CAPE: Commitment-based Privacy-Preserving Payment Channel Scheme in Blockchain

Keke Gai, *Senior Member, IEEE,* Yunwei Guo, Jing Yu, *Member, IEEE,* Weilin Chan, Liehuang Zhu, *Senior Member, IEEE,* Yinqian Zhang, *Senior Member, IEEE*, Weizhi Meng, *Senior Member, IEEE*

Abstract—Ensuring scalability in cryptocurrency systems is significant in guaranteeing real-world utility along with the remarkable increment of cryptographic currency. As an alternative in solving scalability issue, payment channel allows users to deliver extensive offline transactions without uploading massive transaction details to the blockchain, such that increasing efficiency can be achieved. However, the implementation of payment channel still encounters privacy concerns when considering the publicly available transaction amounts and the potentials in mining associations between transaction parties. In this paper, we propose a novel payment channel scheme, entitled Commitment-based Anonymous Payment ChannEl (CAPE), to facilitate unlimited off-chain bidirectional payments while guaranteeing participants' privacy. The proposed scheme adopts zero-knowledge proof (zk-SNARKs) and verifiable timed (VTD) commitments to ensure the anonymity of the relationship between on-chain and off-chain transactions, privacy of transaction amounts, and security of balances. We comprehensively formalize security definitions and present rigorous proofs for each security attribute. Experiment results further demonstrate the practical viability of CAPE.

Index Terms—Payment channel, privacy-preserving, zero-knowledge proof, blockchain, commitment

1 Introduction

Inspired by the dissemination of Bitcoin [1] , trust management has been dramatically changed from a centralized setting to a decentralized setting through a shared public distributed ledger. For example, in Bitcoin system, the trust model will sustain itself as long as the public ledger satisfies the security assumption. However, the decentralized nature of the blockchain poses an obstacle to the further development of blockchain-based cryptocurrencies [2][3]. Compared to traditional centralized solutions [4], decentralized cryptocurrencies [5][6] are restricted by two obstacles, including low transaction rates and low transaction throughputs. This phenomenon implies that the scalability of the blockchain system is a fundamental factor for determining whether cryptocurriencies or Decentralized Applications (DApp) are practical [7] .

Payment Channel Technology (PCT) [8] is an "off-chain" technology, which has emerged as one of the promising solutions to the scalability challenge [9][10][11]. PCT increases the transaction rate of decentralized cryptocurrencies without altering the existing blockchain consensus protocols [12][13]. PCT enables users to securely conduct large-scale offline transactions without interacting with the blockchain beyond opening and closing the payment channel. Theoretically, a channel is able to scale up to an infinite transaction capability if there is sufficient hardware support. Currently, payment channels are widely deployed for large-scale payments of major cryptocurrencies (e.g., Bitcoin, Ether [14] and Ripple [15]) as an effective solution to scalability and performance issues on the blockchain [16].

PCT involves both parties to a transaction recording the initial and final state of the payment channel on the blockchain ledger. In contrast, all intermediate transaction processes are conducted off-chain. A payment channel [17] consists of three stages, including channel opening, off-chain transaction, and closing. Within the channel opening stage, both parties to the channel transaction must lock their respective funds on blockchain to prevent participants from misappropriating or unilaterally withdrawing the locked funds. At the off-chain transaction stage, both sides of the channel generate multiple transactions off-chain to redistribute the funds locked in the channel. In addition, both parties to a channel transaction submit an off-chain transaction to the blockchain at the channel closing stage, which will close the payment channel and lock the funds once the blockchain verifies that the submission is the latest transaction from the off-chain transaction. Then, the locked funds will be consumed.

Although payment channels provide blockchain systems with optional solutions to solving scalability issues, existing cryptocurrency payment channel systems still encounter some issues. In DMC [18], for example, despite payments occur at an off-chain state, any party can access transaction amount information between two parties through payment channel update transactions, ultimately exposed on the blockchain ledger. Similarly, Bolt [19] indicates that the linkage relationships between transaction parties can also be exposed due to an on-chain establishment and closure of

- *K. Gai, Y. Guo, W. Chan, and L. Zhu are with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, 100081 China. E-mail: {gaikeke, 3220211008, chanweilin, liehuangz}@bit.edu.cn*
- *J. Yu is with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China. (E-mail: yujing02@iie.ac.cn).*
- *Y. Zhang is with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, Guangdong, China. Email: yinqianz@acm.org.*
- *W. Meng is with Department of Applied Mathematics and Computer Science Cybersecurity Engineering, Technical University of Denmark, Kongens Lyngby, Denmark. Email: weme@dtu.dk.*

Manuscript received Month XX, 20XX; revised Month XX, 20XX.

payment channels. Consequently, payment channels lacks privacy protections for two types of information in transactions, which are amount and transaction unlinkability (see followings).

- *Amount privacy:* The system shall ensure that apart from the two parties involved in the payment channel transaction, no other party can obtain the specific transaction amount(s).
- *Transaction unlinkability:* The system shall ensure that no other party can explicitly link the sender and receiver of the payment, apart from the two parties involved in the payment channel transaction.

We notice that some prior academic work has explored solutions to address the privacy issues inherent in payment channels. Bolt [19] is a blind signature-based off-chain payment method integrating with Zero-Knowledge Proof (ZKP). The method utilizes a ZKP to protect relationship-related privacy between two parties in a transaction and uses blind signatures to verify channel updates. Similarly, Thyagarajan *et al*. [20] propose a payment channel protocol, called PayMo, which is based on verifiable timed linkable ring signatures. This approach uses Monero to provide a certain level of privacy protection for the payment channel. However, the blind signatures used in Green's [19] proposal are incompatible with Bitcoin. Meanwhile, PayMo [20] is a proposal tailored for Monero and only supports one-way payments, making it unsuitable for further extension to bidirectional payments.

To solve the privacy issue in the payment channel without causing additional consumption, we propose a commitment-based privacy-preserving payment channel scheme, by using ZKP [21] and other technologies (e.g., verifiable timed dlog) [22], [23], to realize privacy-preserving and fair transactions of payment channels. Our approach ensures that users can use the payment channel for transactions while guaranteeing participants' privacy, including amount privacy and transaction unlinkability. Meanwhile, our approach punishes channel users for improper behavior within a predetermined time range by locking a secret value in a timelocked verifiable commitment. Each party locks its value in a timed verifiable commitment and entrusts it to the other party, ensuring it can unlock and access it after a certain period. Therefore, when one party fails in complying with the agreement, the other party can open the timed verifiable commitment within the predetermined time frame to obtain the other party's secret and access all the currency stored in the payment channel.

Main contributions of this work are as follows:

- This work has proposed a commitment-based privacy-preserving scheme for bidirectional payment channels. To the best of our knowledge, the proposed solutions in achieving transaction unlinkability and amount privacy in bidirectional payment channels are proposed for the first time.
- Our approach ensures no party can block or steal other parties' funds when the payment channel is closed. This is completed by both parties locking their secret values in a timed verifiable commitment. Then, when one party fails in adhering to the agreement within the specified time, the other party will penalize them by unlocking the commitment. Meanwhile, our approach provides efficient bidirectional payment channels on any scriptless blockchain.
- The proposed approach is based on Libsnark and Ganache, which has been demonstrated by extensive experiments, covering evaluations of three stages in the paymen channel transaction processes as well as the examination of ZKP in the entire system. The experiment results provide evidence for proving the feasibility of the proposed scheme. The security analysis with proof of the proposed CAPE scheme has been given in this work.

The remainder of this paper is organized as follows. We review the relevant preliminaries in Section 2. The concepts and definitions related to the CAPE scheme are presented in Section 3. Then, the proposed CAPE scheme is presented in Section 4, prior to the performance analyses in Section 5. We review the related approaches in Section 6. Finally, we conclude this paper in Section 7.

2 Preliminaries

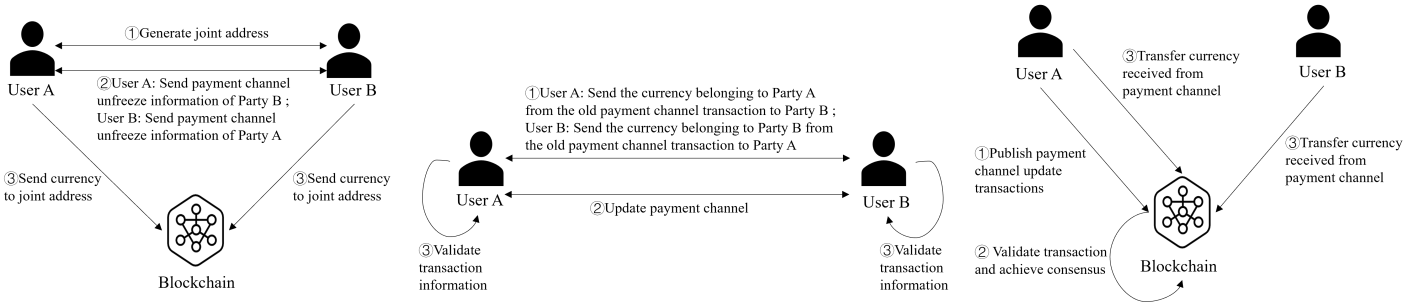
Verifiable timed DLog is proposed on the basis of homomorphic time-lock puzzles [22], [24]. A Verifiable timed dlog is defined with respect to a group G of order q and a generator G. The scheme allows an operator to generate a time lock against Y based on the discrete logarithmic value Y and the corresponding secret value y selected in the group G. In the time lock, on one hand, the time lock guarantees that any party acquiring the time lock can acquire the secret value y corresponding to the discrete logarithmic value Y after the computation time **T**; on the other hand, the time lock guarantees that without having to unlock the time lock, any party can openly verify whether the time lock contains the secret value y corresponding to the known discrete logarithmic value Y. The verifiable timed dlog usually consists of four algorithms: the Commit algorithm, the Verify algorithm, the Open algorithm, and the ForceOp algorithm. The algorithms are described in details as below.

- Commit algorithm: The input of this algorithm is a discrete logarithmic value Y corresponding to the secret value y and the lock time **T**, and its output is the commitment C and the proof π.
- Verify algorithm: The input of the algorithm is a discrete logarithmic value Y, a commitment C with lock time **T** and a proof π.It outputs 1 if and only if the embedding y in C satisfies that it is the secret value corresponding to the discrete logarithmic value Y in the group G, otherwise it outputs 0.
- Open algorithm: The input to this algorithm is the time commitment C of a signature σ, and after **T** units of time have passed, it outputs the secret value y and the random number r used to generate C.
- ForceOp algorithm: Inputs include the time commitment C of the signature σ. After being calculated in sequence for time **T**, it outputs the secret value y corresponding to the discrete logarithm Y.

3 System Concepts

3.1 Data Structure

The data structure consists of a few components as below:



(a) Payment channel setup (b) Payment channel update (c) Payment channel closure

Fig. 1: The workflow of payment channel consists of major procedures, i.e., Payment Channel Setup, Payment Channel Update, and Payment Channel Closure.

Ledger. At any given time T, all users can access $\mathrm{L_T}$, a sequence containing many transactions. Since the proposed payment channel scheme is built upon a fundamental decentralized cryptocurrency (such as Bitcoin), the ledger includes the original transactions and the other seven new types described below. Additionally, the ledger is append-only (if $\mathrm{T}' < \mathrm{T}$, all $\mathrm{L_{T'}}$ will be stored in $\mathrm{L_{T'}}$).

Addresses. Each user possesses at least one pair of address keys (pk, sk), where pk represents the public address and sk represents the private address. The public address pk is published on blockchain, allowing anyone to pay the user. The private address sk receives payments sent to pk.

Zero-knowledge currencies. Zero-knowledge currencies are the transformed form of plaintext base currency users hold. This paper defines three types of zero-knowledge currencies: basic zero-knowledge currency, Type-1 transfer zero-knowledge currency, and Type-2 transfer zero-knowledge currency. The contents included in each type of zero-knowledge currency are as follows:

- Basic zero-knowledge currency c_i: Numerical value v_i, representing the plaintext currency value corresponding to the basic zero-knowledge currency as currency units. Serial number sn_i, uniquely associated with c_i. Address pk_i, the address of the user who owns this zero-knowledge currency. Private key sk_i, the private key of the user who owns this zero-knowledge currency. Random number r_i, used to obfuscate the zero-knowledge currency. Commitment cm_i, a proof that the user possesses v_i amount of the base currency, in the form cm_i=COMM(pk_i, v_i, sn_i, r_i).
- Type-1 transfer zero-knowledge currency c_i: Numerical value v_i, representing the plaintext currency value corresponding to the zero-knowledge currency in base currency units. Serial number sn_i, uniquely associated with c_i. Address pk_i, the address of the user who receives this zero-knowledge currency. Address pk_j, the address of the sender. Random number r_i, used to prevent duplicate serial numbers. The serial number sn_j is associated with the transferred zero-knowledge currency from the sender. Commitment cm_i, a proof that the user possesses v_i amount of the base currency, in the form cm_i=COMM$(pk_i, pk_j, v_i, sn_i, sn_j)$.
- Type-2 transfer zero-knowledge currency c_i: The content of type-2 transfer zero-knowledge currency is similar to type-1 transfer zero-knowledge currency, with two differences. 1) Including the discrete logarithm value Y is a parameter for ensuring fair transactions. 2) The form of commitment cm_i becomes $cm_i = \mathrm{COMM}(pk_i, pk_j, v_i, sn_i, sn_j, Y)$.

Commitment set CMList and serial number set SNList. Both the commitment and serial number set are public data sets. Given any time T, the commitment set $\mathrm{CMList_T}$ contains all the commitments that appear in the ledger transactions $\mathrm{L_T}$. Given any time T, the serial number set $\mathrm{SNList_T}$ contains all the serial numbers that appear in the ledger transactions $\mathrm{L_T}$.

Merkle tree. Given any time T, $\mathrm{CMTree_T}$ represents the Merkle tree constructed from the commitments in $\mathrm{CMList_T}$, with each commitment as a leaf node, and rt_T represents its root. Given any time T, the function $\mathrm{Path_T(cm)}$ represents the authentication path from a commitment cm that appears in $\mathrm{CMList_T}$ to the root rt_T of $\mathrm{CMTree_T}$.

3.2 Overview of CAPE

The key ideas of the proposed scheme covers a few aspects. We use a hash-based commitment to conceal the transaction amounts between users in the payment channel. ZKP and Merkle trees are adopted to ensure the correctness of transactions and the unlinkability between the transaction parties. Timed verifiable commitments guarantee fairness during the payment channel transaction process.

The CAPE scheme consists of five phases, namely, the setup, mint, payment channel setup, payment channel update, and payment channel closure phases. The setup phase occurs before any other phase, during which the standard parameters of the entire system are initialized. The mint phase occurs before the establishment of payment channels between the parties. In the mint phase, both parties to the transaction convert plaintext currency on the blockchain into hidden zero-knowledge currency, where ZKPs are used to ensure the correctness of the transaction. Simultaneously, to conceal the zero-knowledge currency they intend to spend, the newly minted zero-knowledge currency is mixed with other zero-knowledge currencies to form a Merkle tree.

The transaction process between the parties of the payment channel consists of three phases: payment channel

setup phase, payment channel update phase, and payment channel closure phase, as shown in Fig. 1.

The specific flow of the payment channel setup phase is shown in Fig 1a. In the payment channel setup phase, the parties first jointly generate a joint address and then transfer their respective zero-knowledge currencies to the new address. When spending zero-knowledge currency, ZKPs are required to prove ownership of the zero-knowledge currency in the hidden Merkle tree. This is to ensure the fairness of the transaction and to conceal the sender-receiver link. However, before both parties transfer zero-knowledge currency to the joint address, to prevent one party from going offline and preventing the other party from claiming their rightful amount, this paper employs timed verifiable commitments to ensure that both parties of the payment channel can claim their owed zero-knowledge currency after a fixed time.

The specific flow of the payment channel update phase is shown in Fig 1b. In the payment channel update phase, the parties can negotiate the allocation of zero-knowledge currency on the joint address. However, to prevent users from maximizing personal interests by publishing old payment channel update transactions to close the payment channel, before generating a new payment channel update transaction, the old payment channel update transaction's zero-knowledge currency needs to be withdrawn. Additionally, to give the parties of the payment channel time to detect whether the other party has published old transactions, we use timed verifiable commitments to require the user who posts the closing channel transaction to wait for a certain period before obtaining zero-knowledge currency. Thus, when one party publishes an old transaction, the other party can take measures to get all the currency in the payment channel of the party that posted the old transaction to ensure the fairness of the transaction.

The specific flow of the payment channel closur is shown in Fig 1c. In the payment channel closure phase, both parties can close the payment channel by publishing the latest payment channel update transaction. Once the payment channel update transaction is uploaded, miners on the blockchain can verify the correctness of the ZKPs in the transaction. After successful verification, the payment channel is closed completely. However, to protect the security of the currency obtained, each party needs to transfer the accepted currency to a new address separately.

4 Proposed Model

As we mentioned in prior sections, existing payment channel techniques still encounter privacy concerns. It achieves privacy-preserving for the identities and transactions of both P_i and P_j parties to a transaction in a payment channel. The key idea of this scheme is using a hash-based commitment scheme to conceal the transaction amounts between users in the payment channel. Then, ZKP and Merkle trees are utilized to ensure the correctness of transactions and the unlinkability between the transaction parties. VTD commitments guarantee fairness during the payment channel transaction process. This scheme consists of the following five phases: Setup, Mint, Payment Channel Setup, Payment Channel Update and Payment Channel Closure.

Algorithm 1 Mint Algorithm

Input: The list of public parameters pp, the coin value to be converted v_i and address pk_i;

Output: A zero-knowledge currency c_i and a mint transaction tx_{Mint};

1: Randomly sample a random number r_i
2: Compute a new serial number sn_i = PRF(sk_i, r_i)
3: Compute a new commitment cm_i=COMM(pk_i,v_i,sn_i,r_i)
4: Set the information that needs to be disclosed to generate a ZKP x :=(cm_i, v_i, pk_i) and hidden evidence w :=(sn_i, sk_i, r_i)
5: Compute π_{Mint}=zk-SNARKs.GenProof(pk_{zMint}, x, w)
6: Set zero-knowledge currency c_i :=(cm_i,pk_i,v_i,sn_i,r_i)
7: Set m :=($\pi_{\text{Mint}}, cm_i, v_i$)
8: Generate a signature on m using private key sk_i σ_{Mint} =BLS.Sign(m, sk_i)
9: Set tx_{Mint}:=($m, pk_i, \sigma_{\text{Mint}}$)
10: Output zero-knowledge currency $c_i$and mint transaction tx_{Mint}

4.1 Phase I: Setup Phase

The system public parameter list pp will be initialized during the setup phase. In the setup phase: first, for a given security parameter λ, we will generate the public parameter pp_z for it using the Initialization Algorithm in zk-SNARKs;

Then, the KenGen algorithm in zk-SNARKs is used to generate a key pair $(pk_{\text{zi}}, vk_{\text{zi}})$ for each specific circuit $\mathcal{C}_{\text{i}}$ required for a commitment-based anonymous payment channel scheme. These circuits are $\mathcal{C}_{\text{Mint}}$, $\mathcal{C}_{\text{SetSend}}$, $\mathcal{C}_{\text{UnFreeze}}$, $\mathcal{C}_{\text{Update}}$, $\mathcal{C}_{\text{Rev}}$,$\mathcal{C}_{\text{Transfer1}}$ and $\mathcal{C}_{\text{Transfer2}}$, which will be used for the generation and verification of transaction proofs during the payment channel. At the same time, we need to set the relevant public parameter pp_{BLS} of the BLS signature algorithm and the relevant public parameter pp_{DLG} of the discrete logarithm. Note that this phase is executed only once to output the list of public parameters. A trusted third party executes this phase at the beginning of the ledger creation, and it is executed only once, and the output is made public to all users.

4.2 Phase II: Mint Phase

Before engaging in transactions within a commitment-based anonymous payment channel, any participating user must possess a specific quantity of zero-knowledge currency acquired through a minting algorithm. When user P_i intends to convert their currency using the minting algorithm, the following steps are followed: Firstly, user P_i must have at least one public address pk_i on the blockchain, with an adequate amount of plaintext currency in that address. Subsequently, user P_i employs the minting algorithm to generate a mint transaction tx_{Mint}, facilitating the conversion of a designated amount of plaintext currency into zero-knowledge currency. The mint transaction tx_{Mint} associated with user P_i's public address pk_i comprises the following variables:

- Address pk_i: it is the address of transaction sender and the address of the transaction receiver.

- Value v_i: it is the value of minting transactions that need to be transformed from plaintext currency to zero-knowledge currency.
- Commitment Value cm_i: the commitment scheme $COMM$ generates a fresh zero-knowledge currency commitment value. This commitment value encapsulates the hidden components, including the address pk_i, value v_i, the newly generated random number r_i, and a unique string sn_i generated by the PRF function associated with this specific commitment.
- ZKP π_{Mint}: The ZKP is generated by zk-SNARKs.GenProof, is to prove the validity of the following equation for the C_{Mint} circuit.

  1) $cm_i = \text{COMM}\ (pk_i, v_i, sn_i, r_i)$
  2) $sn_i = \text{PRF}\ (sk_i, r_i)$

- Signature σ_{Mint}:User P_i signs the above (π_{Mint},cm_i, v_i) with private key sk_i.

The detailed procedure of the user P_i Mint algorithm is shown above (see Algorithm 1).

4.3 Phase III: Payment Channel Setup Phase

Once both parties involved in the transaction, namely user P_i and user P_j, possess a certain quantity of zero-knowledge currency for conducting transactions through the payment channel, they can proceed to the payment channel establishment phase. During this phase, the following steps are undertaken: Firstly, users P_i and P_j need to generate a payment channel address pk_{2-PC} shared by the two parties and the corresponding private key sk_{2-PC} (the private key sk_{2-PC} consists of sk^i_{2-PC} and sk^j_{2-PC}. sk^i_{2-PC} is private to user P_i and sk^j_{2-PC} is private to user P_j).

Then, users P_i and P_j need to deposit funds (v_i, v_j) to address pk_{2-PC} to complete the payment channel establishment. When the user P_i wants to transfer the funds v_i to the address pk_{2-PC}, it is done by the payment channel establishment and sending algorithm. The algorithm generates a payment channel establishment and sending transaction tx_{SetSend}, which transfers the funds to be transferred by user P_i to address pk_{2-PC}. The tx_{SetSend} transaction consists of the following variables.

- Merkle tree root rt: This is the proof that the commitment cm_i^{old} exists in the ledger;
- Commitment serial number sn_i^{old}: A unique string associated with the commitment cm_i^{old}.
- Commitment value cm_i^{pc}: This value is generated by the commitment scheme COMM, and the content implied in the commitment includes the address pc_{2-PC}, the address pc_i,the transferred value v_i, serial number sn_i^{old} and serial number sn_i^{pc} associated with this commitment value generated by the PRF function;
- ZKP π_{SetSend}: This zero-knowledge proof is generated by zk-SNARKs.GenProof, proving that the following conditions apply to the circuit of $\mathcal{C}_{\text{SetSend}}$.

  1) cm_i^{old} =COMM($pk_i, v_i, sn_i^{old}, r_i^{old}$)
  2) sn_i^{old} =PRF(sk_i, r_i^{old})
  3) cm_i^{pc} =COMM($pk_{2-PC}, pk_i, v_i, sn_i^{old}, sn_i^{pc}$)
  4) sn_i^{pc} =PRF(pk_{2-PC}, r_i^{new})
  5) The path $path_i$ from cm_i^{old} to the rt saved on the ledger is correct (the commitment cm_i^{old} owned by P_i is on the ledger)

Algorithm 2 Payment Channel Establishment and Sending Algorithm

Input: The list of public parameters pp, Merkle root rt, path $path_i$,Zero-knowledge currency c_i^{old} and address pk_{2-PC};

Output: Zero-knowledge currency c_i^{pc} and payment channel establishment and sending transactions tx_{SetSend};

1: Parsec_i^{pc} :=($cm_i^{old}, pk_i, v_i, sn_i^{old}, r_i^{old}, sk_i$)
2: Randomly sample a random number r_i^{new} and compute serial number sn_i^{pc} =PRF(pk_{2-PC}, r_i^{new})
3: Compute the new commitment cm_i^{pc}=COMM(pk_{2-PC}, pk_i, v_i, sn_i^{old}, sn_i^{pc})
4: Set c_i^{pc} :=($cm_i^{pc}, pk_{2-PC}, pk_i, v_i, sn_i^{pc}, sn_i^{old}, r_i^{new}$)
5: Set the information that needs to be disclosed to generate a ZKP x :=($rt, cm_i^{pc}, sn_i^{old}$)
6: Set hidden evidence w :=($path_i, cm_i^{old}, pk_{2-PC}, pk_i, v_i$, sn_i^{pc}, $r_i^{old}, r_i^{new}, sk_i$)
7: Compute π_{SetSend}=zk-SNARKs.GenProof(pk_{zSetSend},x, w)
8: Set tx_{SetSend}:=($rt, cm_i^{pc}, sn_i^{old}, \pi_{\text{SetSend}}$)
9: Output zero-knowledge currency c_i^{pc} and payment channel establishment and sending transactions tx_{SetSend}

The specific execution of Algorithm 2 is shown above.

After users P_i and P_j generate new zero-knowledge currencies c_i^{pc} and c_j^{pc} belonging to the payment channel address pk_{2-PC}: first, they send the information of the newly-generated zero-knowledge currencies to each other through the secured channel, respectively; then, users P_i and P_j send the payment channel establishment and sending transaction to the blockchain to complete the transfer. Before users P_i and P_j post the payment channel establishment and send transactions to the blockchain to complete the transfer, we need to consider the possibility that one party goes offline after the payment channel is established. When this happens, it will result in the other party's funds being locked in the joint address pk_{2-PC} forever. Therefore, to prevent this from happening, before users P_i and P_j post a transaction, users P_i and P_j need to jointly generate their own payment channel unfreeze transactions to ensure that they can retrieve their original funds after a certain period of time.

To unfreeze the funds transferred by user P_i to the payment channel address pk_{2-PC}, user P_i is required to transfer the funds from the payment channel address pk_{2-PC} to the address of P_i via a payment channel unfreezes transaction tx_{UnFreeze}. Meanwhile, for P_i to receive the rightful currency only after a fixed time **T** (the selection of the corresponding time **T** is negotiated off-chain between the two parties to the transaction), it is required in this paper that the payment channel unfreezes transaction tx_{UnFreeze} only becomes effective after the user P_i has supplied the secret value y corresponding to the discrete logarithm value Y, which is locked in the VTD commitment, which can only be obtained after at least a fixed time **T** has elapsed.

Algorithm 3 Payment Channel Unfreeze Algorithm

Input: The list of public parameters pp, c_i^{pc},the P_j has the partial private key sk_{2-PC}^j of address pk_{2-PC}, lock time **T**;

Output: The new zero-knowledge currency $c_i{}^{\mathrm{UnFreeze}}$, payment channel unfreeze transactions tx_{UnFreeze} containing partial signatures,VTD commitment $(C_{VTD}, \pi_{\mathrm{VTD}})$;

1: Parse c_i^{pc} :=($cm_i^{pc}, pk_{2-PC}, pk_i, v_i, sn_i^{pc}, sn_i^{old}, r_i^{old}$)
2: Randomly select secret values y, and compute discrete logarithm $Y = g^y$.
3: Compute the new commitment cm_i^{new} =COMM(pk_i, pk_{2-PC}, v_i, sn_i^{pc}, sn_i^{new}, Y)
4: Set c_i^{new} :=($cm_i^{new}, pk_{2-PC}, pk_i, v_i, sn_i^{pc}, sn_i^{new}, Y$)
5: Set the information that needs to be disclosed to generate a ZKP x :=(cm_i^{pc}, cm_i^{new}, pk_{2-PC}, sn_i^{pc}, Y)
6: Set hidden evidence w :=($pk_i, v_i, sn_i^{old}, sn_i^{new}, r_i^{old}$)
7: Compute π_{UnFreeze}=zk-SNARKs.GenProof($pk_{\mathrm{zUnFreeze}}$, x, w)
8: Set m :=($cm_i^{pc}, cm_i^{new}, sn_i^{pc}, Y, \pi_{\mathrm{UnFreeze}}$)
9: Compute the signature $\sigma_{\mathrm{UnFreeze}}^{(j)}$ of the private key sk_{2-PC}^j for m
10: Generate VTD commitment $(C_{\mathrm{VTD}}, \pi_{\mathrm{VTD}})$ = VTD. Commit(y,T) based on parameters **T** and y
11: Set tx_{UnFreeze} =($m, pk_{2-pc}, \sigma_{\mathrm{UnFreeze}}^{(j)}$)
12: Output VTD commitment $(C_{\mathrm{VTD}}, \pi_{\mathrm{VTD}})$, zero-knowledge currency c_i^{new} and payment channel unfreeze transactions tx_{UnFreeze}

The payment channel unfreezes transaction tx_{UnFreeze} for user P_i is generated as follows: First, P_j runs the payment channel unfreezes algorithm, which generates a payment channel unfreezes transaction tx_{UnFreeze} containing a partial signature and a timed VTD commitment $(C_{\mathrm{VTD}}, \pi_{\mathrm{VTD}})$. The tx_{UnFreeze} transaction contains the following variables.

- Old commitment values cm_i^{pc}: the commitment value corresponding to the old zero-knowledge currency c_i^{pc} is denoted as cm_i^{pc}.;
- Serial number sn_i^{pc}: the string associated with the commitment cm_i^{pc};
- New zero-knowledge currency commitment value cm_i^{new}: it is generated by the commitment scheme COMM, and implicit in the commitment with the address pk_i, the address pk_{2-PC}, the transferred value v_i, the commitment serial number sn_i^{pc} and a unique string sn_i^{new} associated with this commitment value, generated by the PRF function;
- Address pk_{2-PC}: payment channel address;
- ZKP π_{UnFreeze}: this ZKP is generated by zk-SNARKs.GenPr-
  oof that the following conditions apply to the circuit of C_{UnFreeze}.
  1) cm_i^{pc} =COMM($pk_{2-PC}, pk_i, v_i, sn_i^{old}, sn_i^{pc}$)
  2) sn_i^{old} =PRF(pk_{2-PC}, r_i^{old})
  3) cm_i^{new}=COMM($pk_{2-pc}, pk_i, v_i, sn_i^{pc}, sn_i^{new}$)
  4) sn_i^{new}=PRF(pk_i, Y)
- Discrete logarithmic value y and secret value y:$Y = g^y$(g is a public parameter);
- Signature $\sigma_{\mathrm{UnFreeze}}$:Signature of the above (π_{UnFreeze}, cm_i^{pc}, cm_i^{new}, sn_i^{pc}, Y) by the payment channel private key sk_{2-PC}.

The specific execution of Algorithm 3 is shown above.

Next, P_j sends a new zero-knowledge currency c_i^{new}; the payment channel unfreezes transaction tx_{UnFreeze} containing the partial signature and the VTD commitment $(C_{\mathrm{VTD}}, \pi_{\mathrm{VTD}})$, which is output by the payment channel unfreezes algorithm, to P_i. When P_i receives the above message, it first verifies that the signature $\sigma_{\mathrm{UnFreeze}}^{(j)}$ included in the payment channel unfreezes transaction tx_{UnFreeze} is correct, and then verifies via VTD.Verify($Y, C_{\mathrm{VTD}}, \pi_{\mathrm{VTD}}$) that y in tx_{UnFreeze} is generated from the secret value y promised in the VTD commitment. When y is computed, it will use its possession of sk_{2-PC}^i to sign and aggregate the payment channel unfreezes transaction tx_{UnFreeze} to generate the complete signature and add y to it to form the complete payment channel unfreezes transaction, which results in the corresponding currency. The operations above ensure that user P_i unfreezes the currency stored at the payment channel address pk_{2-PC} after a fixed time T. Operations performed for unfreezing the funds stored on the joint public key address pk_{2-PC} by the user P_j are similar and only require the operations between P_i and P_j.

Once the participating users of the payment channel, P_i and P_j, have obtained the VTD commitment sent by the other party through the payment channel unfreezes algorithm and have verified it correctly, they can send their respective payment channel establishment transactions to the chain to accomplish the goal of transferring a certain amount of money to the address pk_{2-PC}. Thus, the payment channel establishment phase is completed.

4.4 Phase IV: Payment Channel Update Phase

When the payment channel is formally established, the transaction parties P_i and P_j can proceed to the next phase, which is the actual payment transaction between the two parties.

(a) Payment Channel Trading Update:

During this phase, the two parties to a payment channel transaction allocate the zero-knowledge currency on the payment channel address pk_{2-PC} based on the transactions between them and reallocate it with each transaction until the end. During each allocation of the currency on the payment channel pk_{2-PC}, two different versions of the transaction payment channel update transaction tx_{Update} are generated, one version under the control of party P_i and the other version under the control of party P_j. By "under the control of party P_i", we mean that party P_i has all the conditions needed to publish the payment channel update transaction tx_{Update}, and the same is true for "under the control of party P_j". The two different versions of the transaction are both re-allocations of the currency on the payment channel pk_{2-PC}. In both versions, the same currency is allocated to the same party. The difference between the two versions is that to prevent a party from closing the payment channel by posting an old payment channel allocation transaction that is most favorable to the party, the currency acquired by the party in the version-controlled by each party can only be spent after a fixed time **T** (the choice

of which is also chosen in advance by the parties). Like the approach described above, the user must provide a secret value y corresponding to a discrete logarithm value Y when spending the currency, and the secret value y is locked to a VTD commitment. It can only be obtained after at least a fixed time **T**.

The specific process of generating a payment channel update transaction tx_{Update} for a version controlled by user P_i is as follows: First, user P_j runs the following payment channel update algorithm, which generates a payment channel update transaction tx_{Update} containing a partial signature and a VTD commitment $(C_{\text{VTD}}, \pi_{\text{VTD}})$. The tx_{Update} transaction consists of the following variables.

- Merkle tree root rt: the proof that the commitment cm_i^{pc} and cm_j^{pc} exists in the ledger;
- Serial numbers sn_i^{pc} and sn_j^{pc}: strings associated with commitment cm_i^{pc} and commitment cm_j^{pc};
- Commitment values cm_i^{new} and cm_j^{new}: it is also generated by the commitment scheme COMM. The contents implicit in the cm_i^{new} commitment are the address pk_i, the address pk_{2-PC}, the explicit value v_i^{new}, serial number sn_i^{new} associated with this commitment value generated by the PRF function, discrete logarithm Y and the serial number sn_i^{pc}; the contents implicit in the cm_i^{new} commitment are the address pk_j, the address pk_{2-PC}, the explicit value v_j^{new}, serial number sn_j^{new} associated with this commitment value generated by the PRF function, discrete logarithmic Y and the serial number sn_j^{pc};
- Address pk_{2-PC}: payment channel address;
- ZKP π_{Update}: this ZKP is a proof generated by zk-SNARKs.GenProof, which is suitable for the circuit of C_{Update}.
  1) cm_j^{pc} =COMM$(pk_{2-PC}, pk_j, v_j^{old}, sn_j^{old}, sn_j^{pc})$
  2) sn_j^{pc} =PRF(pk_{2-PC}, r_j^{old})
  3) cm_i^{pc} =COMM$(pk_{2-PC}, pk_i, v_i^{old}, sn_i^{old}, sn_i^{pc})$
  4) sn_i^{pc} =PRF(pk_{2-PC}, r_i^{old})
  5) cm_j^{new} =COMM$(pk_{2-PC}, pk_j, v_j^{new}, sn_j^{pc}, sn_j^{new})$
  6) sn_j^{new} =PRF(pk_j, r_j^{new})
  7) cm_i^{new} =COMM$(pk_{2-PC}, pk_i, v_i^{new}, sn_i^{pc}, sn_i^{new}, Y)$
  8) sn_i^{new} =PRF(pk_i, r_i^{new})
  9) $v_j^{old} + v_i^{old} = v_j^{new} + v_i^{new}$
  10) The path $path_i$ from cm_i^{pc} to the rt saved on the ledger is correct
  11) The path $path_j$ from cm_j^{pc} to the rt saved on the ledger is correct
- Signature σ_{Update}: this is a signature of the above $(rt, \pi_{\text{Update}}, cm_i^{new}, cm_j^{new}, sn_i^{pc}, sn_j^{pc})$ by the payment channel private key sk_{2-PC}.

Next, P_j sends the new zero-knowledge currency c_i^{new}, the payment channel update transaction tx_{Update} containing the partial signature, and the VTD commitment $(C_{\text{VTD}}, \pi_{\text{VTD}})$, which are output by the payment channel update algorithm, to P_i. When P_i receives the above message, it first verifies that the signature $\sigma_{\text{Update}}^{(j)}$ included in the payment channel update transaction tx_{Update} is correct, and then verifies via VTD.Verify $(Y, C_{\text{VTD}}, \pi_{\text{VTD}})$ that c_i^{new} of Y is generated from the secret value y promised in the VTD commitment. Once all the above verifications are passed, user P_i can start running VTD.ForceOp(C_i), until y is computed after a time **T**. The above operation guarantees that user P_i, can spend the currency obtained from the payment channel address pk_{2-PC}, only after a fixed time **T** . A similar operation is performed for the payment channel update transaction tx_{Update} for the version controlled by the user P_j, by simply interchanging the above operations between P_i and P_j.

Algorithm 4 Payment Channel Update Generation Algorithm

Input: The public parameter list pp,c_i^{pc} and c_j^{pc}, Merkle tree root rt, path $path_i$and $path_j$, the private key sk_{2-PC}^j corresponding to the payment channel part of the user P_j, lock time **T**, plaintext values v_j^{new} and v_i^{new};

Output: Zero-knowledge currency c_i^{new} and c_j^{new},VTD commitment $(C_{\text{VTD}}, \pi_{\text{VTD}})$,payment channel update transaction tx_{Update} containing a partial signature;

1: Parse c_i^{pc} :=$(cm_i^{pc}, pk_{2-PC}, pk_i, v_i^{old}, sn_i^{pc}, sn_i^{old}, r_i^{old})$ and c_j^{pc} :=$(cm_j^{pc}, pk_{2-PC}.,.pk_j, v_j^{old}, sn_j^{pc}, sn_j^{old}, r_j^{old})$
2: Compute the random numbers r_i^{new} and r_j^{new},and compute serial number sn_j^{new} = PRF(pk_j, r_j^{new}) and sn_i^{new} = PRF(pk_i, r_i^{new})
3: Select secret value y, and compute discrete logarithm $Y = g^y$
4: Compute the new commitment value cm_i^{new} =COMM $(pk_{2-PC}, pk_i, v_i^{new}, sn_i^{pc}, sn_i^{new}, Y)$ and cm_j^{new} = COMM $(pk_{2-PC}, pk_j, v_j^{new}, sn_j^{pc}, sn_j^{new}$
5: Set c_i^{new} :=$(cm_i^{new}, pk_{2-PC}, pk_i, v_i^{new}, sn_i^{pc}, sn_i^{new}, r_i^{new}$, Y) and c_j^{new} := $(cm_j^{new}, pk_{2-PC}, pk_j, v_j^{new}, sn_j^{pc}, sn_j^{new}$, r_j^{new})
6: Set the information that needs to be disclosed to generate a ZKP x :=$(rt, cm_j^{new}, cm_i^{new}, pk_{2-PC}$,$sn_j^{pc}, sn_i^{pc}$) and Set hidden evidence w :=$(path_j, path_i, cm_j^{pc}, cm_i^{pc}$, $pk_j, pk_i, v_j^{old}, v_i^{old}, v_j^{new}, v_i^{new}, sn_j^{new}, sn_i^{new}, sn_j^{old}$, $sn_i^{old}, r_j^{new}, r_i^{new}, r_j^{old}, r_i^{old}, Y$)
7: Compute π_{Update}=zk-SNARKs.GenProof $(pk_{\text{zUpdate}}$,x, $w)$,Set $m := (rt, \pi_{\text{Update}}, cm_j^{new}, cm_i^{new}, sn_j^{pc}, sn_i^{pc})$ and Generate partial signature $\sigma_{\text{Update}}^{(j)}$ using partial private key sk_{2-PC}^j to m
8: Set up $tx_{\text{Update}} = (m, pk_{2-PC}, \sigma_{\text{Update}}^{(j)})$
9: Generate VTD commitment (C_{VTD},π_{VTD})
10: Output c_i^{new} and c_j^{new}, $(C_{\text{VTD}}, \pi_{\text{VTD}}), tx_{\text{Update}}$

(b) Payment Channel Rollback Trading Process:

Due to the generation of new payment channel update transactions in the payment channel for the reallocation of zero-knowledge currency on the shared payment channel address, there exists a possibility of users attempting to maximize their personal benefits by broadcasting old payment channel update transactions to close the payment channel. To prevent such occurrences, it is necessary to utilize payment channel rollback transactions before generating each new payment channel update transaction. These rollback transactions ensure the reassignment of zero-knowledge currency generated by the old payment channel update transaction. As a result, users attempting to close

the payment channel by broadcasting old payment channel update transactions will lose all of their currency holdings.

For user P_i, the old payment channel under the version controlled by user P_j will be updated through this phase with the currency c_j^{old} :=(cm_j^{old},pk_{2-PC},pk_j,v_j,sn_j^{pc} ,sn_j^{old},r_j^{old},Y)generated by the exchange belonging to P_j is transferred to it. For transferring the currency belonging to P_j generated by the old payment channel update transaction under the control of the user P_j version to P_i is generated as follows: first, the following payment channel rollback algorithm is run by the user P_j, which generates a payment channel Rollback transaction, tx_{Rev}, where the tx_{Rev} transaction consists of the following variables:

- Old commitment values cm_j^{old}: the old zero-knowledge currency c_j^{old} corresponding to the value of the commitment;
- Serial number sn_j^{old}: the string associated with the commitment cm_j^{old};
- New zero-knowledge currency commitment value cm_i^{new}: is also generated by the commitment scheme COMM, and implicit in the commitment with the address pk_i, the address pk_j, the transferred value v_i, serial number sn_j^{old} and serial number sn_i^{new} associated with this commitment value generated by the PRF function;
- Address pk_j : Address of user P_j;
- ZKP π_{Rev}: this ZKP is generated by zk-SNARKs.GenProof that the following conditions apply to the circuit of C_{Rev}.
  1) cm_j^{old} =COMM($pk_{2-PC}, pk_j, v_j, sn_j^{pc}, sn_j^{old}, Y$)
  2) sn_j^{old} = PRF(pk_j, r_j^{old})
  3) cm_i^{new} = COMM($pk_j, pk_i, v_j, sn_j^{old}, sn_i^{new}$)
  4) sn_i^{new} = PRF(pk_i, r_i^{new})
- Discrete logarithm Y and secret value $y : Y = g^y$(g is a public parameter)
- Signature σ_{Rev}: Signature of the above (π_{Rev}, cm_j^{old}, cm_i^{new}, sn_j^{old}, Y) by the payment channel private key sk_j.

The details of the payment channel rollback algorithm are shown above (see Algorithm 5):

Then, P_j sends the new zero-knowledge currency c_i^{new} and the payment channel rollback transaction tx_{Rev} output by the payment channel rollback algorithm to P_i. Finally, when P_i receives the above information, it verifies that the signature σ_{Rev} included in the payment channel rollback transaction tx_{Rev} is correct, and if correct ends, otherwise the rollback fails. Adding the secret value y corresponding to the discrete logarithm Y to the received tx_{Rev} in the correct case constitutes the complete payment channel rollback transaction. The action taken for the rollback of zero coins belonging to P_i generated by an old payment channel update transaction under the control of the version of user P_i to user P_j is similar. It only requires interchanging the operations between P_i and P_j as described above.

Through the aforementioned two steps, users in the payment channel can achieve security and accuracy during each transaction. However, there are several points to consider during the payment channel update phase: Firstly,

Algorithm 5 Payment Channel Rollback Algorithm

Input: The public parameter list pp, c_j^{old}, public key pk_i of P_i and private key sk_j of P_j;
Output: Zero-knowledge currency c_i^{new}, payment channel rollback transaction tx_{Rev}^i;
1: Parse $c_j^{old} := (cm_j^{old}, pk_{2-PC}, pk_j, v_j, sn_j^{pc}, sn_j^{old}, r_j^{old}, Y)$
2: Randomly sample a random number r_i^{new} and compute serial number sn_i^{new} =PRF(pk_i, r_i^{new})
3: Compute new commitment value cm_i^{new}=COMM (pk_j, pk_i, v_j, sn_j^{old}, sn_i^{new})
4: Set c_i^{new}:=(cm_i^{new}, pk_j, pk_i, v_j, sn_j^{old}, sn_i^{new}, r_i^{new})
5: Set the information that needs to be disclosed to generate a ZKP x:=(cm_j^{old}, cm_i^{new}, pk_j, sn_j^{old}, Y)
6: Set hidden evidence w:=(pk_{2-PC}, pk_i, v_j, sn_j^{pc}, sn_i^{new}, r_j^{old}, r_j^{new})
7: Compute π_{Rev}=zk-SNARKs.GenProof(pk_{zRev}, x, w)
8: Set $m := (\pi_{\text{Rev}}, cm_j^{old}, cm_i^{new}, sn_j^{old}, Y)$.
9: User P_j uses private key sk_j to generate signature σ_{Rev} =BLS.Sign(m, sk_j) for m.
10: Set $tx_{\text{Rev}} := (m, pk_j, \sigma_{\text{Rev}})$.
11: Output zero-knowledge currency c_i^{new},payment channel rollback transaction tx_{Rev}.

Algorithm 6 Transfer Algorithm

Input: The public parameter list pp, c_i^{old}, Merkle tree root rt and path $path_i$, public key pk_i of P_i , new public key pk_i^{new} of P_i and new private key sk_i^{new} of P_i;
Output: New zero-knowledge currency c_i^{new},transfer transaction $tx_{\text{Transfer1}}$;
1: Parse $c_i^{old} := (cm_i^{old}, pk_{2-PC}, pk_i, v_i, sn_i^{pc}, sn_i^{old}, r_i^{old}, Y)$
2: Select new random number r_i^{new} and compute serial number sn_i^{new}=PRF(sk_i^{new}, r_i^{new})
3: Compute new commitment value cm_i^{new}=COMM(v_i, pk_i^{new}, sn_i^{new}, r_i^{new})
4: Set $c_i^{new} := (cm_i^{new}, pk_i^{new}, v_i, sn_i^{new}, r_i^{new}, sk_i^{new})$
5: Set the information that needs to be disclosed to generate a ZKP x:=(rt, cm_i^{new}, pk_i, sn_i^{old}, Y)
6: Set hidden evidence w:=($path_i$, cm_i^{old}, pk_{2-PC}, pk_i^{new}, v_i, sn_i^{pc}, sn_i^{new}, r_i^{old}, r_i^{new}, sk_i^{new})
7: Compute π=zk-SNARKs.GenProof($pk_{\text{zTransfer1}}$,x, w)
8: Set $m := (\pi_{\text{Transfer1}}, rt, cm_i^{new}, sn_i^{old}, Y)$
9: User P_i uses private key sk_i to generate signature $\sigma_{\text{Transfer1}}$ =BLS.Sign(m, sk_i) for m.
10: Set $tx_{\text{Transfer1}} := (m, pk_i, y, \sigma_{\text{Transfer1}})$,where y is the secret value corresponding to the discrete logarithm value Y .
11: Output zero-knowledge currency c_i^{new},transfer transaction $tx_{\text{Transfer1}}$

during the generation of payment channel update transactions in the first step, it is crucial to ensure the security of the transaction amounts for both parties. The other party generates each party's transaction, and a VTD commitment is employed. This commitment ensures that the party initiating the transaction must wait for a duration, denoted as T, before being able to spend their entitled funds. This waiting period gives the other party enough time to utilize the payment channel rollback transaction to transfer their

funds in case the first party broadcasts an old transaction. Finally, it is essential to note that the setting of the waiting period **T** must be greater than the difference between the current time and the time of payment channel closure. This ensures that both parties do not indefinitely wait without broadcasting the latest payment channel update transaction. This precaution prevents the occurrence of prolonged waiting periods without the release of the most recent payment channel update transaction.

4.5 Phase V: Payment Channel Closure Phase

Once both parties to a payment channel have completed their transactions, they can close the channel by posting the latest version of their respective payment channel update transactions without having to communicate with the other party. When user P_i releases an update transaction tx_{Update} for the version of the payment channel under his control, user P_j can immediately receive his share of the currency and transfer it to his own address, while user P_i has to wait for a certain amount of time before he can transfer his share of the currency to his new address. After the user P_i waits for the set time T, he can completely cut off the pk_{2-PC} relationship with the payment channel by executing the transfer algorithm, through which a transfer transaction will be generated for $tx_{\text{Transfer1}}$, which will transfer the zero-knowledge currency c_i^{old} received by user P_i to a new address pk_i^{new}. Where the $tx_{\text{Transfer1}}$ transaction consists of the following variables.

- Merkle tree root rt: This is the proof that the commitment cm_i^{old} exists in the ledger;
- Commitment serial number sn_i^{old}: A unique string associated with the commitment cm_i^{old}.
- Commitment value cm_i^{new}: This value is generated by the commitment scheme COMM, and the content implied in the commitment includes the address pk_i^{new},the transferred value v_i, new random numbers r_i^{new} and serial number sn_i^{new} associated with this commitment value generated by the PRF function;
- Address pk_i: Address of user P_i ;
- ZKP $\pi_{\text{Transfer1}}$: This ZKP is generated by zk-SNARKs.GenProof that the following conditions apply to the circuit of $C_{\text{Transfer1}}$.
  1) cm_i^{old} =COMM($pk_{2-PC}, pk_i, v_i, sn_i^{pc}, sn_i^{old}, Y$)
  2) sn_i^{old} =PRF(pk_i, r_i^{old})
  3) cm_i^{new} =COMM($pk_i^{new}, v_i, r_i^{new}, sn_i^{new}$)
  4) sn_i^{new} =PRF(sk_i^{new}, r_i^{new})
  5) The path $path_i$ from cm_i^{old} to the rt saved on the ledger is correct
- Discrete logarithm Y and secret value $y : Y = g^y$(g is a public parameter)
- Signature $\sigma_{\text{Transfer1}}$: Signature of the above ($\pi_{\text{Transfer1}}$, rt,cm_i^{new}, sn_i^{old}) by the payment channel private key sk_i.

For user P_j, the transfer algorithm can be executed to generate the transfer transaction $tx_{\text{Transfer2}}$ to transfer the currency to the secure address as soon as the payment channel update transaction under the version of P_i takes effect, but unlike $tx_{\text{Transfer1}}$, the ZKP does not need to provide the discrete logarithmic value Y and the corresponding secret value y. The transfer operations performed by user P_j and user P_i after the release of the payment channel update transaction under user P_j's version of the transaction are similar, and only require interchanging the operations between P_i and P_j as described above.

4.6 Security of CAPE

Referring to the security models defined by Zerocash, Bolt, and Blockmaze [25], we have defined two security attributes of CAPE: Transaction Unlinkability and Balance. Refer to Appendix A for the formal security definitions.

Definition 1. *(*Security*). A CAPE scheme* Π *= (Setup,Mint, Payment Channel Setup, Payment Channel Update, Payment Channel Closure) is considered secure if it satisfies transaction unlinkability and balance.*

1) $Transaction\ Unlinkability$. Transaction is unlinkable if it does not leak the linkage between its sender and recipient during fund transfers. If for all PPT adversaries $\mathcal{A}$ and sufficiently large security parameter λ, there exists a negligible function $negl(\lambda)$ such that the following probability holds: $\Pr\left[\text{CAPE}_{\Pi,\mathcal{A}}^{\text{TR-UL}}(\lambda) = 1\right] \leq negl(\lambda)$, then we say that Π is TR-UL secure.

2) $Balance$. This property requires that no bounded adversary $\mathcal{A}$ can own more money than he minted or received via payments from others. If for all PPT adversaries $\mathcal{A}$ and sufficiently large security parameter λ, there exists a negligible function $negl(\lambda)$ such that the following probability holds: $\Pr\left[\text{CAPE}_{\Pi,\mathcal{A}}^{\text{BAL}}(\lambda) = 1\right] \leq negl(\lambda)$, then we say that Π is BAL secure.

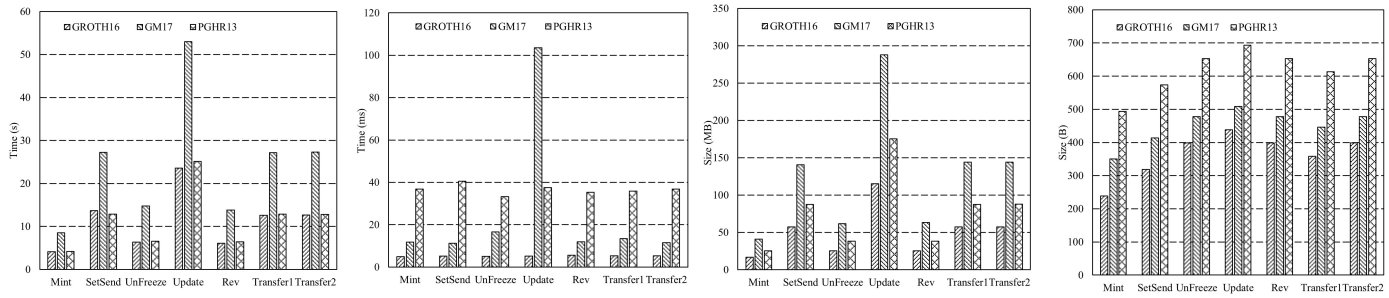
Theorem 1. *The scheme* Π *= (Setup, Mint, Payment Channel Setup, Payment Channel Update, and Payment Channel Closure) is a secure commitment-based anonymous payment channel scheme. (The proof is given in Appendix B)*

5 Experiment and Results

5.1 Experiment Configuration

To benchmark the protocol's performance, we implemented our protocol using the C++ programming language, the Libsnark library, which is a library that implements the zk-SNARKs scheme in C++, and the GMP library.

Cryptographic Libraries: In order to achieve zk-SNARK operations in the CAPE scheme, we have implemented functions for constructing and verifying zk-SNARK proofs based on the Libsnark library [26] . For zk-SNARKs implemented using the Libsnark library, we adopt ALT_BN128 as the default elliptic curve and support different ZKP schemes, including Groth16 [27] , GM17 [28] , and PGHR13 [29] schemes. The key pair (pk, vk) used for zk-SNARKs proof generation or verification is generated and pre-installed on each blockchain node. Additionally, the Merkle tree construction, commitment generation hash function COMM, Hash, and pseudo-random function PRF used in our implementation are all instantiated using the SHA-256 hash function. To implement BLS signatures, this paper develops the BLS signature algorithm based on the GMP library.



(a) Proof generation time (b) Proof verification time (c) Proving key size (d) Verification key size

Fig. 2: The performance of zk-SNARKs used in CAPE.

All experiments were performed on a Ubuntu Linux 16.04 LTS machine equipped with an Intel(R) Core(TM) i7-9700 CPU @3.00GHz CPU and 16GB RAM.

5.2 Experiment Results

5.2.1 zk-SNARKs Performance Evaluation

To determine which zk-SNARK scheme is suitable for CAPE, this paper evaluates the performance of PGHR13, Groth16, and GM17 schemes regarding computation and storage. Fig.2 show performance comparisons of the three schemes, namely PGHR13, Groth16, and GM17, for seven circuits: Mint, SetSend, UnFreeze, Update, Rev, Transfer1, and Transfer2. These comparisons are based on various aspects such as proof/verification key size, proof generation time, and verification time. Fig.2 show that Groth16 has significant advantages over the two schemes PGHR13 and GM17, especially in proofing verification time, and proof/verification key size. In the Mint circuit, the proof verification time of GM17 is 2.4 times that of Groth16, and the proof verification time of PGHR13 is 7.5 times that of Groth16. In the Update circuit, the proof key size of GM17 is 2.5 times that of Groth16, and the proof key size of PGHR13 is 1.5 times that of Groth16. In the Update circuit, the verification key size of GM17 is 1.16 times that of Groth16, and the verification key size of PGHR13 is 1.5 times that of Groth16. In the Update circuit, the proof generation time is similar between Groth16 and PGHR13, while the proof generation time of GM17 is 2.2 times that of Groth16. Taken together, it is evident that the Groth16 solution is the most appropriate for calculating time and space occupation. Therefore, subsequent experiments in this paper use the Groth16 scheme.

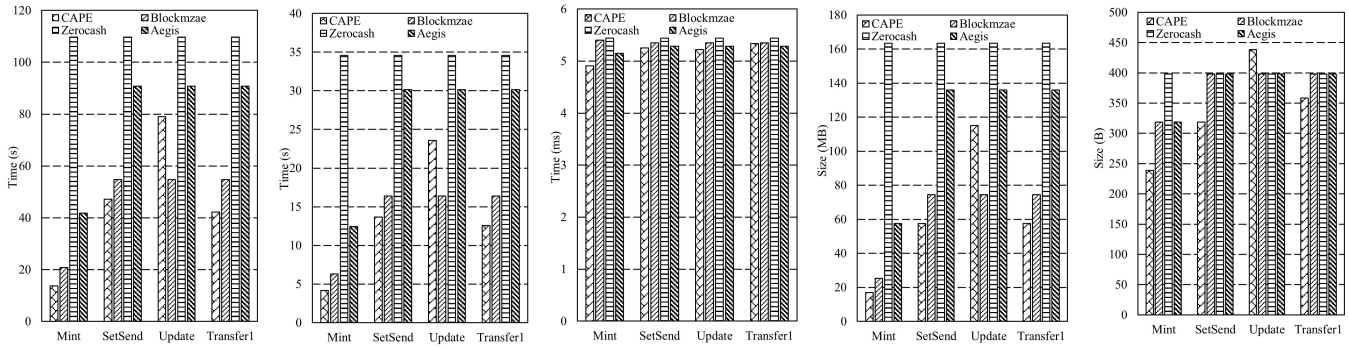
By comparing the experimental results evaluation of different zk-SNARKs circuits under the Groth16 scheme, we can find that the setup time consumption, proof generation time consumption, and the size of the proof or verification key are related to the complexity of the zk-SNARKs circuit relatively. For example, the Update circuit in the payment channel update process consists of eight SHA-256 gate circuits, two Merkle commitment gate circuits, and addition and equal gate circuits. This circuit is the most complex, so that it can be used in the setup and proof generation times. The cost is the largest in terms of the proof or verification key size. In comparisons, the Mint circuit is the simplest, consisting of only two SHA-256 gate circuits. Therefore, the consumption of time and space is the lowest. However, compared with other aspects of zero-knowledge-proof performance, the Setup has the lowest impact on the solution's performance because it can be set in advance. Compared with the above four aspects, the verification time consumption of ZKP is the same for different zk-SNARKs circuits, which is about 5ms. Therefore, it has less impact on on-chain verification.

5.2.2 Comparisons on Blockmaze, Zerocash and Aegis

Fig. 3 shows the experiment results that compare our scheme with other privacy protection schemes in terms of setup time, proof or verification key size, proof generation time, and verification time using zk-SNARKs. Although Blockmaze, Zerocash, and Aegis [30] have different use cases from this scheme, they all adopt a similar method, zk-SNARKs, to achieve privacy protection. From the perspective of algorithmic functionality, the Mint operation in the CAPE scheme is equivalent to Mint in Blockmaze. The CAPE scheme's SetSend, Update, and Transfer1 operations are similar to Deposit in Blockmaze. The Mint, SetSend, Update, and Transfer1 operations in the CAPE scheme are similar to Pour in Zerocash. The Mint operation in the CAPE scheme is similar to Ownership in Aegis. Operations SetSend, Update, and Transfer1 in the CAPE scheme are similar to JoinSplit in Aegis.

Compare to the schemes mentioned above, our scheme has advantages over Blockmaze in terms of setup time, proof or verification key size, and proof generation time, specifically in the Mint operation. In the comparison process of SetSend, Update, Transfer1, and Deposit, only Update and Deposit perform inadequately in the setup time, proof or verification key size, and proof generation time. However, since Update involves multiple off-chain transactions before being included on-chain, it still possesses advantages in practical applications. Unlike Zerocash, the Mint operation in this paper requires zero-knowledge operations.

Moreover, our scheme has advantages in SetSend, Update, and Transfer1 operations over Pour in Zerocash regarding computation time and storage cost, especially the Update circuit. On one hand, the proof generation time for the Pour circuit is 1.4 times more than the Update circuit. On the other hand, Update can be executed off-chain without requiring on-chain consensus, making it more efficient for completing a transaction.



(a) Setup time (b) Proof generation time (c) Proof verification time (d) Proving key size (e) Verification key size

Fig. 3: Comparisons of Blockmaze, Zerocash and Aegis.

Compare to Aegis, the Mint circuit in this paper, corresponding to the recharge operation, our scheme only takes 4.1 seconds to generate a proof. The Ownership circuit in Aegis for the recharge operation takes 12.4 seconds. As such, Aegis is more time-consuming. This paper's computation and storage costs of the SetSend, Update, and Transfer1 circuits are lower than JoinSplit Aegis for the transfer operation between accounts. As for the verification time for the proof, the CAPE scheme in this paper performs similarly to the other three schemes, requiring only around 5ms to complete.

In summary, CAPE shows a superior performance regarding the computational costs associated with ZKPs, showcasing good computational efficiency and storage costs, comparing to other schemes.

5.2.3 Evaluation of Payment Channel

So far we have evaluated the on-chain storage overhead and off-chain communication overhead, for the three phases of the payment channel, we evaluate the message complexity within the channel at each phase, including the number of on-chain transactions and off-chain messages.

The Number of On-chain Transactions: establishing a payment channel requires transaction parties i and j to post one transaction each on the blockchain. While updating the payment channel, both parties do not need to make on-chain transactions. When the payment channel and both parties' honest collaboration are closed, three on-chain transactions are required; otherwise, three transactions are also required when there is a dispute.

The Number of Off-chain Messages: when a payment channel is established, two transactions and two VTD commitments must be exchanged between trading parties. In updating a payment channel, two transactions and two VTD commitments are exchanged between the two parties for each update. When the payment channel is closed, no off-chain message exchange is required. According to the experiment results given in in Table 1, to open a payment channel in the CAPE scheme, both parties must place 445.78 bytes (2 transactions) on the chain and exchange 10.24 KB off-chain. During the process of payment channel update, for each update transaction, both parties need to exchange 11.39 KB off-chain. When the payment channel is closed and both parties are closed honestly, 1791.31 bytes (three transactions) must be placed on the chain; when one party is dishonest, 1794.67 (three transactions) must be placed on the chain.

Comparison with Light and Sleepy. Table 2 shows a comparison of the on-chain storage overhead and off-chain communication overhead during payment channel establishment and payment channel update between the present solution and other existing solutions. Although the present solution has increased the associated privacy protection measures compared to Light and Sleepy, the on-chain storage overhead and off-chain communication overhead during the payment channel establishment and payment channel update process are still within an acceptable range. During the payment channel establishment process, the transaction size to be stored on the chain in this scheme is only 1.3 times that of the Light scheme and 1.3 times that of the Sleepy scheme, and the off-chain communication overhead of this scheme is 12 times that of the Light and 5 times that of the Sleepy, respectively. During the payment channel update process, the communication overhead of this scheme under the chain is 9.6 times that of Light and 4.8 times that of Sleepy, respectively.

Computation Overhead. For the three phases of the payment channel, we evaluate the consumption of in-channel time in each phase (refer to Table 3). In the CAPE scenario, a non-trivial time cost is required at each stage due to the presence of ZKPs and VTD commitments. We ignore the communication overhead for our time-specific measurements. According to our measurements, the time spent on a single update during the payment channel update is 30.1s, comparing to 6.6 TPS (Transaction Per Second) for the same type of zcash blocking transaction, the throughput of transaction per second can be improved to 0.033D TPS when the scheme in this paper is adopted, where D is the number of channels opened on the blockchain. When the number of payment channels opened on the blockchain is 10000, this scheme can provide 33 TPS transaction throughput per second.

Overall, although our proposal incorporates more privacy protection measures, the on-chain storage overhead and off-chain communication costs during the establishment and updating phases of the payment channel remain within acceptable ranges. Specifically, during the payment channel establishment phase, the on-chain storage overhead

TABLE 1: On-chain Storage Overhead and Off-chain Communication Overhead

	Off-chain		On-chain	
Create	$tx^i_{\text{UnFreeze}}, tx^j_{\text{UnFreeze}}, (C^i_{\text{VTD}}, \pi^i_{\text{VTD}}), (C^j_{\text{VTD}}, \pi^j_{\text{VTD}})$	10.24 KB	$tx^i_{\text{SetSend}}, tx^j_{\text{SetSend}}$	445.78 B
Update	$tx^i_{\text{Update}}, tx^j_{\text{Update}}, tx^i_{\text{Rev}}, tx^j_{\text{Rev}}, (C^i_{\text{VTD}}, \pi^i_{\text{VTD}}), (C^j_{\text{VTD}}, \pi^j_{\text{VTD}})$	11.39 KB	-	-
Close(optimistic)	-	-	$tx^i_{\text{Update}}, tx^i_{\text{Transfer1}}, tx^j_{\text{Transfer2}}$	1791.32 B
Close(pessimistic)	-	-	$tx^i_{\text{Update}}, tx_{\text{Rev}}, tx^j_{\text{Transfer2}}$	1794.67 B

TABLE 2: Comparison of Off-chain Communication Overhead with Light and Sleepy

	Channel opening (on-chain)	Channel opening (off-chain)	Channel update
CAPE	445.78 B	10485 B	11663 B
Sleepy	338 B	2026 B	2408 B
Light	338 B	832 B	1214 B

TABLE 3: Computational Overhead

	Joint key	Channel opening	Channel update
i(s)	0.017	21.87	30.12
j(s)	0.017	21.85	30.11

of our approach is slightly higher than other solutions, but the off-chain communication costs remain manageable. In the updating phase of the payment channel, the off-chain communication costs of our approach are also within acceptable ranges compared to other solutions. Additionally, while our proposal incurs some time costs, with an increase in the number of channels, the throughput of transactions per second experiences a significant improvement, demonstrating good scalability and performance advantages. Therefore, experimental results indicate that although our proposed solution incurs higher communication and computational costs compared to traditional payment channel schemes, its performance remains feasible and practical for real-world applications.

6 Related Work

6.1 Payment Channel

Hearn and Spilman [31] initially introduced the payment channel, which built a channel micropayment protocol based on the Bitcoin system. The protocol allows direct Bitcoin transactions between the sender and the receiver in an off-chain channel. However, this scheme still had the constraint of supporting only one-way payments. [32] introduced a two-way micropayment channel scheme via Duplex. However, the Duplex also had a drawback in enlarging the number of payments deriving from the limited capability; whereby the life-cycle of the payment channel was shortened when the number of off-chain payments increased. Poon *et al.* [17] introduced the most popular two-way payment channel scheme through the lightning network. In the lightning network, the primary measures of a two-way payment channel scheme to ensure secure off-chain transactions between the two parties of the transaction were the time lock mechanism and the penalty transaction mechanism. However, establishing a payment channel in a lightning network requires a load of complex operations, which could be less favourable to its rapid development. A new payment channel protocol, xLumi [33], has been created on the basis of blockchain systems. This protocol differed from the Lightning Network in that it significantly reduces the number of interactions and the complexity of opening payment channels during off-chain payments. However, xLumi only created one-way payment channels and does not extend to two-way payment channels.

Furthermore, Lind [34] introduced a new payment channel framework via Teechan, which enabled off-chain micropayments between two parties to a transaction on the existing Bitcoin blockchain. Similar to Duplex and Lightning Network schemes, Teechan used multiple signatures to establish a long-term payment channel between two mutually untrusted parties. In Teechan, payments were made through a single message and can be made simultaneously in both directions, making it a two-way payment channel instead of a one-way one.

6.2 Privacy-Preserving Payment Channel

Although the emergence of payment channels had improved the scalability of blockchain systems, there were still many areas for improvement in protecting privacy. Green and Miers [19] proposed a chain-off payment channel scheme (called Bolt) that provided completely anonymous transactions in which the parties can achieve privacy protection with variable bidirectional payments. By this scheme, transaction data privacy was protected by hiding the transaction amount, the privacy of the relationship between transaction parties was protected using ZKPs, and blind signatures are used to verify channel updates. Compared to our scheme, the use of blind signatures in Bolt was incompatible with Bitcoin, whereas the ZKPs and time-lock verifiable commitments adopted in this paper can be applied to various blockchains. Additionally, the expenses of ZKPs required by Bolt were higher than our approach. Zhang *et al.* [35] proposed a payment channel called Z-Channel to protect privacy, which also used ZKPs to protect user privacy but avoids the problems caused by blind signatures. Compared to our scheme, Z-channel employed the relatively common technique of relative timelocks used in most payment channels, which limited its use in non-scripted blockchains.

In addition, Moreno-Sanchez [36] proposed a payment channel protocol, called DLSAG for Monero, which could

achieve payment channel privacy protection through off-chain transactions on Moreno. However, their solution was not backwards compatible and required great modifications to the Monero transaction scheme. Thyagarajan *et al.* [20] proposed a new payment channel protocol called PayMo for Monero, which did not require any changes to the Monero transaction scheme or adding any features to the script language. Compared to our scheme, both PayMo and DLSAG were tailored proposals specifically designed for Monero, and they only supported one-way payments, making them unsuitable for further expansion into bidirectional payments. Guan *et al.* [37] proposed a payment channel scheme based on Mimblewimble, which tool advantage of the ability of Mimblewimble to hide user identities and transaction amounts to achieve privacy protection for payment transactions. Compared to our scheme, it was tailored for Mimblewimble and only supported one-way payments.

7 Conclusions

This paper proposed a scheme, named CAPE, which utilized commitment schemes, zk-SNARKs, and VTD commitments to achieve privacy protection for users in payment channel transactions. Our scheme concealed the transaction amounts in both on-chain and off-chain transactions within the payment channel, as well as the linking relationships between participants, while ensuring fairness in the transaction process. We provided the specific construction of our CAPE scheme and formal security proofs. Finally, this work evaluated the performance of the CAPE scheme to demonstrate its potentials for deployment in real-world environments. Compared to other privacy-preserving schemes of the same type, this paper incurs less computational and storage costs to achieve privacy. Compared to other payment channel schemes, the various increases in overheads to achieve privacy in this paper are within acceptable limits.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, vol. PP, no. 99, p. 21260, 2008.

[2] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, *et al.*, "On scaling decentralized blockchains: (a position paper)," in *FC Workshops, BITCOIN, VOTING, and WAHC*, (Christ Church, Barbado), pp. 106–125, 2016.

[3] A. Biryukov and S. Tikhomirov, "Deanonymization and linkability of cryptocurrency transactions based on network analysis," in *2019 IEEE European symposium on security and privacy (EuroS&P)*, (Stockholm, Sweden), pp. 172–184, 2019.

[4] M. Trillo, "Stress test prepares visanet for the most wonderful time of the year (2013)," *URL http://www. visa. com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index. html*, 2013.

[5] J. Bonneau, I. Meckler, V. Rao, and E. Shapiro, "Coda: Decentralized cryptocurrency at scale," *Cryptology ePrint Archive*, vol. PP, no. 99, p. 1, 2020.

[6] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *26th SOSP*, (New York), pp. 225–235, 2017.

[7] P. Zheng, Z. Jiang, J. Wu, and Z. Zheng, "Blockchain-based decentralized application: A survey," *IEEE Open Journal of the Computer Society*, vol. PP, no. 99, p. 1, 2023.

[8] N. Papadis and L. Tassiulas, "Blockchain-based payment channel networks: Challenges and recent advances," *IEEE Access*, vol. 8, no. 99, pp. 227596–227609, 2020.

[9] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, *et al.*, "A survey on the scalability of blockchain systems," *IEEE Network*, vol. 33, no. 5, pp. 166–173, 2019.

[10] G. Kaur and C. Gandhi, "Scalability in blockchain: Challenges and solutions," in *Handbook of Research on Blockchain Technology*, pp. 373–406, 2020.

[11] D. Khan, L. T. Jung, and M. A. Hashmani, "Systematic literature review of challenges in blockchain scalability," *Applied Sciences*, vol. 11, no. 20, p. 9372, 2021.

[12] J. Xu, C. Wang, and X. Jia, "A survey of blockchain consensus protocols," *ACM Computing Surveys*, vol. PP, no. 99, p. 1, 2023.

[13] K. Gai, Z. Hu, L. Zhu, R. Wang, and Z. Zhang, "Blockchain meets DAG: a blockdag consensus mechanism," in *20th ICA3PP*, (New York City, NY, USA), pp. 110–125, 2020.

[14] V. Buterin *et al.*, "Ethereum white paper: a next generation smart contract & decentralized application platform," *First version*, vol. 53, no. 99, p. 1, 2014.

[15] F. Armknecht, G. O. Karame, A. Mandal, F. Youssef, and E. Zenner, "Ripple: Overview and outlook," in *8th Int'l Conf. on TRUST, Heraklion*, (Heraklion, Greece), pp. 163–180, 2015.

[16] Y. Zhang, K. Gai, J. Xiao, L. Zhu, and K.-K. R. Choo, "Blockchain-empowered efficient data sharing in internet of things settings," *IEEE J. on Selected Areas in Communi.*, vol. 40, no. 12, pp. 3422–3436, 2022.

[17] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," *White Paper*, vol. PP, no. 99, p. 1, 2016.

[18] S. Liu and J. Wang, "Dmc: Decentralized mixer with channel for transaction privacy protection on ethereum," in *CS & IT Conference Proceedings*, vol. 11, 2021.

[19] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *ACM SIGSAC Conf. on Computer and Communications Security*, (Dallas Texas, USA), pp. 473–489, 2017.

[20] S. A. Thyagarajan, G. Malavolta, F. Schmidt, and D. Schröder, "Paymo: Payment channels for monero," *Cryptology ePrint Archive*, vol. PP, no. 99, p. 1, 2020.

[21] O. Goldreich and Y. Oren, "Definitions and properties of zero-knowledge proof systems," *Journal of Cryptology*, vol. 7, no. 1, pp. 1–32, 1994.

[22] S. A. K. Thyagarajan, A. Bhat, G. Malavolta, N. Döttling, A. Kate, *et al.*, "Verifiable timed signatures made practical," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1733–1750, 2020.

[23] S. A. Thyagarajan, G. Malavolta, and P. Moreno-Sanchez, "Universal atomic swaps: Secure exchange of coins across all blockchains," in *IEEE Symposium on Security and Privacy (SP)*, (San Francisco, CA, USA), pp. 1299–1316, 2022.

[24] G. Malavolta and S. A. K. Thyagarajan, "Homomorphic time-lock puzzles and applications," in *Annual International Cryptology Conference*, (Santa Barbara, CA, USA), pp. 620–649, 2019.

[25] Z. Guan, Z. Wan, Y. Yang, Y. Zhou, and B. Huang, "Blockmaze: An efficient privacy-preserving account-model blockchain based on zk-snarks," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 3, pp. 1446–1463, 2020.

[26] E. Ben-Saason, A. Chiesa, D. Genkin, S. Kfir, E. Tromer, *et al.*, "libsnark: C++ library for zksnark proofs," 2014.

[27] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, (Vienna, Austria), pp. 305–326, 2016.

[28] J. Groth and M. Maller, "Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks," in *Annual Int'l Cryptology Conf.*, (Santa Barbara, CA, USA), pp. 581–612, 2017.

[29] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," *Communications of the ACM*, vol. 59, no. 2, pp. 103–112, 2016.

[30] H. S. Galal and A. M. Youssef, "Aegis: Privacy-preserving market for non-fungible tokens," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 1, pp. 92–102, 2022.

[31] M. Hearn and J. Spilman, "Bitcoin contracts," *URL: https://en. bitcoin. it/% 20wiki/Contracts (visited on 2015-10-08)*, 2015.

[32] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Stabilization, Safety, and Security of Distributed Systems: 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings 17*, (Edmonton, AB, Canada), pp. 3–18, 2015.

[33] N. Ying and T. W. Wu, "xlumi: Payment channel protocol and off-chain payment in blockchain contract systems," *arXiv preprint arXiv:2101.10621*, vol. PP, no. 99, p. 1, 2021.

[34] J. Lind, I. Eyal, P. Pietzuch, and E. G. Sirer, "Teechan: Payment channels using trusted execution environments," *arXiv preprint arXiv:1612.07766*, vol. PP, no. 99, p. 1, 2016.
[35] Y. Zhang, Y. Long, Z. Liu, Z. Liu, and D. Gu, "Z-channel: Scalable and efficient scheme in zerocash," *Computers & Security*, vol. 86, no. 99, pp. 112–131, 2019.
[36] P. Moreno-Sanchez, A. Blue, D. V. Le, S. Noether, B. Goodell, *et al.*, "Dlsag: non-interactive refund transactions for interoperable payment channels in monero," in *24th Int'l Conf. Financial Cryptography and Data Security*, (Kota Kinabalu, Malaysia), pp. 325–345, 2020.
[37] Z. Guan, L. Zhang, Y. Zhang, D. Li, Y. Sun, *et al.*, "Off-chain anonymous payment channel scheme based on mimblewimble," in *IEEE Int'l Conf on ISPA/BDCloud/SocialCom/SustainCom*, (New York), pp. 1469–1475, 2021.

APPENDIX A

A CAPE scheme $\Pi =$ (Initialization, Coin Minting, Payment Channel Establishment, Payment Channel Update, and Payment Channel Closure) is considered secure if it satisfies transaction unlinkability and balance preservation. Referring to the security models defined by Zerocash, Bolt, and Blockmaze, each property is formalized as a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. Each property depends on an experiment involving a finite-state oracle $\mathcal{O}^{\mathrm{CAPE}}$ that interacts with CAPE. In each experiment, the behavior of the honest party is realized by a random oracle $\mathcal{O}^{\mathrm{CAPE}}$.

The random oracle $\mathcal{O}^{\mathrm{CAPE}}$ provides an interface for executing the algorithms defined in the CAPE scheme Π. It is initialized and maintains a state with a set of public parameters pp. Internally, $\mathcal{O}^{\mathrm{CAPE}}$ stores and maintains the following: (i) L, a ledger recording transactions. (ii) $Addr$, a collection of address-secret pairs. (iii) $COIN$, a set of coins.

All of L, $Addr$, and $COIN$ start as empty. $\mathcal{O}^{\mathrm{CAPE}}$ accepts different types of queries, and each query results in updates to L, $Addr$, $COIN$, and the output. Below, we will describe these queries to the random oracle $\mathcal{O}^{\mathrm{CAPE}}$.

- $Query(CreateAccount)$.When receiving the $Create\text{-}Account$query, the challenger $\mathcal{C}$: (i) Computes a key pair (sk, pk). (ii) Adds pk as an address to $Addr$. (iii) Outputs (pk).
- $Query(Mint, pk_i, v)$.When receiving the $Mint$ query, the challenger $\mathcal{C}$: (i) Invokes the $Mint$ algorithm to compute a coin c and a $Mint$ transaction tx_{Mint}. (ii) Adds the coin c to $COIN$ and tx_{Mint} to the ledger L. (iii) Outputs $\perp$.
- $Query(SetSend, cm_i, pk_i, pk_{2-PC})$. When receiving the $SetSend$ query, the challenger $\mathcal{C}$: (i) Invokes the payment channel establishment and sending algorithm to compute a coin c and a $SetSend$ transaction tx_{SetSend}. (ii) Adds the coin c to $COIN$ and tx_{SetSend} to the ledger L. (iii) Outputs $\perp$.
- $Query(UnFreeze, cm, pk_{2-PC}, pk_i)$. When receiving the $UnFreeze$ query, the challenger $\mathcal{C}$: (i) Invokes the payment channel unfreezes algorithm to compute a coin c and an $UnFreeze$ transaction tx_{UnFreeze}. (ii) Adds the coin c to $COIN$ and tx_{UnFreeze} to the ledger L. (iii) Outputs $\perp$.
- $Query(Update, pk_{2-PC}, pk_i, pk_j, cm_i, cm_j, v_i, v_j)$. Wh-en receiving the $Update$ query, the challenger $\mathcal{C}$: (i) Calls the payment channel update generation algorithm to compute two new coins c_i^{new} and c_j^{new}, and an $Update$ transaction tx_{Update}. (ii) Adds the coins c_i^{new} and c_j^{new} to $COIN$, and adds tx_{Update} to the ledger L. (iii) Outputs $\perp$.
- $Query(Rev, pk_{2-PC}, pk_i, pk_j, cm)$: When receiving the Rev query, the challenger $\mathcal{C}$. (i) Invokes the payment channel rollback algorithm to compute a coin c and a Rev transaction tx_{Rev}. (ii) Adds the coin c to $COIN$ and tx_{Rev} to the ledger L. (iii) Outputs $\perp$.
- $Query(Transfer1, pk_i, v)$. When receiving the $Trans\text{-}\ fer1$ query, the challenger $\mathcal{C}$. (i) Invokes the corresponding transfer algorithm for $Transfer1$ to compute a coin c and a $Transfer1$ transaction $tx_{\text{Transfer1}}$. (ii) Adds the coin c to $COIN$ and $tx_{\text{Transfer1}}$ to the ledger L. (iii) Outputs $\perp$.
- $Query(Transfer2, pk_i, v)$. When receiving the $Trans\text{-}fer2$ query, the challenger $\mathcal{C}$. (i) Invokes the corresponding transfer algorithm for $Transfer2$ to compute a coin c and a $Transfer2$ transaction $tx_{\text{Transfer2}}$. (ii) Adds the coin c to $COIN$ and $tx_{\text{Transfer2}}$ to the ledger L. (iii) Outputs $\perp$.
- $Query(Insert, tx)$. When receiving the $Insert$ query, the challenger $\mathcal{C}$ verifies the result by executing different algorithms. If the condition is met, the output result is 1, and the transaction tx is added to L. Otherwise, it aborts.

A.1 Transaction Unlinkability

Let $\mathcal{T}$ be a zero-knowledge transaction table, where tx_{SetSend}, tx_{Update}, tx_{Rev}, tx_{UnFreeze}, $tx_{\text{Transfer1}}$, and $tx_{\text{Transfer2}}$ are generated in response to $SetSend$, $Update$, Rev, $UnFreeze$, $Transfer1$, and $Transfer2$ queries, respectively, in the context of the CAPE protocol. The design of the transaction unlinkability experiment $\mathrm{CAPE}^{\text{TR-UL}}_{\Pi,\mathcal{A}}(\lambda)$ is as follows:

1) Compute $pp := \mathrm{Setup}\left(1^{\lambda}\right)$ and send it to $\mathcal{A}$. Initialize a CAPE random oracle $\mathcal{O}^{\mathrm{CAPE}}$.
2) Whenever $\mathcal{A}$ queries $\mathcal{O}^{\mathrm{CAPE}}$, provide the ledger L in the response to each query operation.
3) Continue responding to queries from $\mathcal{A}$ until $\mathcal{A}$ outputs a pair of zero-knowledge transactions (tx, tx') that satisfy the following conditions: (i) $(tx, tx')\in\mathcal{T}$ and (tx, tx') are transactions of the same type, but $tx \neq tx'$. (ii) If tx=tx_{Setsend}, the sender and receiver of (tx, tx') are not $\mathcal{A}$. (iii) If tx=tx_{Update}, the sender and receiver of (tx, tx') are not $\mathcal{A}$. (iv) If tx=$tx_{\text{Transfer1}}$, the sender and receiver of (tx, tx') are not $\mathcal{A}$. (v) If tx=$tx_{\text{Transfer2}}$, the sender and receiver of (tx, tx') are not $\mathcal{A}$. (vi) If tx=tx_{UnFreeze}, the sender and receiver of (tx, tx') are not $\mathcal{A}$. (vii) If tx=tx_{Rev},(tx, tx'), the sender and receiver of (tx, tx') are not $\mathcal{A}$
4) If any of the following conditions hold, output 1 (indicating that $\mathcal{A}$ wins); otherwise, output 0: If tx=tx_{Setsend}, the sender and receiver of (tx,tx') are the same. If tx=tx_{Update}, the receiver of (tx,tx') is the same. If tx=$tx_{\text{Transfer1}}$, the receiver of (tx,tx') is the same. If tx=$tx_{\text{Transfer2}}$, the receiver of (tx,tx') is the same. If tx=tx_{UnFreeze}, the receiver of (tx,tx') is the same. If tx=tx_{Rev}, the receiver of (tx,tx') is the same.

Definition 2 (TR-UL Security): Let $\Pi =$(System Setup, User Initialization, Payment Channel Establishment, Payment Channel Update, and Payment Channel Closure) be a CAPE scheme. If for all probabilistic polynomial-time adversaries $\mathcal{A}$ and sufficiently large security parameter λ, there exists a negligible function $\mathrm{neg}(\lambda)$ such that the following probability holds: $\Pr\left[\mathrm{CAPE}^{\mathrm{TR-UL}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \leq \mathrm{neg}(\lambda)$, then we say that $\Pi =$ is TR-UL-secure.

A.2 Balance Conservation

For balance conservation, we design the BAL experiment where a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ attempts to attack a given CAPE scheme. First, we define four variables in our security model as follows:

- v_{Unspent} : The total value of all spendable coins in the set $\mathcal{S}_{\text{coin}}$ owned by $\mathcal{A}$. The challenger $\mathcal{C}$ can check whether a coin C owned by $\mathcal{A}$ is spendable by: 1) Checking if a valid tx_{Setsend} transaction can be generated through a Setsend query for coins generated by a tx_{Mint} transaction. 2) Checking if a valid $tx_{\text{Transfer1}}$ transaction can be generated through a Transfer1 query or if a valid $tx_{\text{Transfer2}}$ transaction can be generated through a Transfer2 query for coins generated by tx_{SetSend}, tx_{Update}, $tx_{\text{Transfer1}}$, $tx_{\text{Transfer2}}$, tx_{UnFreeze}, and tx_{Rev} transactions, depending on whether the coin commitment includes a discrete logarithm value.
- v_{Mint} is the total value of all coins minted by $\mathcal{A}$. To compute v_{Mint}, the challenger $\mathcal{C}$ sums up the values of all coins generated by Mint queries using addresses not in Addr, or by directly inserting Mint transactions into the ledger
- $v_{\mathcal{A}\leftarrow\text{Addr}}$ is the total amount paid to $\mathcal{A}$ from addresses in Addr. To compute $v_{\mathcal{A}\leftarrow\text{Addr}}$, the challenger $\mathcal{C}$ uses Transfer2, Transfer1, Setsend, Update, UnFreeze, and Rev queries to look up corresponding transactions in the ledger and aggregates the values transferred to addresses not in Addr.
- $v_{\text{Addr}\leftarrow\mathcal{A}}$ is the total amount paid by $\mathcal{A}$ to addresses in Addr. To compute $v_{\text{Addr}\leftarrow\mathcal{A}}$, the challenger $\mathcal{C}$ first identifies the set $\mathcal{S}$ of all coins sent to parties in Addr through Insert queries, and then sums up the values of the coins in $\mathcal{S}$.

At the end of the experiment, if...$v_{Unspent} + v_{Addr\leftarrow\mathcal{A}} > v_{Mint} + v_{\mathcal{A}\leftarrow Addr}$,$\mathcal{C}$ outputs 1; otherwise, it outputs 0.

Formally, we define the balance conservation experiment $\text{CAPE}^{\text{BAL}}_{\Pi,\mathcal{A}}(\lambda)$ as follows:

- Compute $\text{pp} := \text{Setup}\left(1^{\lambda}\right)$ and send it to $\mathcal{A}$. Initialize a CAPE random oracle $\mathcal{O}^{\text{CAPE}}$.
- Whenever $\mathcal{A}$ queries $\mathcal{O}^{\text{CAPE}}$, provide the ledger L in response to each query operation.
- Continue responding to queries from $\mathcal{A}$ until $\mathcal{A}$ outputs a set of coins $\mathcal{S}_{\text{coin}}$.
- Compute the aforementioned four balance variables.
- If $v_{Unspent} + v_{Addr\leftarrow\mathcal{A}} > v_{Mint} + v_{\mathcal{A}\leftarrow Addr}$, the experiment outputs 1; otherwise, it outputs 0.

Definition 3. (BAL security): Let Π = (setup, user initialization, payment channel establishment, payment channel update, and payment channel closure) be a CAPE scheme. If for all PPT adversaries $\mathcal{A}$ and sufficiently large security parameter λ, the following probability holds: $\Pr\left[\text{CAPE}^{\text{BAL}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \leq \text{negl}(\lambda)$, then we say that Π is BAL-secure.

APPENDIX B

Theorem 1. The scheme Π = (initialization, coin minting, payment channel establishment, payment channel update, and payment channel closure) is a secure commitment-based anonymous payment channel scheme.

B.1 Transaction Non-Linkability Proof

Let $\mathcal{T}$ be the zero-knowledge transaction table, where tx_{SetSend}、tx_{Update}、tx_{Rev}、tx_{UnFreeze}、$tx_{\text{Transfer1}}$, and $tx_{\text{Transfer2}}$ are generated by $\mathcal{O}^{\text{CAPE}}$ in response to $SetSend$, $Update$, Rev, $UnFreeze$, $Transfer1$, and$Transfer2$ queries, respectively. Adversary $\mathcal{A}$ can win the TR-UL experiment only if $\mathcal{A}$ outputs a pair of zero-knowledge transactions (tx, tx') satisfying one of the following conditions: (i) If tx=tx_{Setsend}, both the sender and receiver of (tx, tx') are the same. (ii) If tx=tx_{Update}, both the receivers of (tx,tx') are the same. (iii) If tx=$tx_{\text{Transfer1}}$, both the receivers of (tx, tx') are the same. (iv) If tx=$tx_{\text{Transfer2}}$, both the receivers of (tx, tx') are the same. (v) If tx=tx_{UnFreeze}, both the receivers of (tx, tx') are the same. (vi) If tx=tx_{Rev}, both the receivers of (tx, tx') are the same.

a) Assuming $\mathcal{A}$ outputs a tuple of transactions $(tx_{\text{SetSend}}, tx'_{\text{SetSend}})$ for payment channel establishment, where tx_{SetSend} satisfies the following conditions:

$$tx_{\text{SetSend}} := (rt, cm_i^{pc}, sn_i^{old}, \pi_{SetSend})$$
$$cm_i^{pc} = \text{COMM}(pk_{2-PC}, pk_i, v_i, sn_i^{old}, sn_i^{pc})$$
$$cm_i^{old} = \text{COMM}(pk_i, v_i, sn_i^{old}, r_i^{old})$$

tx'_{SetSend} also satisfies:

$$tx'_{\text{SetSend}} := (rt', cm_i^{pc'}, sn_i^{old'}, \pi'_{SetSend})$$
$$cm_i{}^{pc'} = \text{COMM}(pk'_{2-PC}, pk_i', v_i', sn_i^{old'}, sn_i^{pc'})$$
$$cm_i^{old'} = \text{COMM}(pk_i{}', v_i{}', sn_i^{old'}, r_i^{old'})$$

In order to win the TR-UL experiment, one must find a pair of transactions $(tx_{\text{SetSend}}, tx'_{\text{SetSend}})$ where the receiver $pk_{2-PC} = pk'_{2-PC}$ and the sender $pk_i = pk'_i$.

To achieve the goal of finding the same receiver, $\mathcal{A}$ can determine whether pk_{2-PC} and pk'_{2-PC} are equal in two ways: (a) Obtain the receiver address pk_{2-PC} (or pk'_{2-PC}) from cm_i^{pc} or ($cm_i^{pc'}$). (b)Extract pk_{2-PC} (pk'_{2-PC}) from the zk-SNARKs proof π_{SetSend}(或 π'_{SetSend}). For (a), $\mathcal{A}$ must distinguish the included (pk_{2-PC} and pk'_{2-PC}) without knowing the secret values of (cm_i^{pc} and $cm_i^{pc'}$). This would violate the hiding property of $COMM$, making it impractical for $\mathcal{A}$. For (b), $\mathcal{A}$ cannot extract the receiver address from the zero-knowledge proof due to the unbreakable zero-knowledge property of zk-SNARKs mentioned in Chapter 3.

To achieve the goal of finding the same sender, $\mathcal{A}$ can also determine it through three approaches: (a) distinguishing the sender addresses from the zk-SNARKs proofs, (b) $\mathcal{A}$ first looks up the commitment value (cm_i^{old}, $cm_i^{old'}$) used in the SetSend transaction from its view, and then uses the cmassociated $Mint$ transaction to differentiate sender, (c) distinguishing the sender address from cm_i^{pc} or ($cm_i^{pc'}$).

For approach (a), $\mathcal{A}$ must be derived from different zero-knowledge proofs (π_{SetSend},$\pi_{\text{SetSend}}{}'$) will distinguish (pk_i,$pk_i{}'$), which means that $\mathcal{A}$ needs to destroy the zero-knowledge property of zk-SNARKs.

For approach (b), $\mathcal{A}$ can differentiate the sender (pk_i, $pk_i{}'$) by looking for the previous transactions (tx_{Mint}, tx'_{Mint}) that contain (cm_i^{old}, $cm_i^{old'}$), without knowing other secret values in (cm_i^{old}, $cm_i^{old'}$). However, since cm_i^{old} and $cm_i^{old'}$ are not visible in tx_{SetSend} and tx'_{SetSend}, $\mathcal{A}$ must obtain cm_i^{old} and $cm_i^{old'}$ through two means: (1) the Merkle tree root, and (2) zk-SNARKs proof.

For method (1), $\mathcal{A}$ must be proposed from the Merkle root (rt, rt') (cm_i^{old}, $cm_i^{old'}$), and this needs to destroy the

anti-collision property of CRH. For the way (2), $\mathcal{A}$ must be proved from zk-SNARKs (π_{SetSend} , (cm_i^{old}, $cm_i^{old'}$) is extracted from π'_{SetSend}), which means that $\mathcal{A}$ needs to destroy the zero-knowledge property of zk-SNARKs. For method (c), it will violate the hidden characteristics of COMM for the same reason as above. Therefore, due to the commitment scheme, hash function and zk-SNARKs security features, it is guaranteed that the sender and receiver of the transaction cannot be distinguished in the tx_{SetSen} transaction.

b) Suppose $\mathcal{A}$ outputs a payment channel retracement transaction tuple (tx_{Rev} , tx'_{Rev}), where tx_{Rev} meets the following conditions:

$$tx_{Rev} := \left(\pi_{Rev}, cm_j^{\text{old}}, cm_i^{\text{new}}, sn_j^{\text{old}}, \text{Y}, pk_j, \sigma_{Rev}\right)$$
$$cm_j^{\text{old}} = \text{COMM}\left(pk_{2-PC}, pk_j, v_j, sn_j^{pc}, sn_j^{\text{old}}, Y\right)$$
$$cm_i^{\text{new}} = \text{COMM}\left(pk_j, pk_i, v_j, sn_j^{\text{old}}, sn_i^{\text{new}}\right)$$

,tx'_{Rev} is also satisfied:

$$tx'_{\text{Rev}} := \left(\pi'_{\text{Rev}}, cm_j^{old'}, cm_i^{new'}, sn_j^{old'}, \text{Y}', pk'_j, \sigma'_{\text{Rev}}\right)$$
$$cm_j^{\text{old}} = \text{COMM}\left(pk'_{2-PC}, pk'_j, v'_j, sn_j^{pc'}, sn_j^{old'}, Y'\right)$$
$$cm_i^{new\prime} = \text{COMM}(pk'_j, pk'_i, v'_j, sn_j^{old'}, sn_i^{new\prime})$$

To gain the TR-UL experiment must find a set of transaction (tx_{Rev} , tx'_{Rev}) receivers $pk_i = pk_i^{prime}$. In order to achieve the goal of finding the receiver, $\mathcal{A}$ should choose (a) to get the receiver from cm_i^{new} (or $cm_i^{new'}$) Party address pk_i (or pk'_i) , (b) From zk-SNARKs proof π_{Rev}(or π'_{Rev}) extracts pk_i (or pk'_i). For (a), $\mathcal{A}$ must be unaware of the commitment cm_i^{new} (or $cm_i^{new'}$) under the premise of distinguishing the pk_i (or pk'_i) contained in it, which means This destroys the concealment property of COMM. For (b), also because $\mathcal{A}$ cannot break through the zero-knowledge property of zk-SNARKs mentioned in Section 3, it cannot be obtained from zero-knowledge The address of the recipient is also not available in the proof. In the same way, it is the same for unfreezing transactions corresponding to the same type of payment channel.

c)Suppose $\mathcal{A}$outputs a payment channel update transaction tuple (tx_{Update},tx'_{Update}), where tx_{Update} meets the following conditions:

$$tx_{\text{Update}} = (rt, \pi_{\text{Update}}, cm_j^{\text{new}}, cm_i^{\text{new}}, pk_{2-PC}, sn_j^{pc}, sn_i^{pc}, \sigma_{Update})$$
$$cm_j^{pc} = \text{COMM}\left(pk_{2-PC}, pk_j, v_j^{\text{old}}, sn_j^{\text{old}}, sn_j^{pc}\right)$$
$$cm_i^{pc} = \text{COMM}\left(pk_{2-PC}, pk_i, v_i^{\text{old}}, sn_i^{\text{old}}, sn_i^{pc}\right)$$
$$cm_j^{\text{new}} = \text{COMM}\left(pk_{2-PC}, pk_j, v_j^{\text{new}}, sn_j^{pc}, sn_j^{\text{new}}\right)$$
$$cm_i^{\text{new}} = \text{COMM}\left(pk_{2-PC}, pk_i, v_i^{\text{new}}, sn_i^{pc}, sn_i^{\text{new}}, Y\right)$$

,tx'_{Update} also satisfied:

$$tx'_{\text{Update}} = (rt', \pi'_{\text{Update}}, cm_j^{new'}, cm_i^{new'}, pk'_{2-PC}, sn_j^{pc'}, sn_i^{pc'}, \sigma'_{\text{Update}})$$
$$cm_j^{pc'} = \text{COMM}\left(pk'_{2-PC}, pk'_j, v_j^{old'}, sn_j^{old'}, sn_j^{pc'}\right)$$
$$cm_i^{pc'} = \text{COMM}\left(pk'_{2-PC}, pk'_i, v_i^{old'}, sn_i^{old'}, sn_i^{pc'}\right)$$
$$cm_j^{new'} = \text{COMM}\left(pk'_{2-PC}, pk'_j, v_j^{new'}, sn_j^{pc'}, sn_j^{new'}\right)$$
$$cm_i^{new'} = \text{COMM}\left(pk'_{2-PC}, pk'_i, v_i^{new'}, sn_i^{pc'}, sn_i^{new'}, Y'\right)$$

To win the TR-UL experiment, one must find a set of transactions (tx_{Update},tx'_{Update}) with receivers pk_i=pk'_i and pk_j=pk'_j. In order to achieve the goal of finding the recipient, $\mathcal{A}$ can also make judgments in three ways: (a) Distinguishing the recipient address from the zk-SNARKs proof, (b) $\mathcal{A}$ first looks up the commitment value cm_j^{pc} (and $cm_j^{pc'}$) used in the transaction tx_Update (tx_Update') from its view, and then uses the Setsend transaction associated with the commitment value to distinguish recipients, (c) Distinguishing the sub-recipient address from cm_j^{new} and cm_i^{new} (and $cm_i^{new'}$).

For approach (a), $\mathcal{A}$ has to prove that (pk_i, pk'_i) and (pk_j, pk'_j) are distinguished, which means that $\mathcal{A}$ needs to break the zero-knowledge property of zk-SNARKs. For method (b), $\mathcal{A}$ can be used without knowing the secret values among cm_j^{pc}, cm_i^{pc}($cm_j^{pc'}$), ($cm_i^{pc'}$) and others, by finding the previously contained cm_j^{pc}, cm_i^{pc}($cm_j^{pc'}$), and ($cm_i^{pc'}$) transactions to distinguish the receiver (pk_i, pk'_i) 和 (pk_j, pk'_j). But since cm_j^{pc}, cm_i^{pc}($cm_j^{pc'}$), 和 ($cm_i^{pc'}$) are not visible to tx_{Update}(tx'_{Update}), so $\mathcal{A}$ must obtain cm_j^{pc}, cm_i^{pc}($cm_j^{pc'}$) 和 ($cm_i^{pc'}$) in two ways: (1) Merkle root; (2) zk-SNARKs proof： For method (1), $\mathcal{A}$ must be extracted from Merkle root $rt(rt')$ out of cm_j^{pc}, cm_i^{pc}($cm_j^{pc\prime}$), and ($cm_i^{pc'}$), and this needs to destroy CRH anti-collision properties. For method (2), $\mathcal{A}$ must extract cm_j^{pc}, cm_i^{pc}($cm_j^{pc\prime}$), and ($cm_i^{pc'}$) from the zk-SNARKs proof π_{Update} (π'_{Update} , which means that $\mathcal{A}$ must breaks the zero-knowledge property of zk-SNARKs. Approach (c), for the same reason as above, would violate the hidden property of COMM. Therefore, due to the security features of the commitment scheme, hash function, and zk-SNARKs, it is guaranteed that the receiver of the transaction cannot be distinguished in the tx_{Update} transaction. In the same way, it is the same for the same type of transfer transactions.

B.2 Proof of Balance Conservation

We modify the balance conservation experiment without affecting the $\mathcal{A}$ view: for each tx_{Update} transaction recorded on the ledger L, the challenger $\mathcal{C}$ needs to provide computational evidence w :=($path_j, path_i, cm_j^{pc}, cm_i^{pc}, pk_j, pk_i$, $v_j^{\text{old}}, v_i^{\text{old}}, v_j^{\text{new}}, v_i^{\text{new}}, sn_j^{\text{new}}, sn_i^{\text{new}}, sn_j^{\text{old}}, sn_i^{\text{old}}, r_j^{\text{new}}, r_i^{\text{new}}$, $r_j^{\text{old}}, r_i^{\text{old}}$, Y) for each zk-SNARKs instance x :=(rt, $cm_j^{new}, cm_i^{new}, pk_{2-PC}, sn_j^{pc}, sn_i^{pc}$). In this way, $\mathcal{C}$ obtains an augmented ledger (L, W), where w_i is the witness of the zk-SNARKs instance x_i of the ith Update transaction in L.

Balance conservation ledger: An augmented ledger (L, w) is balanced if the following conditions hold.

Condition I: Each $(tx_{Update}, w) \in (L, W)$ announces two different monetary commitment values cm_j^{pc} and cm_i^{pc} unique information (ie serial number sn_j^{pc} and sn_i^{pc}). Also the monetary commitments cm_j^{pc} and cm_i^{pc} are both output by tx previously recorded on L.

Condition II: There are no two (tx_{Update}, w) and (tx'_{Update},$w'in$(L, W)) contains the only information about the same monetary commitment (i.e. no two transactions contain the same sequence number). Condition III: (cm_j^{pc}, cm_i^{pc} contained in each $(tx_{Update}, w) \in (L, W)$,

cm_j^{new}, cm_i^{new}) corresponds to $(v_j^{\text{new}}, v_i^{\text{new}}, v_j^{\text{old}}, v_i^{\text{old}})$ all satisfy $v_j^{\text{new}} + v_i^{\text{new}} = v_j^{\text{old}} + v_i^{\text{old}}$.

Condition IV: For each of the two monetary commitments cm_j^{pc} and cm_i^{pc}, if they are all the outputs of the Setsend transaction tx_{SetSend} on L, then tx_{SetSend} Witness that the value v_j (or v_i) of cm_j^{pc} (or cm_i^{pc}) contained in w is equal to v_j^{old} (or v_i^{old}).

Condition V: If $\left(tx_{\text{Update}}, w\right) \in (L, W)$ is inserted into L by A, and the cm_i^{pc} and cm_j^{pc} in tx_{Update} are outputs of previous transactions $tx_i{}'$ and $tx_j{}'$ recorded in L, it implies that the receivers in $tx_i{}'$ and $tx_j{}'$ are not users in Addr.

Intuitively, the aforementioned conditions ensure that in L, A does not spend money that was not previously minted or money not under the control of A. Specifically, if (L, W) is balance-preserving, then $v_{Unspent} + v_{Addr \leftarrow A} = v_{Mint} + v_{A \leftarrow Addr}$. For each of the aforementioned cases, we have proven in a contradictory manner that the probabilities of each case not holding can be neglected. In each case, we have demonstrated how to derive a contradiction, thereby providing the proof.

Violation of Condition I. For each $\left(tx_{\text{Update}}, w\right) \in (L, W)$ if not entered by A, then tx_{Update} must satisfy the condition I.

Violation of Condition II. Suppose $\Pr[$A wins but violates condition II] is not negligible. When condition II is violated, L contains two Update transactions $tx_{Update}, tx'_{Update}$ spend the same monetary commitment cm , and show two serial numbers sn and sn . Since $tx_{Update}, tx'_{Update}$ is valid, it must be $sn \neq sn'$. However, if two Update transactions cost cm but produce different sequence numbers, then the corresponding witnesses $\mathrm{w_N w}^{prime}$ contains public information of different cm. However, this contradicts the binding properties of the commitment scheme COMM.

Violating condition III: Let's assume that $\Pr[$A wins but violates condition III], which is non-negligible. In this case, the contradiction is direct: whenever condition III is violated, the equation $v_j^{\text{new}} + v_i^{\text{new}} = v_j^{\text{old}} + v_i^{\text{old}}$ does not hold, violating the soundness of the zk-SNARKs.

Violating condition IV: Let's assume that $\Pr[$A wins but violates condition IV] is non-negligible. It can be observed that when condition IV is violated, the values committed in cm_j^{pc} 和 cm_i^{pc} for the tx_{Update} transaction are different from the values committed in the cm_j^{pc} and cm_i^{pc} of the previous transaction tx recorded in L. This contradicts the binding property of the commitment scheme $COMM$.

Violating condition V: Let's assume that $\Pr[$A wins but violates condition V] is non-negligible. It can be observed that when condition V is violated, L contains an inserted Update transaction tx_{Update} that spends the output of the previous transaction t x' to the address pk in Addr. This would require A to forge the private key signature of pk, violating the unforgeability property of the signature scheme.

Finally, utilizing a similar structural proof, it can be shown that $\mathcal{O}^{\mathrm{CAPE}}$ generates balance-preserving transactions in response to Mint, SetSend, Transfer1, Transfer2, UnFreeze, and Rev queries. Consequently, it can be derived that $\Pr\left[\mathrm{CAPE}_{\prod,\mathcal{A}}^{\mathrm{BAL}}(\lambda) = 1\right]$ is negligible.