for ECDLP-based signatures on the Ethereum platform, and calculate the gas cost for each operation and an ACCS (including two create operations and two redeem operations). For each experimental result, we conduct 100 tests and take the average.

Comparison. We compare the gas cost of our scheme with other ACCS schemes. Such as, Hatch's HTLC scheme (Hatch 2019) (denoted as Hat19), Lisi's ACCS scheme (Lisi et al. 2020), which includes two parts of implementation, one is the payment permission (denoted as LDM20-I) and the other is the review reward (denoted as LDM20-II), Foundry's scheme (Foundry 2020) (denoted as Fou20), Tsabary's Mutual-Assured-Destruction Hashed Time-Locked Contract (MAD-HTLC) scheme (Tsabary et al. 2021) (denoted as TYM21), and two implementations of Hoenisch (Hoenisch 2020, 2020) (denoted as Hoe20-I and Hoe20-II).

The gas costs of three operations for different scheme are shown in Fig. 5, and the gas costs of an ACCS are shown in Fig. 6. The experimental results show that compared with other schemes, our scheme reduces the gas costs by approximately 84.37% to 96.67% in the create operation. In the redeem operation, our scheme reduces

[1] The costs are in USD as per exchange rates of 19 Oct. 2023: ETH/USD 1555.18. Note that the data shown in Figs. 5 and 6 only retains two decimal places.

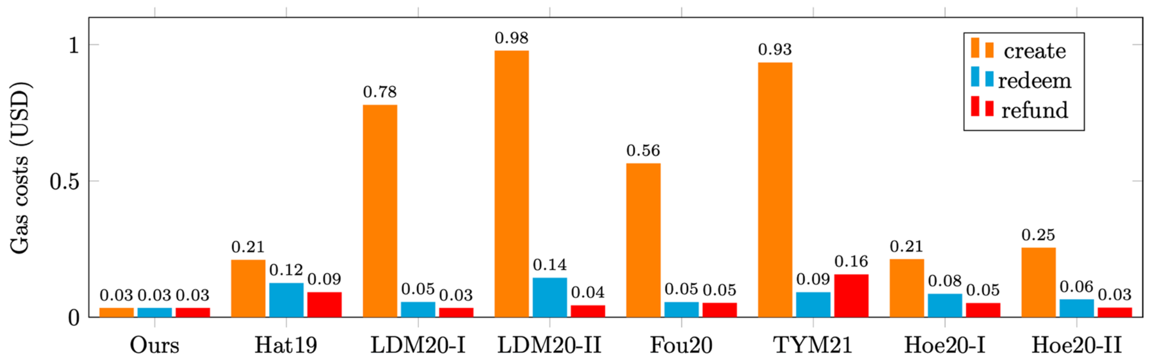

Fig. 5 Gas costs of three operations for different schemes

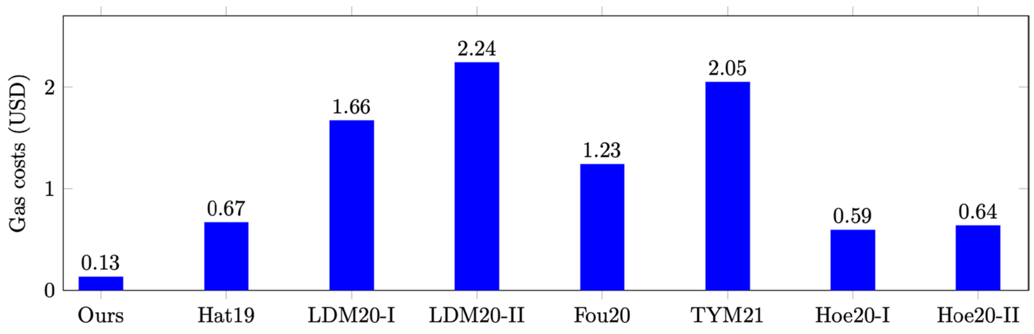

Fig. 6 Gas costs of an ACCS for different schemes

gas costs by approximately 39.54% to 77.17%. In the refund operation, our scheme can reduce gas costs by up to approximately 63.83%. When performing an ACCS, compared with other schemes, our scheme reduces gas costs by approximately 77.91% to 94.17%. This is because our ACCS scheme requires only a standard ETH transfer between two players, which is a basic inexpensive operation in Ethereum. However, HTLC implementations often require significant gas costs to perform the create operation. Therefore, compared to existing schemes, the on-chain gas costs of our scheme are very low.

(2) Off-chain costs

Implementation details. Li et al. (2022) showed that the time costs of ACCS can be roughly divided into on-chain transaction confirmation time and other time overhead of off-chain protocol execution. Since transaction confirmation time is specific to the blockchain platform used, we only consider the off-chain time costs. According to Heilman et al. (2017), we calculate the off-chain time costs of our scheme without considering network latency. We use the libsecp256k1 library (Poelstra 2018) to implement bulletproofs range proofs, and the liblhtlp library (Bhat 2020) to implement the VTS scheme. All experimental results are averages of 1000 runs, and we report the results in seconds.

Time costs of Exchange phase. We evaluate the off-chain time cost of the Exchange phase under different segment lengths, considering both cases of 256-bit and 128-bit private keys. Our experimental results are shown in Fig. 7. The results show that when the private key length is 128 bits, the off-chain time cost in the Exchange phase does not exceed 2.7 s. When the private key length is 256 bits, the off-chain time cost does not exceed 5.3 s.

Comparison I. In the case of a 256-bit private key, we compare the off-chain time costs of our scheme with other schemes in the Exchange phase. Such as, Li's ZeroCross scheme (Li et al. 2022) (denoted as LWL22-I and LWL22-II), Chen's scheme (Chen et al. 2023) (denoted as CYS23), Qin's Blindhub scheme and its optimized scheme (Qin et al. 2023) (denoted as QPM23-I and QPM23-II), Tairi's A^{2}L scheme (Tairi et al. 2021) (denoted as TMM21), and Hanzlik's Sweep-UC scheme (Hanzlik et al. 2022) (denoted as HLT22). Our experimental results are shown in Fig. 8. The results show that the average time cost of our scheme in the Exchange phase is better than other schemes.

Time costs of an ACCS. In the case of a 256-bit private key, we calculate the time costs of our scheme to execute an ACCS. The average off-chain time cost of our scheme is 16.85 s, which has no significant impact on the

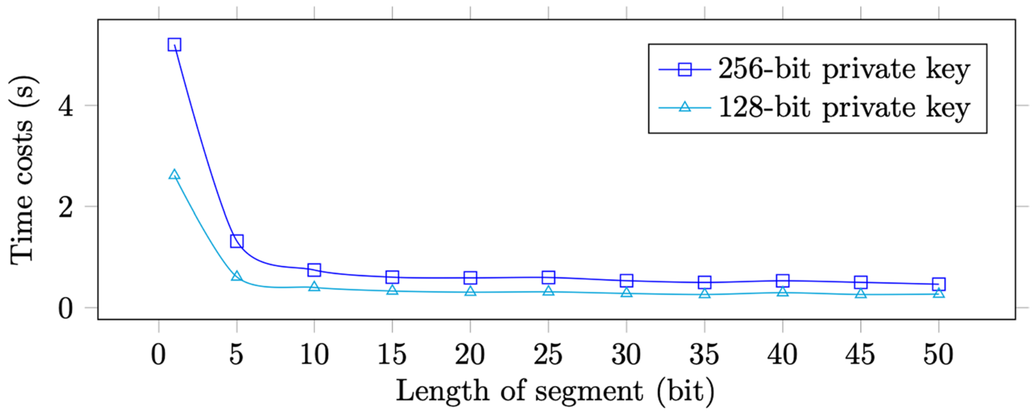

Fig. 7 Time costs of Exchange phase

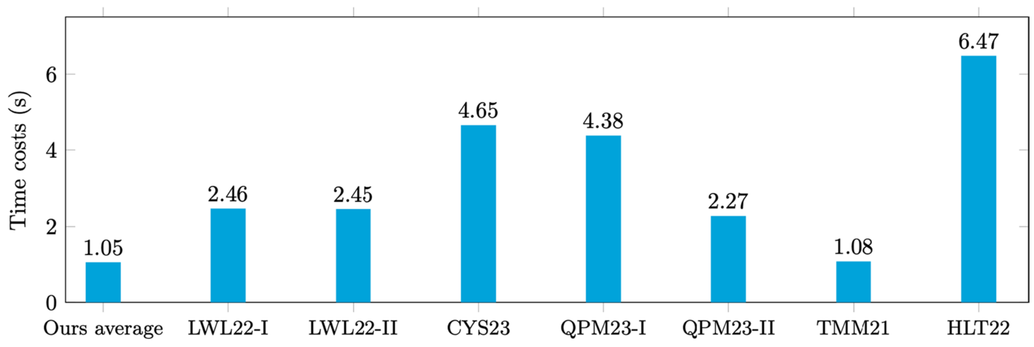

Fig. 8 Time costs of Exchange phase for different schemes

Table 2 Time costs of an ACCS for different schemes

Schemes	Ours average	QPM23-I	QPM23-II	ZHL19	HLT22
Time costs (s)	16.85	17.24	9.15	358.8	12.37

efficiency of ACCS according to (Li et al. 2022). Because compared with the on-chain transaction confirmation time[2], the off-chain time cost of our scheme is very low. Therefore, its impact on the existing system is very small.

Comparison II. In the case of a 256-bit length private key, we compare the off-chain time costs of our scheme with other schemes in executing an ACCS. Such as, Qin's Blindhub scheme and its optimized scheme (Qin et al. 2023) (denoted as QPM23-I and QPM23-II), Zamyatin's XCLAIM scheme (Zamyatin et al. 2019) (denoted as ZHL19), and Hanzlik's Sweep-UC scheme (Hanzlik et al. 2022) (denoted as HLT22). Our experimental results are shown in Table 2. The results show that the off-chain time cost of our scheme is as practical as other schemes.

(3) Experimental conclusion

In summary, although ACCS functionality is easy to be implemented using smart contracts such as HTLC, our protocol is preferable due to its advantages, such as lower on-chain overhead, and practical off-chain time costs similar to other schemes.

Conclusion

In this paper, we propose a generic ACCS scheme, independent from the underlying blockchains. To the best of our knowledge, this is the first solution of this kind. Our results are as follows. First, we define a formal system model of ACCS. Next, we present a generic ACCS scheme meets our model. This scheme admits atomicity in cross-chain swaps without the need for a TTP and protects users' privacy. Finally, by using the NIZK protocol as a tool, we instantiate our generic scheme for ECDLP-based signatures. In addition, we implement our scheme, and the experimental results show that our protocol outperforms the existing ACCS schemes, such as the HTLC-based schemes.

[2] Taking Ethereum (Buterin 2013), Bitcoin (Nakamoto 2008), and Monero (Noether 2014) as examples, their transaction confirmation times are 180 s, 3600 s, and 3840 s respectively.

However, although our research provides some insights into the field of ACCS, we must also acknowledge the limitations and unresolved issues of this research. Therefore, we propose some possible future work here. On the one hand, we will further improve the efficiency of our scheme in the future. The current scheme can meet the requirements of generality, privacy, and atomicity without TTP at the same time. However, there is still room for improvement in efficiency of our scheme. In the future, we will explore more effective cross-chain protocols to improve the real-time nature of transactions. On the other hand, we will expand the cross-chain functionality of the scheme in the future. Our scheme is mainly applicable to ACCS, however, the transfer of assets and data between different chains is also a common application scenario. Here, we take ACCS as a starting point, and in our future work, we will further study solutions that can achieve other cross-chain functions to meet the constantly evolving market demand.

In a word, we believe these future works will contribute to the development of the blockchain and cryptocurrency fields. We encourage more researchers to participate in this field to unlock its potential value and promote its further development.

Appendix A Exchange protocol

The $\mathsf{Exchange}$ protocol consists of 6 algorithms.

- $\mathsf{sk}_1[\cdot] \leftarrow \mathsf{DivideSegment}(\mathsf{sk}_1, l, \kappa)$. Given a private key share sk_1, the algorithm divides it into κ segments, denoted as $\mathsf{sk}_1[\cdot] = (\mathsf{sk}_1[1], ..., \mathsf{sk}_1[\kappa])$. Each segment's length is l bits (with leading zeros added if the length is less than l).
- $(\vec{r}, \vec{\mathsf{com}}, (\pi_{rpi})_{i\in[\kappa]}, \pi_{cc}, \pi_{\mathsf{pk}}) \leftarrow \mathsf{ComProofGen}(\mathsf{sk}_1[\cdot])$. Given κ-Segmentation of sk_1, the algorithm outputs a commitment vector $\vec{\mathsf{com}}$ for each segment in $\mathsf{sk}_1[\cdot]$, its corresponding range proofs $(\pi_{rpi})_{i\in[\kappa]}$, a correct commitment proof π_{cc}, and a Zero-Knowledge Proof of Knowledge (ZKPoK) for the private key share sk_1, denoted as π_{pk}. Specifically, the algorithm does the following:

  1. For each $i \in [\kappa]$, runs $(\mathsf{com_i}, (\mathsf{sk}_1[i], r_i)) \leftarrow \Pi_{\mathsf{Com}}.\mathsf{Com}(h, \mathsf{sk}_1[i])$ to generate commitment $\mathsf{com_i}$ for the i-th segment of the private key share. Next, it runs $\pi_{rpi} \leftarrow \Pi_{\mathsf{RP}}.\mathsf{Proof}(\mathsf{com_i})$ to generate a range proof π_{rpi} for the i-th commitment. This range proof is used to verify that the commitment $\mathsf{com_i}$ encrypts a segment with a binary length of l (if the length is less than l, leading zeros are filled).
  2. Let $\vec{\mathsf{com}} = (\mathsf{com_i})_{i\in[\kappa]}$ and $\vec{r} = (r)_{i\in[\kappa]}$. Next, runs $\pi_{cc} \leftarrow \Pi_{\mathsf{NIZK}}.\mathsf{Proof}(\vec{\mathsf{com}})$ to generate a proof of correct commitment. This proof is used to verify that each commitment item $\mathsf{com_i}$ at position i indeed encrypts the i-th item in the κ -Segmentation of sk_1.
  3. Runs $\pi_{\mathsf{pk}} \leftarrow \Pi_{\mathsf{NIZK}}.\mathsf{Proof}(\mathsf{sk}_1)$ to generate a ZKPoK for sk_1, where sk_1 is the correct private key share corresponding to the public key.
  4. Takes $(\vec{r}, \vec{\mathsf{com}}, (\pi_{rpi})_{i\in[\kappa]}, \pi_{cc}, \pi_{\mathsf{pk}})$ as the output.

- $0/1 \leftarrow \mathsf{ComProofCheck}((\pi_{rpi})_{i\in[\kappa]}, \pi_{cc}, \pi_{\mathsf{pk}})$. Given range proofs $(\pi_{rpi})_{i\in[\kappa]}$, a correct commitment proof π_{cc}, and a ZKPoK π_{pk}, the algorithm outputs 0 to indicate validation failure and outputs 1 to indicate validation success. Specifically, the algorithm does the following:

  1. For each $i \in [\kappa]$, runs $0/1 \leftarrow \Pi_{\mathsf{RP}}.\mathsf{Verify}(\pi_{rpi})$, if it outputs 0, aborts and returns 0 as output.
  2. Runs $0/1 \leftarrow \Pi_{\mathsf{NIZK}}.\mathsf{Verify}(\pi_{cc})$, if it outputs 0, aborts and returns 0 as output.
  3. Runs $0/1 \leftarrow \Pi_{\mathsf{NIZK}}.\mathsf{Verify}(\pi_{\mathsf{pk}})$, if it outputs 0, aborts and returns 0 as output.
  4. Returns 1 as output.

- $(\mathsf{sk}_1[i], r_i) \leftarrow \mathsf{SegComOpen}(\mathsf{sk}_1, \vec{r}, i)$. Given a private key share sk_1, a random value vector $\vec{r}$, and the index i of the commitment to be opened, the algorithm outputs the opening value $(\mathsf{sk}_1[i], r_i)$ of the commitment.
- $0/1 \leftarrow \mathsf{SegComCheck}(h, \mathsf{sk}_1[i], \mathsf{com}, (\mathsf{sk}_1[i], r_i))$. Given a public value h of the commitment, the i-th segment of a private key share, a commitment com, and a corresponding opening value $(\mathsf{sk}_1[i], r_i)$, the algorithm runs $0/1 \leftarrow \Pi_{\mathsf{Com}}.\mathsf{Vf}(h, \mathsf{sk}_1[i], \mathsf{com}, (\mathsf{sk}_1[i], r_i))$ and outputs the result.
- $\mathsf{sk}_1 \leftarrow \mathsf{ConnectSegment}(\mathsf{sk}_1[\cdot])$. Given sk_1's κ-Segmentation, the algorithm outputs sk_1.

Appendix B One-to-one ACCS protocol

Our ACCS protocol can be divided into five phases: setup phase, freezing phase, exchange phase, complete phase and timeout phase. The complete one-to-one ACCS protocol is shown in Fig. 9 and 10.

Appendix C Proof of Theorem 1

Completeness. To prove the correctness of the protocol in Appendix B, we discuss two scenarios: timeout and no timeout.

Setup Phase.

1. Runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ to obtain the following parameters: the public value h of the commitment scheme, user A's asset v_1, user B's asset v_2, the constants const_1 and const_2, the bit length l of private key share segments, the number κ of private key share segments, the timed parameters T_1, T_2, and a delay parameter $\triangle$ used to prevent race conditions and ensure the fairness of the transaction.
2. Runs KeyGen to generate the following public-private key pairs:
   - User A's initial address: $(\mathsf{pk}^{(\mathsf{A})}, \mathsf{sk}^{(\mathsf{A})})$ (with v_1 as the initial asset).
   - User B's initial address: $(\mathsf{pk}^{(\mathsf{B})}, \mathsf{sk}^{(\mathsf{B})})$ (with v_2 as the initial asset).
   - User A's swap address: $(\mathsf{pk}^{(\mathsf{A})}_{\mathsf{swap}}, \mathsf{sk}^{(\mathsf{A})}_{\mathsf{swap}})$.
   - User B's swap address: $(\mathsf{pk}^{(\mathsf{B})}_{\mathsf{swap}}, \mathsf{sk}^{(\mathsf{B})}_{\mathsf{swap}})$.
   - User A's refund address: $(\mathsf{pk}^{(\mathsf{A})}_{\mathsf{rfnd}}, \mathsf{sk}^{(\mathsf{A})}_{\mathsf{rfnd}})$.
   - User B's refund address: $(\mathsf{pk}^{(\mathsf{B})}_{\mathsf{rfnd}}, \mathsf{sk}^{(\mathsf{B})}_{\mathsf{rfnd}})$.
3. Runs $(\mathsf{pk}^{(\mathsf{AB})}, \mathsf{sk}^{(\mathsf{AB})}_{\mathsf{A}}, \mathsf{sk}^{(\mathsf{AB})}_{\mathsf{B}}) \leftarrow \mathsf{JointKeyGen}(1^\lambda)$ to generate the joint address $\mathsf{pk}^{(\mathsf{AB})}$ for freezing A's asset. Here, $\mathsf{sk}^{(\mathsf{AB})}_{\mathsf{A}}$ is the private key of user A, and $\mathsf{sk}^{(\mathsf{AB})}_{\mathsf{B}}$ is the private key of user B.
4. Runs $(\mathsf{pk}^{(\mathsf{BA})}, \mathsf{sk}^{(\mathsf{BA})}_{\mathsf{A}}, \mathsf{sk}^{(\mathsf{BA})}_{\mathsf{B}}) \leftarrow \mathsf{JointKeyGen}(1^\lambda)$ to generate the joint address $\mathsf{pk}^{(\mathsf{BA})}$ for freezing B's asset. Here, $\mathsf{sk}^{(\mathsf{BA})}_{\mathsf{A}}$ is the private key of user A, and $\mathsf{sk}^{(\mathsf{BA})}_{\mathsf{B}}$ is the private key of user B.
5. Runs GenTx to generate the following transactions:
   - User A's freeze transaction: $\mathsf{tx}^{(\mathsf{A})}_{\mathsf{frz}} = \mathsf{tx}(\mathsf{pk}^{(\mathsf{A})}, \mathsf{pk}^{(\mathsf{AB})}, v_1)$.
   - User B's freeze transaction: $\mathsf{tx}^{(\mathsf{B})}_{\mathsf{frz}} = \mathsf{tx}(\mathsf{pk}^{(\mathsf{B})}, \mathsf{pk}^{(\mathsf{BA})}, v_2)$.
   - User A's swap transaction: $\mathsf{tx}^{(\mathsf{A})}_{\mathsf{swap}} = \mathsf{tx}(\mathsf{pk}^{(\mathsf{BA})}, \mathsf{pk}^{(\mathsf{A})}_{\mathsf{swap}}, v_2)$.
   - User B's swap transaction: $\mathsf{tx}^{(\mathsf{B})}_{\mathsf{swap}} = \mathsf{tx}(\mathsf{pk}^{(\mathsf{AB})}, \mathsf{pk}^{(\mathsf{B})}_{\mathsf{swap}}, v_1)$.
   - User A's refund transaction: $\mathsf{tx}^{(\mathsf{A})}_{\mathsf{rfnd}} = \mathsf{tx}(\mathsf{pk}^{(\mathsf{AB})}, \mathsf{pk}^{(\mathsf{A})}_{\mathsf{rfnd}}, v_1)$.
   - User B's refund transaction: $\mathsf{tx}^{(\mathsf{B})}_{\mathsf{rfnd}} = \mathsf{tx}(\mathsf{pk}^{(\mathsf{BA})}, \mathsf{pk}^{(\mathsf{B})}_{\mathsf{rfnd}}, v_2)$.

Fig. 9 One-to-one ACCS protocol - part I

Freeze Phase.

1. User A plays the role of P_1 in the $\mathsf{CommitRfnd}$ protocol, and user B plays the role of P_2, specifically:
   - The input of A is $(\mathsf{T}_1, \mathsf{sk}_{\mathsf{A}}^{(\mathsf{AB})}, \mathsf{pk}^{(\mathsf{AB})}, \mathsf{tx}_{\mathsf{rfnd}}^{(\mathsf{A})})$, and the input of B is $(\mathsf{T}_1, \mathsf{sk}_{\mathsf{B}}^{(\mathsf{AB})}, \mathsf{pk}^{(\mathsf{AB})}, \mathsf{tx}_{\mathsf{rfnd}}^{(\mathsf{A})})$.
   - The output of A is $(\mathsf{com}^{(\mathsf{A})}, \pi^{(A)})$, and the output of B is $(\sigma_{rfnd}^{(A)})$.
2. User A runs $0/1 \leftarrow \mathsf{Verify}(\mathsf{pk}^{(\mathsf{AB})}, \mathsf{tx}_{\mathsf{rfnd}}^{(\mathsf{A})}, \mathsf{com}^{(\mathsf{A})}, \pi^{(A)})$, if outputs 0, aborts.
3. User A runs $\sigma_{frz}^{(A)} \leftarrow \mathsf{Freeze}(\mathsf{sk}^{(\mathsf{A})}, \mathsf{tx}_{\mathsf{frz}}^{(\mathsf{A})})$ to generate the signature of the freeze transaction, and then publishes it to $\mathbb{B}_1$ to freeze his asset v_1.
4. User A starts running $\mathsf{UnFreeze}(\mathsf{com}^{(\mathsf{A})})$.
5. Similarly, user B plays the role of P_1 in the $\mathsf{CommitRfnd}$ protocol, and user A plays the role of P_2, specifically:
   - The input of B is $(\mathsf{T}_2, \mathsf{sk}_{\mathsf{B}}^{(\mathsf{BA})}, \mathsf{pk}^{(\mathsf{BA})}, \mathsf{tx}_{\mathsf{rfnd}}^{(\mathsf{B})})$, and the input of A is $(\mathsf{T}_2, \mathsf{sk}_{\mathsf{A}}^{(\mathsf{BA})}, \mathsf{pk}^{(\mathsf{BA})}, \mathsf{tx}_{\mathsf{rfnd}}^{(\mathsf{B})})$.
   - The output of B is $(\mathsf{com}^{(\mathsf{B})}, \pi^{(B)})$, and the output of A is $(\sigma_{rfnd}^{(B)})$.
6. User B runs $0/1 \leftarrow \mathsf{Verify}(\mathsf{pk}^{(\mathsf{BA})}, \mathsf{tx}_{\mathsf{rfnd}}^{(\mathsf{B})}, \mathsf{com}^{(\mathsf{B})}, \pi^{(B)})$, if outputs 0, aborts.
7. User B runs $\sigma_{frz}^{(B)} \leftarrow \mathsf{Freeze}(\mathsf{sk}^{(\mathsf{B})}, \mathsf{tx}_{\mathsf{frz}}^{(\mathsf{B})})$ to generate the signature of the freeze transaction, and then publishes it to $\mathbb{B}_2$ to freeze his asset v_2.
8. User B starts running $\mathsf{UnFreeze}(\mathsf{com}^{(\mathsf{B})})$.

Exchange Phase.

1. User A plays the role of P_1 in the $\mathsf{Exchange}$ protocol, and user B plays the role of P_2, specifically:
   - The input of user A is $(\mathsf{sk}_{\mathsf{A}}^{(\mathsf{AB})})$, and the input of user B is $(\mathsf{sk}_{\mathsf{B}}^{(\mathsf{BA})})$.
   - The output of user A is $(\mathsf{sk}_{\mathsf{B}}^{(\mathsf{BA})})$, and the output of user B is $(\mathsf{sk}_{\mathsf{A}}^{(\mathsf{AB})})$.

Complete Phase.

1. User A runs $\sigma_{swap}^{(A)} \leftarrow \mathsf{Complete}(\mathsf{sk}_{\mathsf{A}}^{(\mathsf{BA})}, \mathsf{sk}_{\mathsf{B}}^{(\mathsf{BA})}, \mathsf{tx}_{\mathsf{swap}}^{(\mathsf{A})})$ to generate the signature of the swap transaction, and then publishes it on the distributed ledger $\mathbb{B}_2$ to obtain user B's asset v_2.
2. Similarly, user B runs $\sigma_{swap}^{(B)} \leftarrow \mathsf{Complete}(\mathsf{sk}_{\mathsf{A}}^{(\mathsf{AB})}, \mathsf{sk}_{\mathsf{B}}^{(\mathsf{AB})}, \mathsf{tx}_{\mathsf{swap}}^{(\mathsf{B})})$ to generate the signature of the swap transaction, and then publishes it on the distributed ledger $\mathbb{B}_1$ to obtain user A's asset v_1.

Timeout Phase.

1. In the case of timeout, user B completes $\sigma_{rfnd}^{(B)} \leftarrow \mathsf{UnFreeze}(\mathsf{com}^{(\mathsf{B})})$ after T_2 to get the signature of the refund transaction, and then publishes it on $\mathbb{B}_2$ to refund his asset v_2.
2. Similarly, user A completes $\sigma_{rfnd}^{(A)} \leftarrow \mathsf{UnFreeze}(\mathsf{com}^{(\mathsf{A})})$ after T_1 to get the signature of the refund transaction, and then publishes it on $\mathbb{B}_1$ to refund his asset v_1.

Fig. 10 One-to-one ACCS protocol - part II

1. **No timeout.** Assuming that both users participating in ACCS follow the protocol, in the case of no timeout, they freeze their assets to the joint addresses and successfully exchange their private key shares. Therefore, they can sign the swap transactions and publish the signatures to swap their assets. This results in both users' states being 1 at the end of the protocol, i.e., $\mathsf{state}_{\mathsf{A}} = 1 \wedge \mathsf{state}_{\mathsf{B}} = 1$.
2. **Timeout.** Assuming that both users participating in ACCS follow the protocol, the timeout is usually caused by one user being offline, resulting in the failure of the $\mathsf{Exchange}$ protocol. In the case of timeout, users will be unable to reconstruct the complete private key corresponding to the joint address. Consequently, they will not execute the complete phase but execute the unfreeze phase to obtain the signature of the refund transaction, and then publish it for refund. This results in both users' states being 0 at the end of the protocol, i.e., $\mathsf{state}_{\mathsf{A}} = 0 \wedge \mathsf{state}_{\mathsf{B}} = 0$.

Soundness. We use proof by contradiction to prove that the protocol in Appendix B satisfies the soundness property. Assuming that an adversary $\mathcal{A}$ generates a commitment com^* and its corresponding proof π^*, such that the output of the Verify algorithm is 1, but the output of the $\Pi_{\mathsf{DS}}.\mathsf{Vf}$ algorithm is 0. This would imply that the adversary has broken the soundness property of the underlying VTS module Π_{VTS} (Thyagarajan et al. 2020; Thyagarajan 2022).

Timed Privacy. We use proof by contradiction to prove that the protocol in Appendix B satisfies the timed privacy property. Assuming a PPT adversary $\mathcal{A}$ with running time $t \ll \mathsf{T} - \mathsf{const}_1$ generates a valid signature σ^*, resulting in the output of the $\Pi_{\mathsf{DS}}.\mathsf{Vf}$ algorithm being 1. This implies that the adversary has broken the privacy property of the underlying VTS module Π_{VTS} (Thyagarajan et al. 2020; Thyagarajan 2022). Similarly, if an honest user A with running time $t \gg \mathsf{T} + \mathsf{const}_1$ executes the $\mathsf{UnFreeze}$ algorithm and obtains an invalid signature σ, resulting in the output of the $\Pi_{\mathsf{DS}}.\mathsf{Vf}$ algorithm being 0. It also indicates that the adversary has broken the privacy property of the underlying VTS module Π_{VTS} (Thyagarajan et al. 2020; Thyagarajan 2022).

Partial Fairness. The partial fairness property of the protocol ensures that the following situation is impossible: $|\Pr[\mathsf{sk}_\mathsf{A}^{(AB)} = \mathsf{sk}_\mathsf{A}^{'}] - \Pr[\mathsf{sk}_\mathsf{B}^{(BA)} = \mathsf{sk}_\mathsf{B}^{'}]|$ is greater than $\frac{2^{\mathsf{const}_2}}{2^{\lambda-i}} + \mathsf{negl}(\lambda)$, where $\lambda = |\mathsf{sk}_\mathsf{A}^{(AB)}| = |\mathsf{sk}_\mathsf{B}^{(BA)}|$ is the security parameter representing the length of the private key share. Here, i represents the prefix bit length of the private key share $\mathsf{sk}_\mathsf{B}^{(BA)}$ obtained by user A when the $\mathsf{Exchange}$ protocol aborts, and $\mathsf{const}_2 = l$ represents the segment bit length specified by the system in the setup phase.

Let $\mathcal{A}$ be an adversary who breaks the partial fairness property of our protocol. To prove that the protocol in Appendix B satisfies the partial fairness property, we distinguish between two types of adversaries. Type I represents an adversary $\mathcal{A}$ that cannot lie. This type of adversary follows the protocol but may abort prematurely. Type II represents an adversary $\mathcal{A}$ that can lie, meaning they may forge range proofs, NIZK proofs, and/or segment commitments.

1. **Type I - User A is corrupted.** In this case, the adversary $\mathcal{A}$ may choose to abort the $\mathsf{Exchange}$ protocol either before or after user B sends the opening value of a segment commitment during the protocol execution. First, we assume that the adversary $\mathcal{A}$ aborts the $\mathsf{Exchange}$ protocol before user B sends the opening value of a segment commitment during the protocol execution. At this point, the adversary $\mathcal{A}$ obtains the first i bits of $\mathsf{sk}_\mathsf{B}^{(BA)}$, and user B obtains the first $i+l$ bits of $\mathsf{sk}_\mathsf{A}^{(AB)}$. Thus, we have $\Pr[\mathsf{sk}_\mathsf{A}^{(AB)} = \mathsf{sk}_\mathsf{A}^{'}] = \frac{1}{2^{\lambda-i-l}}$ and $\Pr[\mathsf{sk}_\mathsf{B}^{(BA)} = \mathsf{sk}_\mathsf{B}^{'}] = \frac{1}{2^{\lambda-i}}$, which leads to $|\Pr[\mathsf{sk}_\mathsf{A}^{(AB)} = \mathsf{sk}_\mathsf{A}^{(AB)'}] - \Pr[\mathsf{sk}_\mathsf{B}^{(BA)} = \mathsf{sk}_\mathsf{B}^{'}]| = \frac{2^l-1}{2^{\lambda-i}} \leq \frac{2^l}{2^{\lambda-i}} + \mathsf{negl}(\lambda)$. Therefore, in this case, our protocol satisfies the property of partial fairness. Next, we assume that the adversary $\mathcal{A}$ aborts the $\mathsf{Exchange}$ protocol after user B sends the opening value of a segment commitment during the protocol execution. At this point, the adversary $\mathcal{A}$ obtains the first i bits of $\mathsf{sk}_\mathsf{B}^{(BA)}$, and user B also obtains the first i bits of $\mathsf{sk}_\mathsf{A}^{(AB)}$. Thus, we have $\Pr[\mathsf{sk}_\mathsf{A}^{(AB)} = \mathsf{sk}_\mathsf{A}^{'}] = \Pr[\mathsf{sk}_\mathsf{B}^{(BA)} = \mathsf{sk}_\mathsf{B}^{'}] = \frac{1}{2^{\lambda-i}}$, which results in $|\Pr[\mathsf{sk}_\mathsf{A}^{(AB)} = \mathsf{sk}_\mathsf{A}^{'}] - \Pr[\mathsf{sk}_\mathsf{B}^{(BA)} = \mathsf{sk}_\mathsf{B}^{(BA)'}]| = 0 \leq \frac{2^l}{2^{\lambda-i}} + \mathsf{negl}(\lambda)$. Therefore, in this case, our protocol satisfies the property of partial fairness.
2. **Type I - User B is corrupted.** In this case, the adversary $\mathcal{A}$ may choose to abort the $\mathsf{Exchange}$ protocol either before or after user A sends the opening value of a segment commitment during the protocol execution. First, we assume that the adversary $\mathcal{A}$ aborts the $\mathsf{Exchange}$ protocol before user A sends the opening value of a segment commitment during the protocol execution. At this point, the user A obtains the first i bits of $\mathsf{sk}_\mathsf{B}^{(BA)}$, and adversary $\mathcal{A}$ also obtains the first i bits of $\mathsf{sk}_\mathsf{A}^{(AB)}$. Thus, we have $\Pr[\mathsf{sk}_\mathsf{A}^{(AB)} = \mathsf{sk}_\mathsf{A}^{'}] = \Pr[\mathsf{sk}_\mathsf{B}^{(BA)} = \mathsf{sk}_\mathsf{B}^{'}] = \frac{1}{2^{\lambda-i}}$, which leads to $|\Pr[\mathsf{sk}_\mathsf{A}^{(AB)} = \mathsf{sk}_\mathsf{A}^{'}] - \Pr[\mathsf{sk}_\mathsf{B}^{(BA)} = \mathsf{sk}_\mathsf{B}^{(BA)'}]| = 0 \leq \frac{2^l}{2^{\lambda-i}} + \mathsf{negl}(\lambda)$. Therefore, in this case, our protocol satisfies the property of partial fairness. Next, we assume that the adversary $\mathcal{A}$ aborts the $\mathsf{Exchange}$ protocol after user A sends the opening value of a segment commitment during the protocol execution. At this point, the user A obtains the first i bits of $\mathsf{sk}_\mathsf{B}^{(BA)}$, and adversary $\mathcal{A}$ obtains the first $i+l$ bits of $\mathsf{sk}_\mathsf{A}^{(AB)}$. Thus, we have $\Pr[\mathsf{sk}_\mathsf{A}^{(AB)} = \mathsf{sk}_\mathsf{A}^{'}] = \frac{1}{2^{\lambda-i-l}}$ and $\Pr[\mathsf{sk}_\mathsf{B}^{(BA)} = \mathsf{sk}_\mathsf{B}^{'}] = \frac{1}{2^{\lambda-i}}$, which results in $|\Pr[\mathsf{sk}_\mathsf{A}^{(AB)} = \mathsf{sk}_\mathsf{A}^{(AB)'}] - \Pr[\mathsf{sk}_\mathsf{B}^{(BA)} = \mathsf{sk}_\mathsf{B}^{'}]| = \frac{2^l-1}{2^{\lambda-i}} \leq \frac{2^l}{2^{\lambda-i}} + \mathsf{negl}(\lambda)$. Therefore, in this case, our protocol satisfies the property of partial fairness.
3. **Type II.** For Type II, We use proof by contradiction to prove that the protocol in Appendix B satisfies the partial fairness property. Assuming that an adversary $\mathcal{A}$ forges a range proof π_1^*, a NIZK proof π_2^* and/or a segment commitment com^* that

can pass the ComProofCheck and SegComCheck algorithms. This implies that the adversary has broken the soundness property of the underlying zero-knowledge proof module Π_{RP} and/or Π_{NIZK}, and/or the binding property of the underlying commitment scheme module Π_{Com}.

Appendix D Proof of Theorem 2

Before proceeding with formal proof, we first emphasize the role of range proofs, which use Bulletproofs for efficient instantiation. To intuitively understand the reasons behind this, first we describe a possible attack. Subsequently, we will explain why Bulletproofs range proofs can effectively resist such attack.

The zero-knowledge proof protocol of correct commitment proves that C_i encrypts $\theta[i]$, such that $P(2^l) = \sum_{i\in[\kappa]} \theta[i]2^{l(i-1)} = \theta$, where θ is the DL of D. However, this alone does not guarantee the correctness of the protocol, as there exists a simple attack that involves generating commitments for shifted segments, such that C_i encrypts plaintext that shifts $\theta[i]$. In this attack, the final sum of plaintext remains the same, i.e., θ, but each C_i will be opened incorrectly. Specifically, the meaning of this attack is that if the malicious user A aborts during the gradual release of secret segments, then the honest user B will have nothing, while A will have all the correct secret segments that have already been released by B.

We use an example to illustrate the harm of this attack. Assuming that the adversary only changes two secret segments, namely the i-th segment and the $i+1$ -th segment. The modified values are $\theta[i]' = \theta[i] + b[i]$ and $\theta[i+1]' = \theta[i+1] + b[i+1]$. After modification, the commitment value for the i-th segment becomes $C_i' = r_iG + \theta[i]'G_i = r_iG + (\theta[i] + b[i])G_i$, and the commitment value for the $i+1$-th segment becomes $C_{i+1}' = r_{i+1}G + \theta[i+1]'G_{i+1} = r_{i+1}G + (\theta[i+1] + b[i+1])G_{i+1}$. During the gradual release of secret segments, the adversary sends $\theta[i]'$ and $\theta[i+1]'$ to the verifier, claiming that they are $\theta[i]$ and $\theta[i+1]$. The verifier can only check whether the commitment has been opened correctly, but cannot determine whether the opened plaintext is the correct secret segment $\theta[i]$, or a shift of it, i.e., $\theta[i]' = \theta[i] + b[i]$. As a result, the commitments of both secret segments are opened to incorrect values. Specifically, the i-th segment is opened to $\theta[i] + b[i] \neq \theta[i]$, and the $i+1$-th segment is opened to $\theta[i+1] + b[i+1] \neq \theta[i+1]$.

However, this attack can be avoided with a high probability by generating Bulletproofs range proof for commitment C_i. Next, we show how Bulletproofs range proofs prevent this specific attack, and extend it to all possible changes to the encrypted segments $\theta[\cdot]$. There are four cases of this attack:

1. $\theta[i]' \in [0, ..., 2^l - 1] \wedge \theta[i+1]' \in [0, ..., 2^l - 1]$.
2. $\theta[i]' \notin [0, ..., 2^l - 1] \wedge \theta[i+1]' \in [0, ..., 2^l - 1]$.
3. $\theta[i]' \in [0, ..., 2^l - 1] \wedge \theta[i+1]' \notin [0, ..., 2^l - 1]$.
4. $\theta[i]' \notin [0, ..., 2^l - 1] \wedge \theta[i+1]' \notin [0, ..., 2^l - 1]$.

For the latter three cases, there is a high probability that at least one Bulletproofs range proof cannot be verified. In the first case, in order to pass the verification of the zero-knowledge proof protocol of correct commitment, it is necessary to satisfy $\sum_{i\in[\kappa]} b[i]2^{l(i-1)} + \sum_{i\in[\kappa]} b[i+1]2^{l(i-1)} = np$, where n$p$ is a multiple of the order. Every option except $n = 0$ and $b[i] = b[i+1] = 0$ will cause $b[i]$ and $b[i+1]$ to be out of the interval $[0, ..., 2^l - 1]$ with a very high probability.

Perfect Completeness. Since the prover knows the common reference string CRS and the segment vector $\theta[\cdot]$, he can compute $U = \frac{1}{s}(\sum_{i\in[\kappa]} C_i - rG)$ without knowing s. Here, $U = \sum_{i\in[\kappa]} \theta[i]G_{i-1}$. Similarly, since the prover knows the common reference string CRS and the coefficients of the formal polynomial $W(\cdot)$, he can compute V. Here, $V = \sum_{i\in[\kappa-1]} W[i]G_{i-1}$.

Computational Soundness. We assume that the one who breaks the soundness of the protocol is the PPT adversary $\mathcal{A}$, and $\mathcal{B}$ is an adversary we built. $\mathcal{B}$ receives $(G_0, G_1, G_2, ..., G_\kappa)$ and uses this tuple as the CRS, then sends it to $\mathcal{A}$. $\mathcal{A}$ returns the following values:

- $\theta \in \mathbb{Z}_p^*$.
- $\theta^* \in \mathbb{Z}_p^*$ such that $D = \theta^*G$ and $\theta \neq \theta^*$.
- For each $i \in [\kappa]$ there exist $(r_i, \theta[i]) \in (\mathbb{Z}_p^* \times \mathbb{Z}_p^*)^\kappa$ such that $\vec{C} = (C_i)_{i\in\kappa}$, where $C_i = r_iG + \theta[i]G_i$.
- $\pi = (r, U, V) \in \mathbb{Z}_p^* \times \mathbb{G} \times \mathbb{G}$.

The proof is divided into two steps. First we assume that $r \neq \sum_{i\in[\kappa]} r_i \bmod p$. In this case, we can deduce that $U = \frac{1}{s}((\sum_{i\in[\kappa]} r_i - r)G + \sum_{i\in[\kappa]} \theta[i]s^iG)$. Since $\theta[i]$ is known, $\mathcal{B}$ can compute $\frac{1}{s}\theta[i]s^iG = \theta[i]s^{i-1}G = \theta[i]G_{i-1}$. Consequently, $\mathcal{B}$ can deduce $\frac{\sum_{i\in[\kappa]} r_i - r}{s}G$. Given that $\delta = \sum_{i\in[\kappa]} r_i - r \neq 0 \bmod p$ is known, $\frac{1}{s}G = \frac{1}{\delta}(U - \theta[i]G_{i-1})$ can be easily computed, thus breaking the κ-DHI assumption.

Next, we assume that $\sum_{i\in[\kappa]} r_i = r$. When the verification of V passes, it implies that we have $V = \frac{\sum_{i\in[\kappa]}(\theta[i]s^{i-1} - \theta^*[i]2^{l(i-1)})}{s - 2^l}G$. If the adversary $\mathcal{A}$ wins, it means that there exists some $j \in [\kappa]$ such that $\theta[j] \neq \theta^*[j]$. Further analysis reveals that Bulletproofs guarantee the length of $\theta[i]$ and $\theta^*[i]$ to be l, making it impossible to encrypt shifted segment. Thus, we have $\Delta = \sum_{i\in[\kappa]} 2^{l(i-1)}(\theta[i] - \theta^*[i]) \neq 0 \, mod \, p$. We can rephrase V as $V = (\frac{\Delta}{s-2^l} + \frac{\sum_{i\in[\kappa]} \theta[i]s^{i-1} - \theta[i]2^{l(i-1)}}{s-2^l})G$ $= (\frac{\Delta}{s-2^l} + \frac{\sum_{i\in[\kappa]} \theta[i](s^{i-1} - 2^{l(i-1)})}{s-2^l})G = (\frac{\Delta}{s-2^l} + Z(s))G$. Because $\forall i \in [\kappa] : s - 2^l | s^{i-1} - 2^{l(i-1)}$ (where s is required to contain a factor 2^l), $\mathcal{B}$ can efficiently compute the coefficients of the polynomial $Z(\cdot)$. Furthermore, because $\Delta \in \mathbb{Z}_p^*$ is also known, this allows $\mathcal{B}$ to compute $\frac{1}{s-2^l}G = \frac{1}{\Delta}(V - Z(s)G)$, thus breaking the κ-SDH assumption.

Perfect Zero-Knowledge. The simulator correctly generates the common reference string CRS and saves the trapdoor s. For the given statements D and $\vec{C} = (C_i)_{i\in[\kappa]}$, the simulator chooses a random number $r' \in \mathbb{Z}_p^*$, and reveal it as the randomness of $\sum_{i\in[\kappa]} C_i$. Then, the simulator computes $U' = \frac{1}{s}(\sum_{i\in[\kappa]} C_i - r'G)$ and $V' = \frac{1}{s-2}(U' - D)$. The following shows that the r', U', V' generated by the simulator are indistinguishable from the r, U, V in the real experiment:

- r' is uniformly distributed as well as r.
- Let the function $f_1 : \mathbb{G} \to \mathbb{G}$ be defined as $f_1(x) = \sum_{i\in[\kappa]} C_i - xG$. In the simulated experiment, $D' = f_1(r')$, and in the real experiment, $D' = f_1(r)$.
- Let the function $f_2 : \mathbb{G} \to \mathbb{G}$ be defined as $f_2(x) = \frac{1}{s}f_1(x)$. In the simulated experiment, $U' = f_2(r')$, and in the real experiment, $U = f_2(r)$.
- Let the function $f_3 : \mathbb{G} \to \mathbb{G}$ be defined as $f_3(x) = \frac{1}{s-2}(f_2(x) - D)$. In the simulated experiment, $V' = f_3(r')$, and in the real experiment, $V = f_3(r)$.

Acknowledgements
The authors would like to thank the anonymous reviewers for their valuable comments.

Authors' contributions
The author(s) read and approved the final manuscript.

Funding
This work was supported by National Natural Science Foundation of China (Nos. 62172404, 62172411, 61972094, 62202458).

Availability of data and materials
Not applicable.

Declarartions

Competing interests
The authors declare that they have no competing interests.

Received: 15 August 2023 Accepted: 14 December 2023
Published online: 02 June 2024

References

Agbo CC, Mahmoud QH, Eklund JM (2019) Blockchain technology in healthcare: a systematic review. Healthcare 7:56
Asokan N (1998) Fairness in electronic commerce
Bentov I, Ji Y, Zhang F, Breidenbach L, Daian P, Juels A (2019) Tesseract: Real-time cryptocurrency exchange using trusted hardware. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security, pp. 1521–1538
Bhat A (2020) Linearly homomorphic time lock puzzle library. https://github.com/verifiable-timed-signatures/liblhtlp
Boneh D, Boyen X (2008) Short signatures without random oracles and the SDH assumption in bilinear groups. J Cryptol 21(2):149–177
Boneh D, Gentry C, Lynn, B, Shacham H (2003) Aggregate and verifiably encrypted signatures from bilinear maps. In: Advances in Cryptology-EUROCRYPT 2003: international conference on the theory and applications of cryptographic techniques, Warsaw, 2003 Proceedings 22, pp. 416–432. Springer
Buterin V (2013) Ethereum white paper. GitHub Repository 1:22–23
Bünz B, Bootle J, Boneh D, Poelstra A, Wuille P, Maxwell G (2018) Bulletproofs: short proofs for confidential transactions and more. In: 2018 IEEE symposium on security and privacy (SP), pp. 315–334. IEEE
Camacho P (2013) Fair exchange of short signatures without trusted third party. In: Topics in cryptology–CT-RSA 2013: the cryptographers' track at the RSA conference 2013, San Francisco. Proceedings, pp. 34–49 . Springer
Camenisch J, Stadler M (1997) Proof systems for general statements about discrete logarithms. Technical Report/ETH Zurich, Department of Computer Science 260
Chase M, Orrù M, Perrin T, Zaverucha G (2022) Proofs of discrete logarithm equality across groups. Cryptology ePrint Archive
Chen L, Yao Z, Si X, Zhang Q (2023) Three-stage cross-chain protocol based on notary group. Electronics 12(13):2804
Cleve R (1986) Limits on the security of coin flips when half the processors are faulty. In: Proceedings of the eighteenth annual ACM symposium on theory of computing, pp. 364–369
Deshpande A, Herlihy M (2020) Privacy-preserving cross-chain atomic swaps. In: International conference on financial cryptography and data security, pp. 540–549. Springer
Erwig A, Faust S, Hostáková K, Maitra M, Riahi S (2021) Two-party adaptor signatures from identification schemes. In: IACR international conference on public-key cryptography, pp. 451–480. Springer
Foundry F (2020) HTLC solidity implementation. https://github.com/functionalfoundry/ethereum-htlc
Fournier L (2019) One-time verifiably encrypted signatures aka adaptor signatures
Garay J, Kiayias A, Leonardos N (2015) The bitcoin backbone protocol: analysis and applications. In: Annual international conference on the theory and applications of cryptographic techniques, pp. 281–310. Springer
Gennaro R, Goldfeder S (2018) Fast multiparty threshold ECDSA with fast trustless setup. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, pp. 1179–1194
Glaeser N, Maffei M, Malavolta G, Moreno-Sanchez P, Tairi E, Thyagarajan SAK (2022) Foundations of coin mixing services. In: Proceedings of the 2022 ACM SIGSAC conference on computer and communications security, pp. 1259–1273
Gordon SD, Katz J (2012) Partial fairness in secure two-party computation. J Cryptol 25(1):14–40
Gugger J (2020) Bitcoin-monero cross-chain atomic swap. Cryptology ePrint Archive
Guo Y, Liang C (2016) Blockchain application and outlook in the banking industry. Financ Innov 2:1–12
Hanser C, Rabkin M, Schröder D (2015) Verifiably encrypted signatures: security revisited and a new construction. In: Computer Security–ESORICS 2015:

20th European symposium on research in computer security, Vienna, 2015, Proceedings, Part I 20, pp. 146–164 . Springer
Hanzlik L, Loss J, Thyagarajan SA, Wagner B (2022) Sweep-uc: swapping coins privately. Cryptology ePrint Archive
Hatch C (2019) Hashed timelock contract ethereum. https://github.com/chatch/hashed-timelock-contract-ethereum
Hei Y, Li D, Zhang C, Liu J, Liu Y, Wu Q (2022) Practical AgentChain: a compatible cross-chain exchange system. Futur Gener Comput Syst 130:207–218
Heilman E, Alshenibr L, Baldimtsi F, Scafuro A, Goldberg S (2017) Tumblebit: an untrusted bitcoin-compatible anonymous payment hub. In: Network and distributed system security symposium
Herlihy M (2018) Atomic cross-chain swaps. In: Proceedings of the 2018 ACM symposium on principles of distributed computing, pp. 245–254
Herlihy M, Liskov B, Shrira L (2019) Cross-chain deals and adversarial commerce. arXiv preprint arXiv:1905.09743
Hoenisch P, Mazumdar S, Moreno-Sanchez P, Ruj S (2022) Lightswap: an atomic swap does not require timeouts at both blockchains. In: International workshop on data privacy management, pp. 219–235. Springer
Hoenisch P, Pino LS (2021) Atomic swaps between bitcoin and monero. arXiv preprint arXiv:2101.12332
Hoenisch P (2020) COMIT contracts. https://github.com/comit-network/blockchain-contracts/blob/82cf33c0d01e445f2bd05bf3eb32a0143e672ab5/src/ethereum/rfc003/ether_htlc.rs
Hoenisch P (2020) COMIT contracts. https://github.com/comit-network/blockchain-contracts/blob/82cf33c0d01e445f2bd05bf3eb32a0143e672ab5/src/ethereum/rfc003/erc20_htlc.rs
Kiayias A, Russell A, David B, Oliynykov R (2017) Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Annual international cryptology conference, pp. 357–388 . Springer
Li Y, Weng J, Li M, Wu W, Weng J, Liu J-N, Hu S (2022) ZeroCross: a sidechain-based privacy-preserving cross-chain solution for Monero. J Parallel Distrib Comput 169:301–316
Lisi A, De Salve A, Mori P, Ricci L (2020) Practical application and evaluation of atomic swaps for blockchain-based recommender systems. In: Proceedings of the 2020 3rd international conference on blockchain technology and applications, pp. 67–74
Litecoin (2011) https://litecoin.com/en/
Luu L, Narayanan V, Zheng C, Baweja K, Gilbert S, Saxena P (2016) A secure sharding protocol for open blockchains. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, pp. 17–30
Manevich Y, Akavia A (2022) Cross chain atomic swaps in the absence of time via attribute verifiable timed commitments. In: 2022 IEEE 7th european symposium on security and privacy (EuroS &P), pp. 606–625. IEEE
Mazumdar S (2022) Towards faster settlement in HTLC-based cross-chain atomic swaps. In: 2022 IEEE 4th international conference on trust, privacy and security in intelligent systems, and applications (TPS-ISA), pp. 295–304. IEEE
Midorikawa S (2019) Elliptic-Curve Cryptography Library. https://github.com/elliptic-shiho/ecpy
Mitsunari S, Sakai R, Kasahara M (2002) A new traitor tracing. IEICE Trans Fundam Electron Commun Comput Sci 85(2):481–484
Nakamoto S (2008) Bitcoin whitepaper. URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)
Noether S (2014) Review of cryptonote white paper. HYPERLINK http://monero.cc/downloads/whitepaper_review.pdf
Noether S (2018) Discrete logarithm equality across groups
Poelstra A (2018) Library for EC operations on curve secp256k1. https://github.com/apoelstra/secp256k1-zkp
Poon J, Dryja T (2016) The bitcoin lightning network: scalable off-chain instant payments
Qin X, Pan S, Mirzaei A, Sui Z, Ersoy O, Sakzad A, Esgin MF, Liu JK, Yu J, Yuen TH (2023) Blindhub: Bitcoin-compatible privacy-preserving payment channel hubs supporting variable amounts. In: 2023 IEEE symposium on security and privacy (SP), pp. 2462–2480. IEEE
Reyna A, Martín C, Chen J, Soler E, Díaz M (2018) On blockchain and its integration with IoT. Challenges and opportunities. Future Gener Comput Syst 88:173–190
Sahai A (1999) Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: 40th annual symposium on foundations of computer science (Cat. No. 99CB37039), pp. 543–553. IEEE
Schwartz D, Youngs N, Britto A (2014) The ripple protocol consensus algorithm. Ripple Labs Inc White Paper 5(8):151
Shlomovits O, Leiba O (2020) Jugglingswap: scriptless atomic cross-chain swaps. arXiv preprint arXiv:2007.14423
Smith C (2023) SymPy. https://github.com/sympy/sympy
Tairi E, Moreno-Sanchez P, Maffei M (2021) A^{2}l: Anonymous atomic locks for scalability in payment channel hubs. In: 2021 IEEE symposium on security and privacy (SP), pp. 1834–1851. IEEE
Thyagarajan SAK, Malavolta G (2021) Lockable signatures for blockchains: scriptless scripts for all signatures. In: 2021 IEEE symposium on security and privacy (SP), pp. 937–954. IEEE
Thyagarajan SA (2022) Cryptographic locks for scriptless cryptocurrency payments. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Thyagarajan SAK, Bhat A, Malavolta G, Döttling N, Kate A, Schröder D (2020) Verifiable timed signatures made practical. In: Proceedings of the 2020 ACM SIGSAC conference on computer and communications security, pp. 1733–1750
Thyagarajan SA, Malavolta G, Moreno-Sanchez P (2022) Universal atomic swaps: secure exchange of coins across all blockchains. In: 2022 IEEE symposium on security and privacy (SP), pp. 1299–1316. IEEE
TierNolan (2013) Atomic Swap - Bitcoin Wiki. https://en.bitcoin.it/wiki/Atomic_swap
Tsabary I, Yechieli M, Manuskin A, Eyal I (2021) MAD-HTLC: because htlc is crazy-cheap to attack. In: 2021 IEEE symposium on security and privacy (SP), pp. 1230–1248. IEEE
Wang G, Nixon M (2021) Intertrust: towards an efficient blockchain interoperability architecture with trusted services. In: 2021 IEEE international conference on blockchain (Blockchain), pp. 150–159 . IEEE
Zamani M, Movahedi M, Raykova M (2018) Rapidchain: scaling blockchain via full sharding. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, pp. 931–948
Zamyatin A, Al-Bassam M, Zindros D, Kokoris-Kogias E, Moreno-Sanchez P, Kiayias A, Knottenbelt WJ (2021) Sok: communication across distributed ledgers. In: Financial cryptography and data security: 25th international conference, FC 2021, Virtual Event, Part II 25, pp. 3–36. Springer
Zamyatin A, Harz D, Lind J, Panayiotou P, Gervais A, Knottenbelt W (2019) Xclaim: trustless, interoperable, cryptocurrency-backed assets. In: 2019 IEEE symposium on security and privacy (SP), pp. 193–210. IEEE

Publisher's Note

Yang Tao Yang Tao (Email: taoyang@iie.ac.cn) is the corresponding author of this paper. She is an engineer in the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, China. She obtained her PhD from the Institute of Information Engineering, Chinese Academy of Sciences. Her main research interests include post-quantum cryptography and lattice-based cryptography.