DMC: DECENTRALIZED MIXER WITH CHANNEL FOR TRANSACTION PRIVACY PROTECTION ON ETHEREUM

Su Liu and Jian Wang

College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China

ABSTRACT

Ethereum is a public blockchain platform with smart contract. However, it has transaction privacy issues due to the openness of the underlying ledger. Decentralized mixing schemes are presented to hide transaction relationship and transferred amount, but suffer from high transaction cost and long transaction latency. To overcome the two challenges, we propose the idea of batch accounting, adopting batch processing at the time of accounting. For further realization, we introduce payment channel technology into decentralized mixer. Since intermediate transactions between two parties do not need network consensus, our scheme can reduce both transaction cost and transaction latency. Moreover, we provide informal definitions and proofs of our scheme's security. Finally, our scheme is implemented based on zk-SNARKs and Ganache, and experimental results show that the higher number of transactions in batch, the better our scheme performs.

KEYWORDS

Ethereum, transaction privacy, decentralized coin mixer, payment channel, zero-knowledge proof.

1. INTRODUCTION

During the past few years, the blockchain technology has drawn tremendous interests from IT industries (e.g. Google, Alibaba and Amazon) to financial institutions (e.g., Goldman Sachs and JP Morgan). Currently, main application areas of blockchain involve finance, payment, data services and so on. Especially in the financial industry, transaction is a significant component, representing the main financial activity of enterprises and individuals. The importance of data places great demands on security and privacy of blockchain.

Ethereum is a distributed append-only public transaction ledger maintained by consensus protocols. However, it suffers from transaction privacy leakage [1], [2] due to the decentralized nature of blockchain. Though the generated account addresses are pseudonymous, but it is possible to link these addresses with real world identities by deanonymization techniques, such as address clustering [3], [4] and transaction graph analysis [5]. Furthermore, transaction relationship and transferred amount between users can be directly obtained via analysing the underlying public ledger, and moreover, attackers can infer users' income levels, spending habits, etc. Therefore, the openness and sensitiveness of transaction information force Ethereum community to design solutions to guarantee transaction privacy.

David C. Wyld: CSITY, NWCOM, SIGPRO, ASOFT, AIFZ, BDIoT, ITCCMA, CLSB, DTMN, MLNLP - 2021
pp. 125-145, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.111412

Recently, some schemes and projects [6]-[9], [11], [13]-[15] have been proposed, attempting to solve the privacy problem while ensuring the verifiability of transactions and the reliability of the ledger. Among them, the main method is coin mixer [8], [9], [11], [13]-[15], where senders deposit some coins into a centralized third party or a smart contract, then the third party or contract transfers the equivalent coins to receivers when they withdraw from the mixer. These mixing mechanisms can be categorized in two classes: (i) centralized mixer, simple but lack of security and privacy; (ii) decentralized mixer, secure but heavy computation. In the following, we will focus on the problem of decentralized mixer.

In the decentralized mixer with any mixing amount, compared with fixed mixing amount, there are three main operations: (i) deposit, the sender deposits some coins into mixer contract in the form of note; (ii) transfer, the sender exploits created notes inside the contract to transfer to the recipient; (iii) withdraw, the recipient redeems the corresponding coins from the contract. Notably, the verification of transactions generated by the above three operations will consume hundreds of thousands gas for heavy cryptographic computation. Moreover, the high gas price (about $\$24/10^6$ gas at the time of writing) due to expensive computing resources will impose a further cost burden. On the other hand, the transaction latency due to the underlying mechanism (block time interval of about 15 seconds and transaction confirmation time of about 10 minutes [16]) and network congestion is also intolerable. In all, the costly transaction fee (equal to the multiplication of gas used and gas price) and the long transaction latency make it challenge to put the decentralized mixing scheme into practice.

To address the aforementioned issues, we introduce the concept of batch accounting. It means that not every time a transaction occurs, it is submitted to the blockchain, but when several transactions are completed, the final transaction result is recorded on the blockchain. Inspired by payment channel technique, whose key idea is to only record the final transaction result on the blockchain, ignoring intermediate transaction process between two parties, we employ payment channel as the technical support behind batch accounting, proposing a decentralized mixer with channel (DMC). The original transfer operation in the above mixing scheme is completed through payment channel, i.e., off-chain transmission of transaction messages. More specifically, when need to transact, the sender utilizes notes in mixer contract to create a channel note, served as a payment channel. The deposit in the channel is equal to the denomination of the channel note. Then via the channel, the sender creates new transactions (including the total amount that the sender needs to transfer to the recipient so far) and sends them directly to the recipient through anonymous network. After receiving a transaction, the recipient verifies and decides whether to accept it. Essentially, the transaction between two parties is the redistribution of the deposit in the channel. When not need the channel again, the recipient closes the channel by posting the latest received transaction to blockchain.

Since intermediate transactions between two parties do not go through the Ethereum network, no decentralized consensus is required. Hence, our scheme can achieve the reduction in transaction cost and latency. Specifically, due to being free from the influence of underlying mechanism and network congestion, the latency of off-chain transactions can be decreased to the communication delay of anonymous network. On the other hand, although the recipient still needs to verify the correctness of off-chain transactions, the verification is performed locally rather than on the Ethereum virtual machine (EVM), so our scheme gets rid of the expensive computing resources on Ethereum and achieves very low transaction cost. Compared with original transactions, the cost and latency of off-chain transactions are both negligible.

Contributions. In summary, the main contributions of this paper are as follows.

(1) We propose the idea of batch accounting and construct a decentralized mixer with channel called DMC, which reduces transaction cost and latency while hides the transaction relationship and transferred amount. Our scheme is suitable for frequent transactions between senders and recipients. Because there is no need for network consensus for most transactions of two parties, the reduction in cost and latency is achieved.

(2) We implement DMC based on zk-SNARKs and Ganache, and conduct a number of experiments to evaluate its performances. The results show that our scheme is feasible. Combined with theoretical and experimental analysis of DMC, we can get the average transaction cost and transaction latency are both approximately $1/n$ of other mixing schemes, n denoting the unfixed number of transactions in batch.

Paper Organization. The rest of the paper is organized as follows. Section 2 provides some background on Ethereum and payment channel, and explains the cryptographic primitives. Section 3 describes the decentralized mixing scheme. Then, we construct our scheme DMC based on the above decentralized scheme in Section 4. Furthermore, Section 5 details the implementation of DMC, evaluates its performance, and draws comparisons with other schemes. The related work is reviewed in Section 6. Finally, we conclude this paper in Section 7.

2. PRELIMINARIES

In this section, we outline the related background of Ethereum and payment channel. In addition, we describe the cryptographic primitives for DMC: commitment scheme, public key encryption scheme and zero-knowledge proof zk-SNARKs.

2.1. Ethereum

In Ethereum, account is an important concept, indexed by address. There are two types of accounts — Externally Owned Account (EOA), representing a user with a pair of public key pk and secret key sk; and contract account, representing a smart contract with code and storage. The interaction between accounts is made through transactions generated by EOA. A transaction is composed of the destination account address, the transferred amount v, an optional data field $data$ (specifying the called function and the passed parameters), and a signature, etc. In this paper, we denote a specific transaction by $\mathsf{tx} = (data)$.; if the transferred amount v exists, it should be pointed out in addition. Each transaction needs to pay a certain transaction fee for operations made in the transaction, which uses gas as the unit for measuring the computational and storage resources. Take some contract operations for example, storing costs 20,000 units of gas while writing and reading cost 5,000 and 200 respectively [18].

2.2. Payment Channel

The payment channel technology [19], [20] is an important proposal to address the challenges of the scalability and transaction fee. Lightning network [21] and Raiden network [22] are popular examples deployed on Bitcoin and Ethereum respectively. In Bitcoin or Ethereum, a payment channel is corresponding to a multi-signature address or a smart contract. The payment channel technique includes three procedures: (i) opening a channel, the sender deposits into a multi-signature address/smart contract to create a payment channel; (ii) off-chain transactions, the sender sends signed transaction messages directly to the recipient without passing through the blockchain network; 3) closing a channel, the recipient withdraws from the channel and the remaining coins are returned to the sender. Essentially, the transactions between the two parties are the redistribution of the deposit in the channel.

2.3. Cryptographic Building Blocks

Next, we will describe the cryptographic building blocks involved in our scheme. More details about these cryptographic primitives are available in [23]. In the following, λdenotes the security parameter and CRH denotes collision-resistant hash function.

Commitment Scheme. A commitment scheme is composed of two algorithms (Comm, Open) such that:

•$cm \leftarrow$ Comm (m, r): given message m and randomness r, output commitment cm.
• $\{0, 1\} \leftarrow$ Open (cm, m, r): given commitment cm, message m and randomness r, output 1 if cm= Comm(m, r) and 0 otherwise.

For the purposes of this paper, we will use the commitment scheme which is statistically binding and computationally hiding.

Public Key Encryption Scheme. A public key encryption scheme comprises a triple of algorithms (KeyGen, Encrypt, Decrypt) such that:

•$(pk, sk) \leftarrow$ KeyGen(1^{λ}): given security parameter 1^{λ}, output a pair of public key pk and secret key sk.
•$c \leftarrow$ Encrypt (pk, m): given public key pk and plaintext m, output ciphertext c.
•$m \leftarrow$ Decrypt (sk, c): given secret key sk and ciphertext c, output plaintext m or an invalid symbol ⊥.

In this paper, we directly adopt user's public and secret key on Ethereum. So, it's a good choice to use *eciespy* [24], an Elliptic Curve Integrated Encryption Scheme for Ethereum.

Non-Interactive Zero-Knowledge Proof (NIZK). A non-interactive zero-knowledge proof is a two-party protocol between a prover and a verifier with two stages. At the proving stage, the prover uses private data to generate the proof without interaction with the verifier. At the verifying stage, the verifier checks the validity of the proof while obtaining no more information.

Let $\mathcal{R}$ be a binary relation for instance x and witness ω, and let $\mathcal{L}$ be corresponding language $\mathcal{L} = \{x \mid \exists\omega: (x, \omega) \in \mathscr{R}\}$. NIZK is a protocol where a prover tries to convince a verifier that an instance x is in the language $\mathcal{L}$. In addition, NIZK can permit proving computational statements, but the computational problem needs to be converted into an arithmetic circuit.

A NIZK for arithmetic circuit $\mathcal{C}$ consists of four algorithms (Setup, KeyGen, Prove, Verify) such that:

• $(pp) \leftarrow \text{Setup}_{\text{zkp}}(1^{\lambda})$: given security parameter λ, output public parameters pp.
• $(pk, vk) \leftarrow \text{KeyGen}_{\text{zkp}}(pp, \mathcal{C})$: given pp and arithmetic circuit $\mathcal{C}$, output proving key pk and verification key vk.
• $\pi \leftarrow$ Prove (pk, x, ω): given proving key pk, instance x and witness ω, output non-interactive proof π if $(x, \omega) \in \mathcal{R}$.
• (0, 1) ← Verify (vk, x, π): given verification key vk, instance x, and a proof π, output 1 if $x \in \mathscr{L}$; otherwise 0.

In this paper, we use zero-knowledge Succint Non-interactive ARgument of Knowledge (zk-SNARK), the most preferable NIZK that has succinct proof size and sublinear verification time. zk-SNARK satisfies the following properties: completeness, succinctness, soundness and zero-knowledge.

3. DECENTRALIZED MIXER

In this section, we present a general decentralized mixing scheme by summarizing existing mixing schemes. It utilizes commitment scheme to hide the transaction information and meanwhile applies zero-knowledge proof to ensure the validity of transactions. We firstly provide data structures used in decentralized mixer, then describe its mixing mechanism.

3.1. Data Structures

Note. Like a cheque, a note $note$ includes an owner pk, a denomination v and a random number r (to ensure the uniqueness), i.e., $note = (pk, v, r)$. The following two concepts are associated to a note.

• Commitment, cm = Comm (pk, v, r): obviously, a commitment cm is the commitment to a note $note = (pk, v, r)$, used to hide specific information of the note.
• Serial number, sn = CRH (sk, r): a serial number sn, also called nullifier, is the hash of r and the secret key sk corresponding to pk, used to prevent double-spending issues.

CMTree. CMTree denotes a Merkle tree whose leaf nodes are commitments of created notes. The existence of commitments in CMTree are viewed as proof of ownership of coins in the mixer. Every time a commitment is inserted, the root of CMTree is updated. These generated Merkle roots are then added to an array, denoted by Roots.

SNSet. To prevent double-spending issues, all serial numbers of spent notes are recorded in an array, denoted by SNSet. If the corresponding serial number is in SNSet, it indicates that the note has been spent, otherwise the note can be spent.

3.2. The Mixing Mechanism

In Ethereum, the decentralized mixer is implemented by smart contract. Users make interactions with the mixer contract to deposit, transfer and withdraw. The decentralized mixing mechanism, described in Figure 1, consists of the next three components. Note that we ignore the Create Account algorithm, because it is the same as the creation of accounts in Ethereum. The original accounts in Ethereum are completely compatible with our scheme.

Setup. The setup algorithm is executed only once by a trusted third party (TTP) to generate public parameters and to deploy a mixer contract. Note that the setup algorithm can use secure multi-party computation techniques to mitigate the trust requirement for TTP.

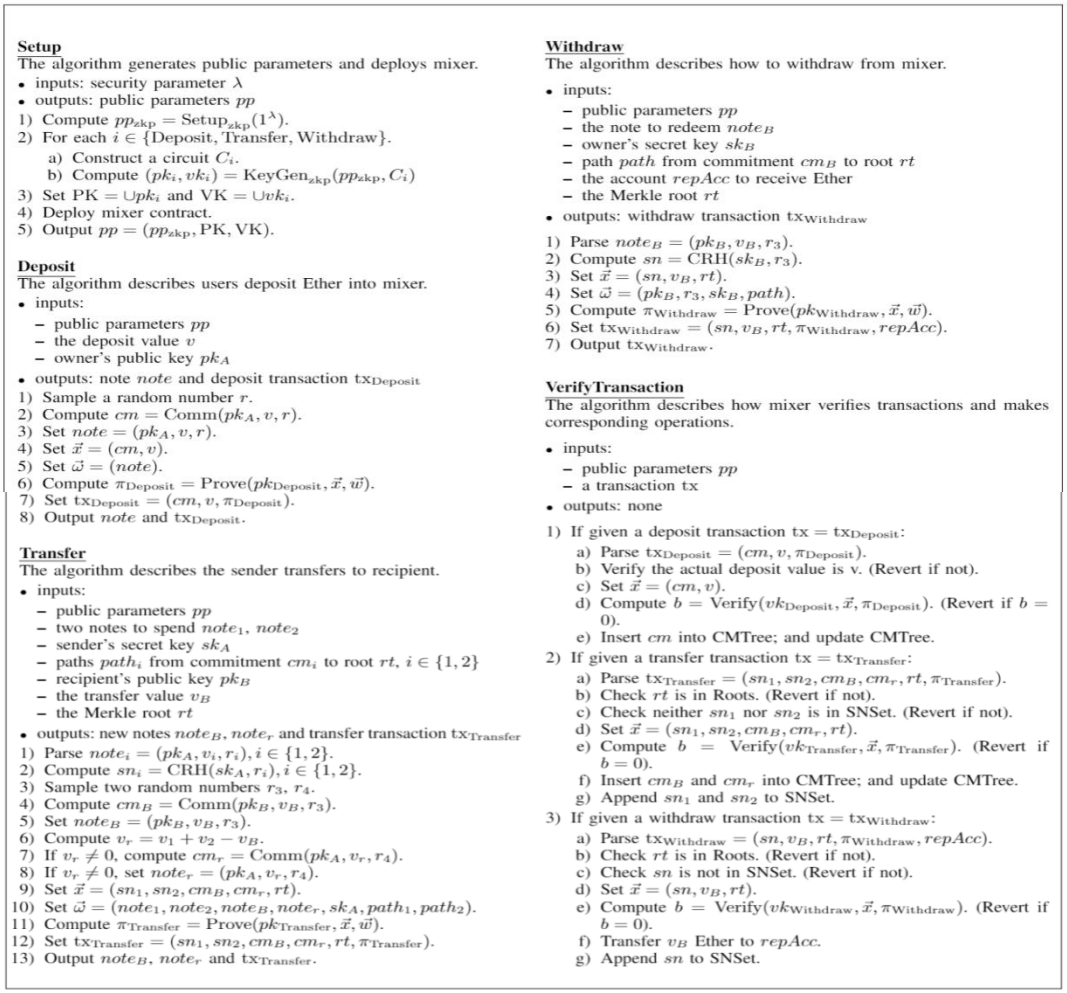
User Algorithms. A user can run the following algorithms to interact with the mixer contract and create valid transactions. For convenience, take sender Alice (A) and recipient Bob (B) for instance.
• Deposit: The Deposit algorithm is to convert some Ether into an equivalent note, e.g., Alice deposits v Ether to mixer.
• Transfer: The Transfer algorithm is to destroy some old notes and create some new notes. For example, Alice uses her two notes to transfer v_B Ether to Bob.

• Withdraw: The Withdraw algorithm is to redeem some note into equivalent Ether, e.g., Bob redeems v_B Ether from mixer.

Mixer Contract. Firstly, users use one of three algorithms mentioned above to generate corresponding transactions, and send them to the blockchain network. After users submitting transactions, the mixer contract verifies and conducts related operations according to the Verify Transaction algorithm. Specifically, the mixer contract is responsible for verifying the correctness of transactions (e.g., verifying zero-knowledge proofs and serial numbers), and if passing the verification, making corresponding changes (e.g., inserting new commitments into CMTree and serial numbers into SNSet).

4. DMC: DECENTRALIZED MIXER WITH CHANNEL

The section gives a detailed description of our scheme based on the decentralized mixing scheme in Section 3. We first discuss the intuition of the scheme based on the next three attempts, then



Setup
The algorithm generates public parameters and deploys mixer.
- inputs: security parameter λ
- outputs: public parameters pp

1) Compute $pp_{\text{zkp}} = \text{Setup}_{\text{zkp}}(1^{\lambda})$.
2) For each $i \in \{\text{Deposit}, \text{Transfer}, \text{Withdraw}\}$.
   a) Construct a circuit C_i.
   b) Compute $(pk_i, vk_i) = \text{KeyGen}_{\text{zkp}}(pp_{\text{zkp}}, C_i)$
3) Set $\text{PK} = \cup pk_i$ and $\text{VK} = \cup vk_i$.
4) Deploy mixer contract.
5) Output $pp = (pp_{\text{zkp}}, \text{PK}, \text{VK})$.

Deposit
The algorithm describes users deposit Ether into mixer.
- inputs:
  - public parameters pp
  - the deposit value v
  - owner's public key pk_A
- outputs: note $note$ and deposit transaction $\text{tx}_{\text{Deposit}}$

1) Sample a random number r.
2) Compute $cm = \text{Comm}(pk_A, v, r)$.
3) Set $note = (pk_A, v, r)$.
4) Set $\vec{x} = (cm, v)$.
5) Set $\vec{\omega} = (note)$.
6) Compute $\pi_{\text{Deposit}} = \text{Prove}(pk_{\text{Deposit}}, \vec{x}, \vec{w})$.
7) Set $\text{tx}_{\text{Deposit}} = (cm, v, \pi_{\text{Deposit}})$.
8) Output $note$ and $\text{tx}_{\text{Deposit}}$.

Transfer
The algorithm describes the sender transfers to recipient.
- inputs:
  - public parameters pp
  - two notes to spend $note_1$, $note_2$
  - sender's secret key sk_A
  - paths $path_i$ from commitment cm_i to root rt, $i \in \{1, 2\}$
  - recipient's public key pk_B
  - the transfer value v_B
  - the Merkle root rt
- outputs: new notes $note_B$, $note_r$ and transfer transaction $\text{tx}_{\text{Transfer}}$

1) Parse $note_i = (pk_A, v_i, r_i), i \in \{1, 2\}$.
2) Compute $sn_i = \text{CRH}(sk_A, r_i), i \in \{1, 2\}$.
3) Sample two random numbers r_3, r_4.
4) Compute $cm_B = \text{Comm}(pk_B, v_B, r_3)$.
5) Set $note_B = (pk_B, v_B, r_3)$.
6) Compute $v_r = v_1 + v_2 - v_B$.
7) If $v_r \neq 0$, compute $cm_r = \text{Comm}(pk_A, v_r, r_4)$.
8) If $v_r \neq 0$, set $note_r = (pk_A, v_r, r_4)$.
9) Set $\vec{x} = (sn_1, sn_2, cm_B, cm_r, rt)$.
10) Set $\vec{\omega} = (note_1, note_2, note_B, note_r, sk_A, path_1, path_2)$.
11) Compute $\pi_{\text{Transfer}} = \text{Prove}(pk_{\text{Transfer}}, \vec{x}, \vec{w})$.
12) Set $\text{tx}_{\text{Transfer}} = (sn_1, sn_2, cm_B, cm_r, rt, \pi_{\text{Transfer}})$.
13) Output $note_B$, $note_r$ and $\text{tx}_{\text{Transfer}}$.

Withdraw
The algorithm describes how to withdraw from mixer.
- inputs:
  - public parameters pp
  - the note to redeem $note_B$
  - owner's secret key sk_B
  - path $path$ from commitment cm_B to root rt
  - the account $repAcc$ to receive Ether
  - the Merkle root rt
- outputs: withdraw transaction $\text{tx}_{\text{Withdraw}}$

1) Parse $note_B = (pk_B, v_B, r_3)$.
2) Compute $sn = \text{CRH}(sk_B, r_3)$.
3) Set $\vec{x} = (sn, v_B, rt)$.
4) Set $\vec{\omega} = (pk_B, r_3, sk_B, path)$.
5) Compute $\pi_{\text{Withdraw}} = \text{Prove}(pk_{\text{Withdraw}}, \vec{x}, \vec{w})$.
6) Set $\text{tx}_{\text{Withdraw}} = (sn, v_B, rt, \pi_{\text{Withdraw}}, repAcc)$.
7) Output $\text{tx}_{\text{Withdraw}}$.

VerifyTransaction
The algorithm describes how mixer verifies transactions and makes corresponding operations.
- inputs:
  - public parameters pp
  - a transaction tx
- outputs: none

1) If given a deposit transaction $\text{tx} = \text{tx}_{\text{Deposit}}$:
   a) Parse $\text{tx}_{\text{Deposit}} = (cm, v, \pi_{\text{Deposit}})$.
   b) Verify the actual deposit value is v. (Revert if not).
   c) Set $\vec{x} = (cm, v)$.
   d) Compute $b = \text{Verify}(vk_{\text{Deposit}}, \vec{x}, \pi_{\text{Deposit}})$. (Revert if $b = 0$).
   e) Insert cm into CMTree; and update CMTree.
2) If given a transfer transaction $\text{tx} = \text{tx}_{\text{Transfer}}$:
   a) Parse $\text{tx}_{\text{Transfer}} = (sn_1, sn_2, cm_B, cm_r, rt, \pi_{\text{Transfer}})$.
   b) Check rt is in Roots. (Revert if not).
   c) Check neither sn_1 nor sn_2 is in SNSet. (Revert if not).
   d) Set $\vec{x} = (sn_1, sn_2, cm_B, cm_r, rt)$.
   e) Compute $b = \text{Verify}(vk_{\text{Transfer}}, \vec{x}, \pi_{\text{Transfer}})$. (Revert if $b = 0$).
   f) Insert cm_B and cm_r into CMTree; and update CMTree.
   g) Append sn_1 and sn_2 to SNSet.
3) If given a withdraw transaction $\text{tx} = \text{tx}_{\text{Withdraw}}$:
   a) Parse $\text{tx}_{\text{Withdraw}} = (sn, v_B, rt, \pi_{\text{Withdraw}}, repAcc)$.
   b) Check rt is in Roots. (Revert if not).
   c) Check sn is not in SNSet. (Revert if not).
   d) Set $\vec{x} = (sn, v_B, rt)$.
   e) Compute $b = \text{Verify}(vk_{\text{Withdraw}}, \vec{x}, \pi_{\text{Withdraw}})$. (Revert if $b = 0$).
   f) Transfer v_B Ether to $repAcc$.
   g) Append sn to SNSet.

Figure 1. Decentralized mixer mechanism.

describe the specific construction of the scheme Π = (Setup, Deposit, Open-Channel, Offchain-Transfer, CloseChannel, Withdraw, VerifyTransaction, VerifyOffchainTransfer). At last, we provide the security definitions and proofs of our scheme.

4.1. Overview

For further realization of batch accounting, we apply the payment channel technology to the decentralized mixer. Specifically, some changes are made to the transfer operation of decentralized mixer. Here, we outline our construction in three incremental steps; the construction details see below. Note that the scheme of ZETH [11] is taken as base for the design of our proposed work.

Attempt 1: the basic framework. We first describe the basis framework of our scheme and point out its existing problems. Inspired by existing payment channel schemes in Bitcoin and Ethereum, where the channel corresponds to a multi-signature address or a smart contract respectively, we match the channel with a channel note denoted by $chnt$. In the OpenChannel phase, sender Alice utilizes unspent notes to create a channel note as a channel, i.e., $chnt_{AB} = (pk_A, v_{AB}, r_{AB})$, pk_A denoting the owner of the channel, and v_{AB} denoting the deposit locked in the channel later used to transfer to the recipient. In the OffchainTransfer phase, whenever Alice needs to transfer to Bob, she firstly creates a transaction $\text{tx}^i_{\text{OffchainTransfer}} = (chsn_{AB}, cm^i_B, cm^i_A, \pi^i_{\text{OffchainTransfer}})$ and a note $note^i_B = (pk_B, v^i_B, r^i_B), i \in [1, n]$, the value v^i_B representing the total amount Alice needs to transfer to Bob so far, and then transfers them to Bob through an anonymous network such as Tor [25]. The transactions between two parties are essentially the redistribution of the deposit in the channel. In the CloseChannel phase, either of them can post the latest transaction message to the blockchain network to get their money back.

However, the above draft may damage the interests of the recipient. The first problem with the attempt is that the channel note may be spent many times. For example, during the transaction between Alice and Bob, Alice utilizes the same channel note $chnt_{AB}$ to transact with Carl and generates $tx'_{\text{OffchainTransfer}} = (chsn_{AB}, cm'_B, cm'_A, \pi'_{\text{OffchainTransfer}})$ and a note $note_C = (pk_C, v_C, r_C)$. When Carl first closes the channel, the mixer verifies and adds $chsn_{AB}$ into SNSet. And then when Bob tries to close the channel, his transaction will be rejected because the channel note $chnt_{AB}$ has been spent. The second problem is when closing the channel, if the dishonest sender submits previous transactions not the latest transaction, i.e., $\text{tx}^i_{\text{OffchainTransfer}}, i < n$, not $\text{tx}^n_{\text{OffchainTransfer}}$, then the interest of recipient will be damaged due to the total transferred amount is included in the latest transaction.

Attempt 2: maintaining the recipient's interests. We make the second attempt to address the above challenges. To solve the double-spending problem, we require to define the channel's recipient pk_B, the channel note being $chnt_{AB} = (pk_A, v_{AB}, r_{AB}, pk_B)$. When $chnt_{AB}$ is spent to create new note $note_X$ (X is B or C), zero-knowledge proof $\pi_{\text{OffchainTransfer}}$ is needed to prove the recipient in $chnt_{AB}$ is consistent with the owner of $note_X$. Therefore, the channel $chnt_{AB}$ is only used to transact with the recipient defined in the channel note, i.e., pk_B. To prevent the sender from broadcasting previous transactions, we rule that only the recipient can close the channel. The idea is accomplished by introducing difficult problems: (i) the recipient generates a difficult problem $diff_{AB}$ with a solution x, and only sends $diff_{AB}$ to the sender; (ii) the sender defines $diff_{AB}$ in the channel note, i.e., $chnt_{AB} = (pk_A, v_{AB}, r_{AB}, pk_B, diff_{AB})$. It requires that only the one who knows the solution to the difficult problem can close the channel, so only the recipient can make it.

However, the second attempt may harm the interests of the sender. If the dishonest recipient never closes the channel, the balance $v_{AB} - v_B$ in the channel will never be returned to the sender. Not knowing the solution to the difficult problem, the sender has no choice but to wait the recipient to close the channel. This situation will harm the sender's interests.

Attempt 3: maintaining the sender's interests. To overcome the above shortcoming, we set a deadline for a channel, which requires the recipient to close the channel before the deadline, otherwise the sender will have the right to close the channel. When Alice creates a channel, she defines a deadline ddl_{AB} in the channel note, i.e., $chnt_{AB} = (pk_A, v_{AB}, r_{AB}, pk_B, diff_{AB}, ddl_{AB})$. When the deadline has passed, the sender can close the channel by proving that the current time is greater than the deadline. In order to prevent the sender from submitting the previous transaction when closing the channel, it is required that the recipient ought to close before the deadline.

In conclusion, we setup a one-way transaction channel from sender to recipient, which adds the defined recipient to prevent double-spending issues; applies difficult problems to ensure the interests of recipients; and uses the deadline to urge the recipient to close the channel on time.

4.2. Construction of DMC

In the following description, we detail the construction of DMC based on the mixing mechanism in Section 3. A DMC scheme Π is a tuple of algorithms (Setup, Deposit, OpenChannel, OffchainTransfer, CloseChannel, Withdraw, VerifyTransaction, VerifyOffchainTransfer).

Setup. The algorithm generates a list of public parameters. To prove the validity of transactions, we build specific circuits which are taken to create keys for proof generation and verification. And the mixer contract is deployed on Ethereum. The detailed process proceeds as follows:

Setup
The algorithm generates public parameters and deploys mixer.
- inputs: security parameter λ
- outputs: public parameters pp

1) Compute $pp_{\text{zkp}} = \text{Setup}_{\text{zkp}}(1^{\lambda})$.
2) For each $i \in \{\text{Deposit}, \text{OpenChannel}, \text{OffchainTransfer}, \text{Withdraw}, \text{Difficulty}, \text{Deadline}\}$
   a) Construct a circuit C_i.
   b) Compute $(pk_i, vk_i) = \text{KeyGen}_{\text{zkp}}(pp_{\text{zkp}}, C_i)$.
3) Set $\text{PK} = \cup pk_i$ and $\text{VK} = \cup vk_i$.
4) Deploy mixer contract.
5) Output $pp = (\text{PK}, \text{VK})$.

Deposit. The algorithm builds a Deposit transaction $\text{tx}_{\text{Deposit}}$ to convert some Ether into an equivalent note. The transaction $\text{tx}_{\text{Deposit}}$ is composed of these variables.

• A new note commitment cm.
• A deposit value v.
• A zero-knowledge proof π_{Deposit}, proving the following equation: $cm = \text{Comm}(pk_A, v, r)$.
The detailed process proceeds as follows:

Deposit
The algorithm describes users deposit Ether into mixer.
- inputs:
  - public parameters pp
  - the deposit value v
  - owner's public key pk_A
- outputs: note $note$ and Deposit transaction $\text{tx}_{\text{Deposit}}$

1) Sample a random number r.
2) Compute $cm = \text{Comm}(pk_A, v, r)$.
3) Set $note = (pk_A, v, r)$.
4) Set $\vec{x} = (cm, v)$.
5) Set $\vec{\omega} = (note)$.
6) Compute $\pi_{\text{Deposit}} = \text{Prove}(pk_{\text{Deposit}}, \vec{x}, \vec{w})$.
7) Set $\text{tx}_{\text{Deposit}} = (cm, v, \pi_{\text{Deposit}})$.
8) Output $note$ and $\text{tx}_{\text{Deposit}}$.

OpenChannel. The algorithm generates a OpenChannel transaction $\text{tx}_{\text{OpenChannel}}$, which utilizes n (let n =2) notes to create a channel note. The channel note is used as a channel for later transactions. The transaction $\text{tx}_{\text{OpenChannel}}$ is composed of these variables.

• Two serial numbers of spent notes sn_1 and sn_2.
• A channel note commitment $chcm_{AB}$.
• A balance commitment cm_r.
• The Merkle root rt.
• A zero-knowledge proof $\pi_{\text{OpenChannel}}$, proving the following equations.
-cm_i = Comm(pk_A, v_i, r_i), $i \in \{1,2\}$; cm_r =Comm(pk_A, v_r, r_3).
-$chcm_{AB}$ = Comm$(pk_A, v_{AB}, r_{AB}, pk_B, diff_{AB}, ddl_{AB})$.
-$cm_i \in$ CMTree, $i \in \{1,2\}$.
-$v_1 + v_2 = v_{AB} + v_3$.

The detailed process proceeds as follows:

OpenChannel
The algorithm describes the sender creates a channel.

- inputs:
  - public parameters pp
  - two notes to spend $note_1$, $note_2$
  - sender's secret key sk_A
  - paths $path_i$ from commitment cm_i to root rt, $i \in \{1,2\}$
  - recipient's public key pk_B
  - the deposit value v_{AB} locked in channel
  - the difficult problem $diff_{AB}$
  - the deadline ddl_{AB}
  - the Merkle root rt
- outputs: new notes $chnt_{AB}$, $note_r$ and OpenChannel transaction $\text{tx}_{\text{OpenChannel}}$

1) Parse $note_i = (pk_A, v_i, r_i), i \in \{1,2\}$.
2) Compute $sn_i = \text{CRH}(sk_A, r_i), i \in \{1,2\}$.
3) Sample two random numbers r_{AB}, r_3.
4) Compute $chcm_{AB} = \text{Comm}(pk_A, v_{AB}, r_{AB}, pk_B, diff_{AB}, ddl_{AB})$.
5) Set $chnt_{AB} = (pk_A, v_{AB}, r_{AB}, pk_B, diff_{AB}, ddl_{AB})$.
6) Compute $v_r = v_1 + v_2 - v_{AB}$.
7) If $v_r \neq 0$, compute $cm_r = \text{Comm}(pk_A, v_r, r_3)$.
8) If $v_r \neq 0$, set $note_r = (pk_A, v_r, r_3)$.
9) Set $\vec{x} = (sn_1, sn_2, chcm_{AB}, cm_r, rt)$.
10) Set $\vec{\omega} = (note_1, note_2, chnt_{AB}, note_r, sk_A, path_1, path_2)$.
11) Compute $\pi_{\text{OpenChannel}} = \text{Prove}(pk_{\text{OpenChannel}}, \vec{x}, \vec{w})$.
12) Set $\text{tx}_{\text{OpenChannel}} = (sn_1, sn_2, chcm_{AB}, cm_r, rt, \pi_{\text{OpenChannel}})$.
13) Output $chnt_{AB}$, $note_r$ and $\text{tx}_{\text{OpenChannel}}$.

OffchainTransfer. The algorithm utilizes the created channel to transfer to the recipient, generating an OffchainTransfer transaction $\text{tx}_{\text{OffchainTransfer}}$ and a new note $note_B$ which are sent to recipient through anonymous network. Every time the sender intends to transfer, she will execute the algorithm, redistributing the deposit in the channel. The transaction $\text{tx}_{\text{OffchainTransfer}}$ is composed of these variables.

• The serial number of channel note $chsn_{AB}$.
• A transfer commitment cm_B.
• A balance commitment cm_A.
• The Merkle root rt.
•A zero-knowledge proof$\pi_{\text{OffchainTransfer}}$, proving the following equations.
-$chcm_{AB}$ = Comm$(pk_A, v_{AB}, r_{AB}, pk_B, diff_{AB}, ddl_{AB})$.
-cm_i = Comm(pk_i, v_i, r_i), $i \in \{A, B\}$.
-sn_{AB} = CRH(sk_A, r_{AB}).
-$chcm_{AB} \in$ CMTree.
-$v_{AB} = v_B + v_A$.

The detailed process proceeds as follows:

OffchainTransfer
The algorithm describes the sender creates a channel.
- inputs:
  - public parameters pp
  - the channel note $chnt_{AB}$
  - sender's secret key sk_A
  - paths $path_{AB}$ from commitment $chcm_{AB}$ to root rt
  - the value v_B to transfer to recipient in total
  - the Merkle root rt
- outputs: new notes $note_B$, $note_A$ and OffchainTransfer transaction $\text{tx}_{\text{OffchainTransfer}}$

1) Parse $chnt_{AB} = (pk_A, v_{AB}, r_{AB}, pk_B, diff_{AB}, ddl_{AB})$.
2) Compute $chsn_{AB} = \text{CRH}(sk_A, r_{AB})$.
3) Sample two random numbers r_B, r_A.
4) Compute $cm_B = \text{Comm}(pk_B, v_B, r_B)$.
5) Set $note_B = (pk_B, v_B, r_B)$.
6) Compute $C = \text{Encrypt}(pk_B, note_B)$.
7) Compute $v_A = v_{AB} - v_B$.
8) If $v_A \neq 0$, compute $cm_A = \text{Comm}(pk_A, v_A, r_A)$.
9) If $v_A \neq 0$, set $note_A = (pk_A, v_A, r_A)$.
10) Set $\vec{x} = (chsn_{AB}, cm_B, cm_A, rt)$.
11) Set $\vec{\omega} = (chnt_{AB}, note_B, note_A, sk_A, path_{AB})$.
12) Compute $\pi_{\text{OffchainTransfer}} = \text{Prove}(pk_{\text{OffchainTransfer}}, \vec{x}, \vec{w})$.
13) Set $\text{tx}_{\text{OffchainTransfer}} = (chsn_{AB}, cm_B, cm_A, rt, \pi_{\text{OffchainTransfer}})$.
14) Output $note_B$, $note_A$ and $\text{tx}_{\text{OffchainTransfer}}$.

CloseChannel. The CloseChannel operation is divided into two cases to discuss.

1) **CloseChannelbyDiff.** The algorithm describes the recipient generates a CloseChannel transaction $\text{tx}_{\text{CloseChannelbyDiff}}$. The recipient utilizes the solution to difficult problem to generate zero-knowledge proof, and posts $\text{tx}_{\text{CloseChannelbyDiff}}$ to blockchain network before the deadline of the channel. The transaction $\text{tx}_{\text{CloseChannelbyDiff}}$ is composed of the next variables.

• The latest off-chain transaction $\text{tx}_{\text{OffchainTransfer}}$.
• The difficult problem $diff_{AB}$.
• A zero-knowledge proof $\pi_{\text{Difficulty}}$, proving the following equation: x is a solution to $diff_{AB}$.

The detailed process proceeds as follows:

CloseChannelbyDiff
The algorithm describes how to close the channel by difficult problems.
- inputs:
  - public parameters pp
  - the latest off-chain transaction $\text{tx}_{\text{OffchainTransfer}}$
  - the difficult problem $diff_{AB}$
  - the solution x to the difficult problem
- outputs: CloseChannelbyDiff transaction $\text{tx}_{\text{CloseChannelbyDiff}}$

1) Parse $\text{tx}_{\text{OffchainTransfer}}=(chsn_{AB}, cm_B, cm_A, rt, \pi_{\text{OffchainTransfer}})$.
2) Set $\vec{x} = (diff_{AB})$.
3) Set $\vec{\omega} = (x)$.
4) Compute $\pi_{\text{Difficulty}} = \text{Prove}(pk_{\text{Difficulty}}, \vec{x}, \vec{w})$.
5) Set $\text{tx}_{\text{CloseChannelbyDiff}} = (\text{tx}_{\text{OffchainTransfer}}, diff_{AB}, \pi_{\text{Difficulty}})$.
6) Output $\text{tx}_{\text{CloseChannelbyDiff}}$.

2) **CloseChannelbyDdl.** The algorithm describes the sender generates a CloseChannel transaction $\text{tx}_{\text{CloseChannelbyDdl}}$. If the recipient does not close the channel in time, the sender can do by proving to mixer contract that the deadline has passed. The transaction $\text{tx}_{\text{CloseChannelbyDdl}}$ is composed of the next variables.

• The latest off-chain transaction$\text{tx}_{\text{OffchainTransfer}}$.
• The difficult problem ddl_{AB}.
• A zero-knowledge proof π_{Deadline}, proving the following equation: $ct > ddl_{AB}$.

The detailed process proceeds as follows:

CloseChannelbyDdl
The algorithm describes how to close the channel by the deadline.
- inputs:
  - public parameters pp
  - the latest off-chain transaction $\text{tx}_{\text{OffchainTransfer}}$
  - the deadline ddl_{AB}
  - the current time ct
- outputs: CloseChannelbyDdl transaction $\text{tx}_{\text{CloseChannelbyDdl}}$

1) Parse $\text{tx}_{\text{OffchainTransfer}}=(chsn_{AB}, cm_B, cm_A, rt, \pi_{\text{OffchainTransfer}})$.
2) Set $\vec{x} = (ddl_{AB})$.
3) Set $\vec{\omega} = (ct)$.
4) Compute $\pi_{\text{Deadline}} = \text{Prove}(pk_{\text{Deadline}}, \vec{x}, \vec{w})$.
5) Set $\text{tx}_{\text{CloseChannelbyDdl}}=(\text{tx}_{\text{OffchainTransfer}}, ddl_{AB}, \pi_{\text{Deadline}})$.
6) Output $\text{tx}_{\text{CloseChannelbyDdl}}$.

Withdraw. The algorithm constructs a Withdraw transaction to redeem a note into equivalent Ether. The transaction $\text{tx}_{\text{Withdraw}}$ is composed of the following variables.

• The serial numbersn_B.
• The withdraw valuev_B.
• The Merkle root rt.
• An account$repAddr$ to receive Ether.
•A zero-knowledge proofπ_{Withdraw}, proving the following equations.
-cm_B = Comm(pk_B, v_B, r_B).
-sn_B = CRH(sk_B, r_B).
-$cm_B \in$ CMTree.
The detailed process proceeds as follows:

Withdraw
The algorithm describes how to withdraw from mixer.
- inputs:
  - public parameters pp
  - the note to redeem $note_B$
  - owner's secret key sk_B
  - path $path_B$ from commitment cm_B to root rt
  - the account $repAddr$ to receive Ether
  - the Merkle root rt
- outputs: Withdraw transaction $\text{tx}_{\text{Withdraw}}$

1) Parse $note_B = (pk_B, v_B, r_B)$.
2) Compute $sn_B = \text{CRH}(sk_B, r_B)$.
3) Set $\vec{x} = (sn_B, v_B, rt)$.
4) Set $\vec{\omega} = (pk_B, r_B, sk_B, path_B)$.
5) Compute $\pi_{\text{Withdraw}} = \text{Prove}(pk_{\text{Withdraw}}, \vec{x}, \vec{w})$.
6) Set $\text{tx}_{\text{Withdraw}} = (sn_B, v_B, rt, \pi_{\text{Withdraw}}, repAddr)$.
7) Output $\text{tx}_{\text{Withdraw}}$.

VerifyTransaction. This algorithm checks by the mixer contract all transactions except OffchainTransfer transactions. The contract verifies the uniqueness of serial numbers, the correctness of note commitments and the validity of Merkle root. If all checks are satisfied, it will perform corresponding operations: (i) add commitments into CMTree; (ii) append serial numbers to SNSet; or (iii) transfer Ether to defined account. The detailed process proceeds as follows:

VerifyTransaction
The algorithm describes how mixer verifies transactions and makes corresponding operations.
- inputs:
  - public parameters pp
  - a transaction tx
- outputs: none

1) If given a Deposit transaction $\text{tx} = \text{tx}_{\text{Deposit}}$:
   a) Parse $\text{tx}_{\text{Deposit}} = (cm, v, \pi_{\text{Deposit}})$.
   b) Verify the actual deposit value is v. (Revert if not).
   c) Set $\vec{x} = (cm, v)$.
   d) Compute $b = \text{Verify}(vk_{\text{Deposit}}, \vec{x}, \pi_{\text{Deposit}})$. (Revert if $b = 0$).
   e) Insert cm into CMTree; and update CMTree.
2) If given a OpenChannel transaction $\text{tx} = \text{tx}_{\text{OpenChannel}}$:
   a) Parse $\text{tx}_{\text{OpenChannel}}=(sn_1, sn_2, chcm_{AB}, cm_r, rt, \pi_{\text{OpenChannel}})$.
   b) Check rt is in Roots. (Revert if not).
   c) Check neither sn_1 nor sn_2 is in SNSet. (Revert if not).
   d) Set $\vec{x} = (sn_1, sn_2, chcm_{AB}, cm_r, rt)$.
   e) Compute $b = \text{Verify}(vk_{\text{vOpenChannel}}, \vec{x}, \pi_{\text{OpenChannel}})$. (Revert if $b = 0$).
   f) Insert $chcm_{AB}$ and cm_r into CMTree; and update CMTree.
   g) Append sn_1 and sn_2 to SNSet.
3) If given a CloseChannelbyDiff transaction$\text{tx}=\text{tx}_{\text{CloseChannelbyDiff}}$:
   a) Parse $\text{tx}_{\text{CloseChannelbyDiff}}=(\text{tx}_{\text{OffchainTransfer}}, diff_{AB}, \pi_{\text{Difficulty}})$.
   b) Parse $\text{tx}_{\text{OffchainTransfer}}=(chsn_{AB}, cm_B, cm_A, rt, \pi_{\text{OffchainTransfer}})$.
   c) Check rt is in Roots. (Revert if not).
   d) Check sn_{AB} is not in SNSet. (Revert if not).
   e) Set $\vec{x_1} = (chsn_{AB}, cm_B, cm_A, rt)$.
   f) Compute $b_1=\text{Verify}(vk_{\text{OffchainTransfer}}, \vec{x_1}, \pi_{\text{OffchainTransfer}})$. (Revert if $b_1 = 0$).
   g) Set $\vec{x_2} = (diff_{AB})$.
   h) Compute $b_2 = \text{Verify}(vk_{\text{Difficulty}}, \vec{x_2}, \pi_{\text{Difficulty}})$. (Revert if $b_2 = 0$).
   i) Insert cm_B and cm_A into CMTree; and update CMTree.
   j) Append $chsn_{AB}$ to SNSet.
4) If given a CloseChannelbyDdl transaction $\text{tx}=\text{tx}_{\text{CloseChannelbyDdl}}$:
   a) Parse $\text{tx}_{\text{CloseChannelbyDdl}}=(\text{tx}_{\text{OffchainTransfer}}, diff_{AB}, \pi_{\text{Deadline}})$.
   b) Parse $\text{tx}_{\text{OffchainTransfer}}=(chsn_{AB}, cm_B, cm_A, rt, \pi_{\text{OffchainTransfer}})$.
   c) Check rt is in Roots. (Revert if not).
   d) Check sn_{AB} is not in SNSet. (Revert if not).
   e) Set $\vec{x_1} = (chsn_{AB}, cm_B, cm_A, rt)$.
   f) Compute $b_1 = \text{Verify}(vk_{\text{OffchainTransfer}}, \vec{x_1}, \pi_{\text{OffchainTransfer}})$. (Revert if $b_1 = 0$).
   g) Set $\vec{x_2} = (Ddl_{AB})$.
   h) Compute $b_2 = \text{Verify}(vk_{\text{Deadline}}, \vec{x_2}, \pi_{\text{Deadline}})$. (Revert if $b_2 = 0$).
   i) Insert cm_B and cm_A into CMTree; and update CMTree.
   j) Append $chsn_{AB}$ to SNSet.
5) If given a Withdraw transaction $\text{tx} = \text{tx}_{\text{Withdraw}}$:
   a) Parse $\text{tx}_{\text{Withdraw}} = (sn, v_B, rt, \pi_{\text{Withdraw}}, repAcc)$.
   b) Check rt is in Roots. (Revert if not).
   c) Check sn is not in SNSet. (Revert if not).
   d) Set $\vec{x} = (sn, v_B, rt)$.
   e) Compute $b = \text{Verify}(vk_{\text{Withdraw}}, \vec{x}, \pi_{\text{Withdraw}})$. (Revert if $b = 0$).
   f) Transfer v_B Ether to $repAcc$.
   g) Append sn to SNSet.

VerifyOffchainTransfer. This algorithm checks OffchainTransfer transactions by the recipient. If passed, the transaction and note messages are stored. The detailed process proceeds as follows:

VerifyOffchainTransfer
The algorithm describes how the recipient verifies transactions and makes corresponding operations.

- inputs:
  - public parameters pp
  - the transaction $\text{tx}_{\text{OffchainTransfer}}$
  - the note ciphertext C
- outputs: none

1) Parse $\text{tx}_{\text{OffchainTransfer}}=(chsn_{AB}, cm_B, cm_A, rt, \pi_{\text{OffchainTransfer}})$.
2) Check rt is in Roots. (Revert if not).
3) Check $chsn_{AB}$ is not in SNSet. (Revert if not).
4) Set $\vec{x} = (chsn_{AB}, cm_B, cm_A, rt)$.
5) Compute $b = \text{Verify}(vk_{\text{OffchainTransfer}}, \vec{x}, \pi_{\text{OffchainTransfer}})$. (Revert if $b = 0$).
6) Compute $note_B = \text{Decrypt}(sk_B, C)$.
7) Verify cm_B is equal to $\text{Comm}(note_B)$. (Revert if not).
8) Store $\text{tx}_{\text{OffchainTransfer}}$ and $note_B$.

4.3. Security of DMC

Following the security model defined in the Zerocash [17] and BlockMaze [7], we define two secure properties of DMC: transaction unlinkability and overdraft safety. The formal security definitions are provided in Appendix A.

Definition 1 (Security)**.** A DMC scheme is secure if it satisfies transaction unlinkability and overdraft safety as defined in the experiments in Figure 2. (Note, in $\text{DMC}_{\Pi,\mathcal{A}}^{\text{TR-UL}}(\lambda)$, participant Of denotes sender or recipient of transactions, addrOf denotes addresses of the adversary. In $\text{DMC}_{\Pi,\mathcal{A}}^{\text{OD-SF}}(\lambda)$, InOut is used to compute the income and outcome related to the account of $\mathcal{A}$.)

1) Transaction unlinkability. The property, defined by the TR-UL experiment, means that no probabilistic polynomial-time (PPT) adversary can recognize the transaction linkage between the sender and recipient. The scheme Π is transaction unlinkable if

$$\Pr[\text{DMC}_{\Pi,\mathcal{A}}^{\text{TR-UL}}(\lambda) = 1] \leq negl(\lambda) \tag{1}$$

where $\Pr[\text{DMC}_{\Pi,\mathcal{A}}^{\text{TR-UL}}(\lambda) = 1]$ represents the winning probability of $\mathcal{A}$ in the TR-UL experiment.

2) Overdraft Safety. The property, formalized in the OD-SF experiment, shows that no PPT adversary can spend more coins than what he deposits and receives from others. The scheme Π is overdraft safe if

$$\Pr[\text{DMC}_{\Pi,\mathcal{A}}^{\text{OD-SF}}(\lambda) = 1] \leq negl(\lambda) \tag{2}$$

where $\Pr[\text{DMC}_{\Pi,\mathcal{A}}^{\text{OD-SF}}(\lambda) = 1]$ means the winning probability of $\mathcal{A}$ in the OD-SF experiment.

Theorem 1. The tuple Π = (Setup, Deposit, OpenChannel, OffchainTransfer, CloseChannel, Withdraw, VerifyTransaction, VerifyOffchainTransfer) is a secure DMC scheme. (The proof is provided in Appendix B.)

$\text{DMC}_{\Pi,\mathcal{A}}^{\text{TR-UL}}(\lambda)$:
1) $pp \leftarrow \text{Setup}(1^\lambda)$
2) $\text{TX} \leftarrow \mathcal{A}^{\mathcal{O}^{\text{DMC}}}(pp)$
3) $(\text{tx}, \text{tx}') \leftarrow \mathcal{A}^{\mathcal{O}^{\text{DMC}}}(\text{TX})$
4) if $\text{participantOf}(\text{tx}, \text{tx}') \in \text{addrOf}(\mathcal{A})$ then return 0
5) if $(\text{tx}, \text{tx}') = \text{tx}_{\text{CloseChannel}}$ then
   $\text{tx}_1 = \text{tx}.\text{tx}_{\text{OffchainTransfer}}$, $\text{tx}_2 = \text{tx}'.\text{tx}_{\text{OffchainTransfer}}$
   return $\text{tx}_1 \neq \text{tx}_2 \wedge \text{tx}_1.\text{sender} = \text{tx}_2.\text{sender}$
6) return 0

$\text{DMC}_{\Pi,\mathcal{A}}^{\text{OD-SF}}(\lambda)$:
1) $pp \leftarrow \text{Setup}(1^\lambda)$
2) $\text{TX} \leftarrow \mathcal{A}^{\mathcal{O}^{\text{DMC}}}(pp)$
3) $\text{NCS} \leftarrow \mathcal{A}^{\mathcal{O}^{\text{DMC}}}(\text{TX})$
4) $(v_{\text{Deposit}}, v_{\text{Acc}\to\mathcal{A}}, v_{\text{Withdraw}}, v_{\mathcal{A}\to\text{Acc}}, v_{\text{unspent}}) \leftarrow$ *InOut*(TX, NCS)
5) if $(v_{\text{Withdraw}} + v_{\mathcal{A}\to\text{Acc}} + v_{\text{unspent}} > v_{\text{Deposit}} + v_{\text{Acc}\to\mathcal{A}})$ then return 1
6) return 0

Figure 2. The transaction unlinkability and overdraft safety experiment for DMC.

5. IMPLEMENT AND PERFORMANCE EVALUATION

In this section, we first instantiate cryptographic building blocks, and then implement our DMC scheme as a specific mixer contract. At last, we conduct comprehensive experiments to evaluate its performance. Our source code will be available at https://github.com/LS291730/DMC.

5.1. Cryptographic Building Blocks

As for collision-resistant cryptographic hash function (CRH), we choose MiMC [26] hash. Compared to other hash functions (e.g. SHA-256 and Keccak), MiMC is friendly to arithmetic circuits, creating lower number of constraints and operations. The commitment scheme is directly instantiated using MiMC hash function, same as the difficult problem and the hash algorithm used in the Merkle tree.

We use the *eciespy*, Elliptic Curve Integrated Encryption Scheme (ECIES) for secp256k1 in Python, for the encryption scheme. In this scheme, transaction messages are directly encrypted with Ethereum public key and decrypted with Ethereum private key.

We take Groth16 [27] as our instance of zk-SNARKs due to its efficiency in term of proof size and verification time. Groth16 is an excellent zk-SNARK proving scheme which, compared with other schemes, has a smaller proof size with fixed 256 bytes and a faster verification speed at millisecond level. Note that in our implementation, the setup phase of zero-knowledge proof is created by a trusted third party.

5.2. Implementation

We implement our DMC scheme based on zk-SNARK tools (e.g. *circom* [27], a low-level circuit language and a compiler, and *snarkjs* [28], a JavaScript implementation of zk-SNARKs), and Ethereum tools (e.g. Web3.py [29], a Python library for interacting with Ethereum, and Ganache, a local Ethereum blockchain which generates some virtual accounts that we can use during development.). For user algorithms, written by Python, they allow users to create transactions. Users can send via Web3.py these transactions to the blockchain network, interacting with mixer contract. In addition, we use *cir com* to construct arithmetic circuits; and later apply zero-knowledge tool *snarkjs* to generate and validate zero-knowledge proofs. For mixer contract, it is programmed by Solidity, compiled to EVM bytecode and later deployed on Ganache. The functions in mixer, such as *deposit*, *openChannel*, *closeChannel* and *withdraw*, will verify corresponding transactions and make corresponding operations. Note that DMC currently only supports private transfer of Ether, but can later be expanded to support various tokens, such as ERC-20 and ERC-721 tokens.

zk-SNARKs for DMC transactions. For these transactions in DMC (i.e., Deposit, OpenChannel, OffchainTransfer, CloseChannel and Withdraw), we utilize zk-SNARKs to construct zero-knowledge proofs according to their respective circuits. The common reference string (CRS) related to each zero-knowledge proof is generated by a trusted third party and later destroyed to guarantee security. And the generated key pairs for proof generation and verification are public, available to users and mixer contract.

5.3. Performance Evaluation

We conduct experiments to evaluate the performance of the proposed mixing scheme. First of all, we estimate the performance of zero-knowledge proofs. Then, we measure the gas cost consumed

by transactions involved in our scheme and analyse the main factors for the gas cost. At last, we analyse the decrease in transaction cost and latency. Note that the following experiments are executed 10 times and we take the average value.

We now consider the performance of zero-knowledge proofs in terms of setup time, key pair size and proof generation/verification time. These are summarized in table 1. Note that the generator time refers to the time executing both $\mathrm{Setup}_{\mathrm{zkp}}$ and $\mathrm{KeyGen}_{\mathrm{zkp}}$ algorithms for each of zero-knowledge proofs. For each proof, the generator time depends on the complexity of circuit (e.g., circuit $\mathcal{C}_{\mathrm{Deposit}}$ contains 1 MiMC gadget while circuit $\mathcal{C}_{\mathrm{OpenChannel}}$ contains 6 MiMC gasgets and 2 Merkle tree gadgets), the same as the generation time. Furthermore, the generator time is linearly dependent on the size of the proving key. Instead, the size of verification key and the time of proof verification are maintained stable, irrelevant to the circuit's complexity.

Table 1. The performance of zero-knowledge proofs.

ZKP	Generator time	Proving key size	Verification key size	Proof generation time	Proof verification time
Deposit	49.4s	1.0MB	640B	0.77s	0.376s
OpenChannel	6m51s	6.2MB	832B	1.53s	0.368s
OffchainTransfer	3m50s	4.1MB	768B	1.33s	0.371s
Withdraw	52.3s	1.8MB	704B	0.91s	0.360s

The cost to deploy the DMC mixing contract is 2,294,567 gas. Table 2 shows the gas cost consumed by these transactions sent to the mixer contract. For Deposit transactions, it consumes the first sender 1,090,661 gas to process the first transaction, but 610,653 (given in the table) for the other transactions. The extra gas is costed to set storage in the EVM. A majority of the gas cost lies in two chief operations: the verification of zero-knowledge proofs and the update of the Merkle tree CMTree, both of which cost approximately 200,000 gas. The numbers of verification and update operations involved in each transaction are given in Table 3. As seen from the table, the CloseChannel transaction costs the most gas while the Withdraw transaction costs the least because the former has two verification and update operations respectively while the latter needs to verify zero-knowledge proof only once.

Table 2. The gas cost of transactions for interacting with mixer contract.

Transaction	Gas cost	#VerifyProof	#UpdateTree
$\mathrm{tx}_{\mathrm{Deposit}}$	610,653	1	1
$\mathrm{tx}_{\mathrm{OpenChannel}}$	919,108	1	2
$\mathrm{tx}_{\mathrm{CloseChannelbyDiff}}$	1,118,862	2	2
$\mathrm{tx}_{\mathrm{CloseChannelbyDdl}}$	1,126,453	2	2
$\mathrm{tx}_{\mathrm{Withdraw}}$	288,421	1	0

Compared with related work, the gas cost of related decentralized mixing schemes is given in Table 3 (Note that *k* and *j* denote the number of participants in the ring signature or shuffle and the number of malicious shuffles respectively). In the first two schemes, one deposit transaction corresponds to one withdrawal transaction without an extra transfer operation. When withdrawing from mixer contract, Möbius and Miximus utilize verifiable ring signature and zero-knowledge proof respectively to create the withdrawal transaction. So, the cost of withdrawal operation in Möbius grows linearly with the number of participants in ring signature, and that in Miximus is relatively high for proof verification. In MixEth, before withdrawing, several shuffle operations are required to perform to break the transaction relationship, and then recipients need to check the correctness of the preceding shuffles. The latter two schemes use existing deposits in

the mixer to transfer to recipients. Zether costs more gas for applying encryption scheme and Σ-Bullets to transfer. While in our scheme DMC, because the sender sends transaction messages directly to the recipient through anonymous network, there is no need to interact with Ethereum network except one transaction to open a channel (919,108 gas) and another to close the channel (1,118,862 gas).

Table 3. Comparison between gas costs of different decentralized mixing schemes.

Mixer	**Deposit**	**Withdraw**	**Transfer**	**Total**
Möbius [9]	76,123	335,714k	-	1,418,979
Miximus [10]	732,815	1,903,305	-	2,636,120
MixEth [12]	99,254	113,265	366,216+10,000k+227,563j	1,528,987
Zether [8]	260,000	384,000	7,188,000	7,832,000
DMC	610,653	288,421	0	3,547,697/n

For a complete transaction between two parties, the total transaction cost includes depositing, transferring (if exists) and withdrawing operations. Here, we set both the number of participants in ring signature/shuffle and the number of malicious shuffles to 4, i.e., $k = 4$ and $j = 4$. In our scheme, we suppose that the sender and recipient make n off-chain transactions in total via the channel, i.e., the number of transactions in batch is n. Since there is no need for consensus for these transactions, the total cost of n transactions only covers depositing, opening and closing of channels and withdrawing operations. By comparison with other schemes, the average cost of a transaction is $3{,}547{,}697/n$, which is approximately $1/n$ of others. For the same reason, the transaction latency is also about $1/n$ of other schemes. Because these n transactions are free from the effect of the underlying block generation mechanism and network congestion. On the other hand, the communication delay of anonymous network, compared with Ethereum network, is negligible.

Overall, the experimental results show that our proposed scheme is feasible on Ethereum. From theoretical and experimental analysis of DMC, we obtain that the average transaction cost and transaction latency are both about $1/n$ of other mixing schemes.

6. RELATED WORK

Currently, transaction privacy-preserving schemes mainly include coin mixer, ring signature, zero-knowledge proof and trusted computation, etc. However, in this paper, we only focus the mixing schemes, more specifically, the decentralized ones. First, based on the balance model, the decentralized mixing schemes can be divided into account-based model [8] and UTXO-based model [11]. On the other hand, these schemes can also be classified to any mixing amount [8], [11] and fixed mixing amount [9], [12]. Some decentralized mixing schemes in the literature are briefly introduced as follows.

Mobius [9] presents a decentralized mixer, which only supports for transactions of fixed denominations. The scheme just involves deposit and withdrawal operations. To deposit, the sender derives a new stealth address to hide the recipient. When withdrawing, the recipient generates verifiable ring signature to prove his ownership of coins. The ring signature obscures recipients, however the gas cost consumed by signature verification increases linearly with the size of recipient set.

Zether [8] proposes an account-based coin mixer, i.e., users' deposits are placed in accounts in the form of ciphertext. When transferring, it utilizes homomorphic encryption to hide transaction

amount, uses an anonymous account set to hide the sender and recipient, and exploits zero-knowledge proof to ensure the validity of transactions. Though there is no need for trusted setup, the overhead scales linearly with the size of the anonymous set.

MixEth [12], a trustless coin mixer, does not rely on a trusted setup. It uses shuffling method to break the relationship of two parties coin mixing, achieving strong notions of anonymity. Shuffling and challenging rounds are made in turns. Computing the shuffle is done off-chain, verifying the correctness of the new shuffling on-chain.

7. CONCLUSIONS

The decentralized mixing scheme suffers from high transaction cost for complex operations and expensive computing resources and long transaction latency for block generation mechanism and network congestion. In this paper, we adopt the idea of batch accounting to improve efficiency, reducing the transaction cost and latency issues. As the technical support behind batch accounting, we introduce payment channel technology into the mixing scheme and propose a decentralized mixer with channel called DMC. DMC works well in combination with the advantages of decentralized mixer and payment channel, decreasing the transaction cost and latency, while breaking the transaction relationship and hiding the transaction value. By the created channel, the transactions between two parties are transmitted through anonymous network. Since these off-chain transactions avoid network consensus, we achieve the decrease in transaction cost and latency.

Future scope of our proposed scheme is (i) There is need of utilizing secure multi-party computation (MPC) to avoid the trusted setup of zero-knowledge proof. (ii) The channel between the two sides needs to be expanded from one-way to two-way. (iii) The mixer with channel scheme will be likely to scale into other blockchain system, such as Bitcoin. (iv) The privacy protection method, not just the mixing scheme, can be combined with the two-layer scaling solutions to improve performance.

ACKNOWLEDGEMENTS

This work is partly supported by the National Key Research and Development Program of China (No. 2020YFB1005900), the Research Fund of Guangxi Key Laboratory of Trusted Software (No. kx201906), and the Research Fund of Guangxi Key Lab of Multi-source Information Mining \& Security (No. MIMS20-07).

REFERENCES

[1] Béres, F., Seres, I. A., Benczúr, A. A., &Quintyne-Collins, M. (2020). “Blockchain is watching you: Profiling and deanonymizing ethereum users”. *arXiv preprint arXiv*:2005.14051.
[2] Klusman, R., &Dijkhuizen, T. (2018). “Deanonymisation in ethereum using existing methods for bitcoin”.
[3] Victor, F. (2020). “Address clustering heuristics for Ethereum”. *In International Conference on Financial Cryptography and Data Security*, pp. 617-633.
[4] Payette, J., Schwager, S., & Murphy, J. (2017). “Characterizing the ethereum address space”.
[5] Chan, W., & Olmsted, A. (2017). “Ethereum transaction graph analysis”. *In 2017 12th international conference for internet technology and secured transactions*, pp. 498-500.
[6] Ma, S., Deng, Y., He, D., Zhang, J., &Xie, X. (2020). “An efficient NIZK scheme for privacy-preserving transactions over account-model blockchain”. *IEEE Transactions on Dependable and Secure Computing*, Vol. 18, No. 2,pp. 641-651.

[7] Guan, Z., Wan, Z., Yang, Y., Zhou, Y., & Huang, B. (2020). "Blockmaze: An efficient privacy-preserving account-model blockchain based on zk-SNARKs". *IEEE Transactions on Dependable and Secure Computing*.
[8] Bünz, B., Agrawal, S., Zamani, M., &Boneh, D. (2020). "Zether: Towards privacy in a smart contract world". *In International Conference on Financial Cryptography and Data Security*, pp. 423-443.
[9] Meiklejohn, S., & Mercer, R. (2018). "Möbius: Trustless Tumbling for Transaction Privacy. Proceedings on Privacy Enhancing Technologies", pp. 105-121.
[10] Barry Whitehat. (2018). "Miximus: zksnark-based trustless mixing for Ethereum". https://github.com/barryWhiteHat/miximus.
[11] Rondelet, A., & Zajac, M. (2019). "Zeth: On integrating zerocash on ethereum". *arXiv preprint arXiv*:1904.00905.
[12] Seres, I. A., Nagy, D. A., Buckland, C., &Burcsi, P. (2019). "Mixeth: efficient, trustless coin mixing service for ethereum". *In International Conference on Blockchain Economics, Security and Protocols*.
[13] Zachary J. Williamson. (2018). The AZTEC protocol. Available at: https://github.com/Aztec Protocol/AZTECblob/master/AZTEC.pdf.
[14] Nightfall implementation. Available at: https://github.com/EYBlockchain/nightfall.
[15] Tornado cash implementation. Available at: https://github.com/tornadocash/tornado-core.
[16] Gervais, A., Karame, G. O., Wüst, K., Glykantzis, V., Ritzdorf, H., &Capkun, S. (2016). "On the security and performance of proof of work blockchains". *In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. pp. 3-16.
[17] Sasson, E. B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., &Virza, M. (2014). "Zerocash: Decentralized anonymous payments from bitcoin". *In 2014 IEEE Symposium on Security and Privacy*. pp. 459-474.
[18] Gavin Wood. (2014). "Ethereum: A Secure Decentralised Generalised Transaction Ledger". *Ethereum project yellow paper*.
[19] Tremback, J., & Hess, Z. (2015). "Universal payment channels".
[20] Dziembowski, S., Faust, S., &Hostáková, K. (2018). "General state channel networks". *In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. pp. 949-966.
[21] Poon, J., &Dryja, T. (2016). "The bitcoin lightning network: Scalable off-chain instant payments". Available at: https:// lightning.network/lightning-network-paper.pdf.
[22] Raiden Network. Available at: https://raiden.network.
[23] Bellare, M., &Rogaway, P. (2005). "Introduction to modern cryptography". UcsdCse, 207, 207.
[24] Elliptic Curve Integrated Encryption Scheme for secp256k1 in Python. Available at: https://gith ubcom/ecies/py.
[25] Dingledine, R., Mathewson, N., &Syverson, P. (2004). "Tor: The second-generation onion router". *Naval Research Lab Washington DC*.
[26] Albrecht, M., Grassi, L., Rechberger, C., Roy, A., &Tiessen, T. (2016). "MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity". *In International Conference on the Theory and Application of Cryptology and Information Security*. pp. 191-219.
[27] Groth, J. (2016). "On the size of pairing-based non-interactive arguments". *In Annual international conference on the theory and applications of cryptographic techniques*. pp. 305-326.
[28] Circom. Available at: https://docs.circom.io/.
[29] Snarkjs. Available at: https://github.com/iden3/snarkjs.
[30] Web3.py. Available at: https://web3py.readthedocs.io.

APPENDIX A: DEFINITION OF SECURITY

A DMC scheme Π = (Setup, Deposit, OpenChannel, OffchainTransfer, CloseChannel, Withdraw, VerifyTransaction) is secure if it satisfies transaction unlinkability and overdraft safety. For security definitions, we design two experiments which are employed based on a stateful DMC oracle $\mathcal{O}^{\mathrm{DMC}}$. The $\mathcal{O}^{\mathrm{DMC}}$ provides queries for adversary $\mathcal{A}$, these queries being interfaces for executing the algorithms defined in Π. The oracle is initialized by public parameters *pp* and stores a transaction set TX, a set NCS including a list of a tuple $(note, cm, sn)$ and a set of accounts Acc. Below, we describe these queries made to the oracle $\mathcal{O}^{\mathrm{DMC}}$.

- $\mathcal{Q}$= (**CreateAccount**). The challenger $\mathcal{C}$: (i) computes a key pair (sk, pk) and an address $addr$; (ii) adds $addr$ into Acc; (iii) outputs $(addr, pk)$.
- $\mathcal{Q}$= (**Deposit**, v, pk_A). The challenger $\mathcal{C}$: (i) computes a tuple $(note, cm, sn)$ and a transaction $\text{tx}_{\text{Deposit}}$ by calling Deposit algorithm; (ii) adds $(note, cm, sn)$ to NCS and $\text{tx}_{\text{Deposit}}$ to TX.
- $\mathcal{Q}$ = (**OpenChannel**, $note_1, note_2, v_{AB}, pk_B, diff_{AB}, ddl_{AB}$). The challenger $\mathcal{C}$: (i) computes two tuples $(chnt_{AB}, chcm_{AB}, chsn_{AB})$ and $(note_r, cm_r, sn_r)$ and a transaction $\text{tx}_{\text{OpenChannel}}$ by calling OpenChannel algorithm; (ii) adds these two tuples to NCS and $\text{tx}_{\text{OpenChannel}}$ to TX.
- $\mathcal{Q}$= (**CloseChannel**, $\text{tx}_{\text{OffchainTransfer}}, x$). The challenger $\mathcal{C}$: (i) computes $\text{tx}_{\text{CloseChannel}}$ by calling CloseChannelbyDiff algorithm; (ii) adds $\text{tx}_{\text{CloseChannel}}$ to TX. Note that we only consider the case that the recipient actively closes the channel.
- $\mathcal{Q}$ = (**Withdraw**, $v, addr$). The challenger $\mathcal{C}$: (i) computes $\text{tx}_{\text{Withdraw}}$ by calling Withdraw algorithm; (ii) adds $\text{tx}_{\text{Withdraw}}$ to TX.
- $\mathcal{Q}$= (**Insert**, tx). The challenger $\mathcal{C}$ verifies the output of VerifyTransaction algorithm: if the output is 1, adds the tx to TX; otherwise, it aborts.

A.1 Transaction Unlinkability

Let $\mathcal{T}$ be the set of transaction $\text{tx}_{\text{OffchainTransfer}}$ attached with **CloseChannel** queries. We define the transaction unlinkability experiment $\text{DMC}_{\Pi,\mathcal{A}}^{\text{TR}-\text{UL}}(\lambda)$as follows.

1) The public parameters pp = Setup(1^{λ}) are computed and provided to $\mathcal{A}$.
2) Whenever $\mathcal{A}$ queries $\mathcal{O}^{\text{DMC}}$, answer this query with transaction set TX at each step.
3) Continue answering queries until $\mathcal{A}$ sends a pair of transactions (tx, tx') with the requirements: (i) $(\text{tx}, \text{tx}' \in \mathcal{T})$; (ii) $\text{tx} \neq \text{tx}'$; (iii) the senders and recipients of tx, tx' are not $\mathcal{A}$.
4) The experiment outputs 1 if the senders of (tx, tx') are same and the recipients of (tx, tx') are also same. Otherwise, it outputs 0.

Definition 2 (TR-UL Security)**.** A DMC scheme Π = (Setup, Deposit, OpenChannel, OffchainTransfer, CloseChannel, Withdraw, VerifyTransaction, VerifyOffchainTransfer) is TR-UL secure, if for PPT adversary $\mathcal{A}$, there is a negligible function $negl$ such that$\Pr[\text{DMC}_{\Pi,\mathcal{A}}^{\text{TR}-\text{UL}}(\lambda) = 1] \leq negl(\lambda)$.

A.2 Overdraft Safety

We design the overdraft safety experiment, which means PPT adversary $\mathcal{A}$ tries to attack a given DMC scheme. Firstly, we define five variables for the security model.

- v_{Deposit}, the total value deposited by $\mathcal{A}$. To compute v_{Deposit}, the challenger $\mathcal{C}$ finds out all Deposit transactions recorded in TX via **Deposit** queries and sums up these values which were transferred from $\mathcal{A}$.

- $v_{\text{Acc}} \rightarrow \mathcal{A}$, the total value received by $\mathcal{A}$ from accounts in Acc. To compute $v_{\text{Acc}} \rightarrow \mathcal{A}$, the challenger $\mathcal{C}$ looks up all $\text{tx}_{\text{OffchainTransfer}}$ in CloseChannel transactions recorded in TX via **CloseChannel** queries and adds the values whose recipient are $\mathcal{A}$.

- v_{Withdraw}, the total value redeemed by $\mathcal{A}$. To compute v_{Withdraw}, the challenger $\mathcal{C}$ finds out all Withdraw transactions recorded in TX via **Withdraw** queries and sums up these values which were transferred to $\mathcal{A}$.

- $v_{\mathcal{A}} \rightarrow \text{Acc}$, To compute $v_{\mathcal{A}} \rightarrow \text{Acc}$, the challenger $\mathcal{C}$ looks up all $\text{tx}_{\text{OffchainTransfer}}$ in CloseChannel transactions recorded in TX via **CloseChannel** queries and adds the values whose sender are $\mathcal{A}$.

- v_{unspent}, the spendable amount in cm and $chcm$. The challenger $\mathcal{C}$ can check whether corresponding $note/chnt$ is spendable as follows. For cm, $\mathcal{C}$ checks if a **Withdraw** query which redeems $note$ generates a valid transaction $\text{tx}_{\text{Withdraw}}$. For $chcm$, $\mathcal{C}$ first uses $chnt$ to create an off-chain transaction $\text{tx}_{\text{OffchainTransfer}}$ via a **OffchainTransfer** query, and then checks if a **CloseChannel** query yields a valid transaction $\text{tx}_{\text{CloseChannel}}$, which closes the channel $chnt$ using $\text{tx}_{\text{OffchainTransfer}}$.

For an honest account u, $v_{\text{Withdraw}} + v_{\mathcal{A}\rightarrow\text{Acc}} + v_{\text{unspent}} > v_{\text{Deposit}} + v_{\text{Acc}\rightarrow\mathcal{A}}$.

Formally, we define the overdraft safety experiment $\text{DMC}_{\Pi,\mathcal{A}}^{\text{OD}-\text{SF}}(\lambda)$ as follows.

1) The public parameters pp = Setup (1^{λ}) are computed and provided to $\mathcal{A}$.
2) Whenever $\mathcal{A}$ queries $\mathcal{O}^{\text{DMC}}$, answer this query with transaction set TX at each step.
3) Continue answering queries until $\mathcal{A}$ sends a set NCS.
4) Compute the five variables mentioned above.
5) The experiment outputs 1 if $v_{\text{Withdraw}} + v_{\mathcal{A}\rightarrow\text{Acc}} + v_{\text{unspent}} > v_{\text{Deposit}} + v_{\text{Acc}\rightarrow\mathcal{A}}$. Otherwise, it outputs 0.

Definition 2 (OD-SF Security)**.** A DMC scheme Π = (Setup, Deposit, OpenChannel, OffchainTransfer, CloseChannel, Withdraw,VerifyTransaction, VerifyOffchainTransfer) is OD-SFsecure, if for PPT adversary $\mathcal{A}$, there is a negligible function $negl$ such that $\Pr[\text{DMC}_{\Pi,\mathcal{A}}^{\text{OD}-\text{SF}}(\lambda) = 1] \leq negl(\lambda)$.

APPENDIX B: PROOF OF SECURITY

A DMC scheme Π = (Setup, Deposit, OpenChannel, OffchainTransfer, CloseChannel, Withdraw, VerifyTransaction, VerifyOffchainTransfer) is secure if it satisfies transaction unlinkability and overdraft safety.

B.1 Proof of Transaction Unlinkability

Let $\mathcal{T}$ be the set of transaction $\text{tx}_{\text{OffchainTransfer}}$ attached with **CloseChannel** queries. $\mathcal{A}$ wins the TR-UL experiment when it outputs a pair of transactions (tx, tx') if the senders of (tx, tx') are same and the recipients of (tx, tx') are also same. Suppose $\mathcal{A}$ outputs a pair of transactions $\text{tx}_{\text{CloseChannel}}, \text{tx}'_{\text{CloseChannel}}$. The $\text{tx}_{\text{OffchainTransfer}}$ in $\text{tx}_{\text{CloseChannel}}$ satisfies the following equations:

1) $\text{tx}_{\text{OffchainTransfer}} = (chsn_{AB}, cm_B, cm_A, rt, \pi_{\text{OffchainTransfer}})$.
2) cm_B = Comm(pk_B, v_B, r_B).
3) cm_A = Comm(pk_A, v_A, r_A).

and the $\text{tx}'_{\text{OffchainTransfer}}$ in $\text{tx}'_{\text{CloseChannel}}$ satisfies the following equations:

1) $tx'_{\text{OffchainTransfer}} = (chsn'_{AB}, cm'_B, cm'_A, rt', \pi'_{\text{OffchainTransfer}})$.
2) cm'_B =Comm(pk'_B, v'_B, r'_B).
3) cm'_A =Comm(pk'_A, v'_A, r'_A).

$\mathcal{A}$ wins the TR-UL experiment if the senders and recipients contained in $(\text{tx}_{\text{OffchainTransfer}}, \text{tx}'_{\text{OffchainTransfer}}$ are the same, i.e., $pk_A = pk'_A$ and $pk_B = pk'_B$. There are two ways for $\mathcal{A}$ to distinguish whether $pk_i = pk'_i, i \in \{A, B\}$: (i) distinguish public keys from commitments; (ii) distinguish public keys from the zero-knowledge proofs.

For condition (i), $\mathcal{A}$ must distinguish $pk_i = pk'_i$ based on different commitments $(cm_i, cm'_i), i \in \{A, B\}$ without knowing other secret values, which means that $\mathcal{A}$ ought to break the hiding property of the commitment scheme. For condition (ii), $\mathcal{A}$ must distinguish $pk_i = pk'_i, i \in \{A, B\}$ based on different zero-knowledge proofs $\pi_{\text{OffchainTransfer}}, \pi'_{\text{OffchainTransfer}}$, which means that $\mathcal{A}$ ought to break the proof of knowledge property of the zk-SNARKs. However, due to the security of commitment scheme and zk-SNARks, $\mathcal{A}$ cannot distinguish the two pairs of public keys.

B.2 Proof of Overdraft Safety

We modify the overdraft safety experiment without affecting the view of $\mathcal{A}$. First, for each $\text{tx}_{\text{OffchainTransfer}}$ inside CloseChannel transaction $\text{tx}_{\text{CloseChannel}}$ in TX, $\mathcal{C}$ computes a witness $\vec{\omega} = (chnt_{AB}, note_B, note_A, sk_A, path_{AB})$ for the instance $\vec{x} = (chsn_{AB}, cm_B, cm_A, rt)$. Then, $\mathcal{C}$ constructs an augmented transaction set (TX, W), a list of matched pairs $(\text{tx}_{\text{OffchainTransfer}}, \vec{\omega})$.

Definition 3 (Overdraft safe transaction set)**.** An augmented transaction set (TX, W) is overdraft safe if the following holds.

1) Each $(\text{tx}_{\text{OffchainTransfer}}, \vec{\omega})$ in (TX, W) contains openings (e.g., $chsn_{AB}$) of a channel note commitment $chcm_{AB}$, which is the output of a transaction $\text{tx}_{\text{OpenChannel}}$ that precedes $\text{tx}_{\text{OffchainTransfer}}$ in TX.

2) No two $(\text{tx}_{\text{OffchainTransfer}}, \vec{\omega})$ and $(tx'_{\text{OffchainTransfer}}, \vec{\omega}')$ in (TX, W) contain openings of the same note commitment.

3) Each $(\text{tx}_{\text{OffchainTransfer}}, \vec{\omega})$ in (TX, W) contains openings of $chcm_{AB}, cm_B, cm_A$ to values v_{AB}, v_B, v_A respectively, satisfying $v_{AB} = v_B, +v_A$.
4) For each $(\text{tx}_{\text{OffchainTransfer}}, \vec{\omega})$ in (TX, W), if $chcm_{AB}$ is the output of a transaction $\text{tx}_{\text{OpenChannel}}$ in TX, then its witness ω contains an opening of $chcm_{AB}$ to a value v that is equal to v_{AB}.

5) For each $(\text{tx}_{\text{OffchainTransfer}}, \vec{\omega})$ in (TX, W), where $\text{tx}_{\text{OpenChannel}}$ is inserted by $\mathcal{A}$, it holds that if $chcm_{AB}$ is the output of an earlier transaction $\text{tx}_{\text{OpenChannel}}$, then the public value v in $\text{tx}_{\text{OpenChannel}}$ is equal to $chcm_{AB}$.

One can prove that (TX, W) is overdraft safe if the equation holds: $v_{\text{Withdraw}} + v_{\mathcal{A}\to\text{Acc}} + v_{\text{unspent}} > v_{\text{Deposit}} + v_{\text{Acc}\to\mathcal{A}}$. For each case mentioned above, we prove that five cases are in a negligible probability by way of contradiction. Note that we denote by $\Pr[\mathcal{A}(\overline{Con_k}) = 1]$ a non-negligible probability that $\mathcal{A}$ wins but violates condition $k, k \in \{1,2,3,4\}$.

$\mathcal{A}$ violates Condition 1. During construction of $\mathcal{O}^{\mathrm{DMC}}$, every $(\mathrm{tx}_{\mathrm{OffchainTransfer}}, \vec{\omega})$ in (TX, W) where $\mathrm{tx}_{\mathrm{OffchainTransfer}}$ is not inserted by $\mathcal{A}$ satisfies condition 1; thus, $\Pr[\mathcal{A}(\overline{Con_1}) = 1]$ is a probability that $\mathcal{A}$ inserts $\mathrm{tx}_{\mathrm{OffchainTransfer}}$ to construct $(\mathrm{tx}_{\mathrm{OffchainTransfer}}, \vec{\omega}) \in$ (TX, W) where $chcm_{AB}$ used in $\mathrm{tx}_{\mathrm{OffchainTransfer}}$ is not the output note commitment of any previous transactions before $\mathrm{tx}_{\mathrm{OffchainTransfer}}$ in TX.

Note that the validity of $\mathrm{tx}_{\mathrm{OffchainTransfer}}$ implies that the witness ω contains a valid path $path_{AB}$ for a Merkle tree constructed by commitments in earlier transactions. However, a contradiction can be found: if $chcm_{AB}$ does not previously exist in TX, then $path_{AB}$ is not a valid path but with a valid root rt. Therefore, this violates the property of collision resistance of CRH.

$\mathcal{A}$ violates Condition 2. When condition 2 is violated, TX contains two transactions $\mathrm{tx}_{\mathrm{OffchainTransfer}}$ and $tx'_{\mathrm{OffchainTransfer}}$ that spend the same note commitment $chcm_{AB}$, and yield two different serial numbers $chsn_{AB}$ and $chsn'_{AB}$. Obviously, $\Pr[\mathcal{A}(\overline{Con_2}) = 1]$ is a probability that $\mathcal{A}$ inserts a pair of transactions where $chcm_{AB} = chcm'_{AB}$ and $chsn_{AB} \neq chsn'_{AB}$. However, if the two transactions spend the same $chcm_{AB}$ but create different serial numbers, then corresponding witnesses ω and ω' include different opening of $chcm_{AB}$.Therefore, this contradicts the binding property of the commitment scheme.

$\mathcal{A}$ violates Condition 3. $\Pr[\mathcal{A}(\overline{Con_3}) = 1]$ is a probability that the equation $v_{AB} \neq v_B + v_A$ holds. When violating condition 3, the equation $v_{AB} = v_B + v_A$ does not hold, so violating the soundness of zk-SNARKs during the construction of zero-knowledge proof $\pi_{\mathrm{OffchainTransfer}}$.

$\mathcal{A}$ violates Condition 4. Each $(\mathrm{tx}_{\mathrm{OffchainTransfer}}, \vec{\omega})$ in (TX, W) contains values (i.e., v_{AB}) of $chcm_{AB}$, and $chcm_{AB}$ is also the output commitment to values (including v'_{AB}) in a OpenChannel transaction $\mathrm{tx}_{\mathrm{OpenChannel}}$. Obviously, $\Pr[\mathcal{A}(\overline{Con_4}) = 1]$ is a probability that the euqation $v_{AB} \neq v'_{AB}$ holds. Thus, this contradicts the binding property of commitment scheme.

AUTHORS

Su Liu, female, master's degree. Her main research fields are blockchain technology, privacy protection, etc

Jian Wang, male, doctor, professor, doctoral supervisor. His main research fields are key management, cryptographic protocol, privacy protection, etc.