blockchains

MDPI

Article

Zk-SNARKs-Based Anonymous Payment Channel in Blockchain

Yunwei Guo [1], Haochen Liang [2,*], Liehuang Zhu [2] and Keke Gai [2]

[1] School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China; 3220211008@bit.edu.cn
[2] School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China; liehuangz@bit.edu.cn (L.Z.); gaikeke@bit.edu.cn (K.G.)
* Correspondence: 3220221492@bit.edu.cn

Abstract: Payment channels serve as an effective solution to the scalability problem of cryptocurrencies, which significantly increase transaction rates by allowing users to conduct large-scale offline transactions off-chain without posting everything to the blockchain. However, the existing payment channels lack privacy protection for the transaction amount and the linking relationship between the two parties to the transaction. Therefore, in order to address the scalability and privacy issues of cryptocurrencies such as Bitcoin, this paper proposes a zk-SNARKs-based anonymous payment channel (zk-APC), which supports an unlimited number of off-chain payments between the payer and the payee and protects the privacy of the participants. Specifically, the proposed scheme achieves relational anonymity and amount privacy for both on-chain and off-chain transactions in the payment channel through utilizing zero-knowledge proof (zk-SNARKs) and commitment schemes. This paper proves that the proposed method is more effective than similar schemes through a performance evaluation.

Keywords: payment channel; blockchain; zk-SNARKs; privacy

Citation: Guo, Y.; Liang, H.; Zhu, L.; Gai, K. Zk-SNARKs-Based Anonymous Payment Channel in Blockchain. *Blockchains* **2024**, *2*, 20–39. https://doi.org/10.3390/blockchains2010002

Academic Editor: Faisal Jamil

Received: 29 December 2023
Revised: 23 January 2024
Accepted: 29 January 2024
Published: 5 February 2024

1. Introduction

The origins of blockchain technology can be traced back to Satoshi Nakamoto, the creator of the Bitcoin system [1]. Bitcoin, proposed by Nakamoto, is the first widely applied decentralized digital currency system, revolutionizing the traditional digital currency model that relied on trust in a third-party trusted center. The revolutionary aspect of blockchain technology [2–6] lies in the construction of a new trust model, where trust among users is not dependent on a single node but based on trust in the entire system. In Bitcoin, every transaction is recorded in the blockchain, a public ledger maintained by a group of decentralized nodes, ensuring its security. This public ledger establishes a reliable trust foundation for the Bitcoin system. As long as the blockchain meets the security assumptions of the entire system, its trust model can continue to operate. However, despite the advantages of decentralization, security, and traceability brought by blockchain technology [7,8], its low transaction speed and long transaction time pose scalability challenges. For instance, Bitcoin supports only 6 to 7 transactions per second, and Ethereum [9,10] up to 20 transactions per second, in stark contrast to Visa, which can handle up to 47,000 transactions per second. This low transaction throughput is insufficient for daily high-frequency trading needs, let alone executing complex smart contracts. Therefore, addressing the scalability issues of blockchain technology is crucial to its further proliferation.

Among the technologies addressing the scalability issues of decentralized cryptocurrencies [11–13], payment channel technology is one of the favored solutions. This technology increases the transaction speed of cryptocurrencies and aims to enable high-frequency daily offline transactions globally, achieving almost instantaneous transaction completion. The core idea of payment channel schemes [14,15] is to conduct transactions off-chain and submit the final state to the blockchain, thereby reducing the load on the blockchain. Payment channel technology divides the process between the transaction parties into three

main stages: establishing the payment channel, off-chain transactions, and closing the payment channel. In the establishment phase, the initiator locks the estimated transaction funds on the blockchain, controlled jointly by both parties, to prevent misappropriation. Then, during the off-chain transaction phase, each off-chain transaction results in the initiator reallocating funds within the payment channel.Finally, in the closing phase, the receiver submits the latest off-chain transactions to the blockchain for verification. The benefit of this technology is that users can securely conduct large-scale offline transactions, without frequently interacting with the blockchain. This allows scaling blockchain theory to unlimited transactions, thus supporting daily high-frequency offline transactions for billions of users worldwide.

However, while payment channel technology offers a solution to the scalability issues of blockchain, it inherits many of the well-known weaknesses of blockchain technology, the most prominent being the leakage of private information [16,17]. The current use scenarios of payment channel technology need more protection for two types of private information: the transaction amount, and the relationship between the transaction parties. First, in the existing payment channel model, although transactions are conducted off-chain, the final records on the blockchain ledger still reveal the payment amount. Additionally, establishing and closing payment channels may expose the relationship between the transaction parties. In response to the privacy issues inherent in blockchain technology, recent studies have proposed some solutions, such as ZCash [18], Blockmaze [19], and Monero [20], which have partially resolved these two types of privacy issues of blockchain. However, these privacy mechanisms do not directly address the privacy issues in the payment channel setup. Therefore, while payment channels offer a partial solution to the scalability issues of blockchain, privacy remains a challenge that urgently needs to be addressed.

Our contribution. To resolve the privacy issues in payment channels without affecting their regular use, we propose a zk-SNARKs-based anonymous payment channel, utilizing technologies including zk-SNARKs and verifiable timed commitments [21,22] to achieve privacy protection and fair transactions in payment channels. This scheme ensures regular operation during transactions between parties through the payment channel, while guaranteeing participants' privacy, including the privacy of transaction amounts and the unlinkability of transaction parties. Our contributions are summarized as follows:

- We present the details of a zk-SNARKs-based anonymous payment channel, which retains all the advantages of the original payment channel schemes while being more robust and secure. Specifically, we achieve privacy protection by integrating blockchain systems with zero-knowledge proofs and commitment schemes and ensure fair transactions using verifiable timed commitments.
- Experimental validation of the zk-SNARKs-based anonymous payment channel was conducted, including testing the three stages of the payment channel transaction process and related tests of zk-SNARKs within the entire system.

The remainder of this paper is organized as follows: Section 2 reviews related methods, and Section 3 describes the preparatory work related to zk-SNARKs. Section 6 introduces the relevant concepts and definitions of the scheme, Section 5 details the complete zk-SNARKs-based anonymous payment channel, and Sections 6 and 7 discuss related issues and analyze the performance. Section 8 concludes the paper.

2. Related Work

The purpose of payment channels is to reduce transaction costs and enable rapid payments in cryptocurrency, aiming to increase transaction rates and decrease the transaction fees in current cryptocurrency systems, while potentially penalizing malicious recipients. Hearn and Spilman initially proposed payment channel technology and informally introduced a unidirectional micropayment protocol for the Bitcoin system [23], facilitating small off-chain Bitcoin payments. Subsequently, Ying proposed a novel unidirectional payment channel protocol, xLumi [24], which ensures the security of payment channel funds through a set of simple mathematical rules, significantly reducing the complexity of establishing

payment channels, thereby enabling easy implementation on any blockchain with the necessary infrastructure. Xu introduced a new unidirectional payment channel protocol, Super [25], allowing off-chain one-way payments between a payer and multiple payees and penalizing double-spending, thus greatly expanding the applicability of unidirectional payment channels.

Although the advent of payment channels has enhanced the scalability of blockchain systems, they still possess numerous weaknesses regarding protecting privacy. Consequently, many specific payment channels have been designed for cryptocurrency systems, to address these privacy concerns. Green and Miers proposed a payment channel scheme, Bolt [26], capable of anonymous transactions on Zcash. In this scheme, the transaction parties in the payment channel can achieve privacy-protected off-chain variable-amount payments through hiding transaction amounts and using blind signatures to validate channel updates. However, Bolt faces several issues, including using blind signature algorithms that are incompatible with Bitcoin. Zhang et al. introduced a privacy-protecting payment channel named Z-Channel [27], utilizing zero-knowledge proofs to safeguard user privacy, thereby avoiding the problems associated with blind signatures. However, it still employs the relative time-lock technique common in most payment channels, hindering its wider adoption. Moreover, Moreno and others proposed a payment channel protocol for Monero, DLSAG [28], enabling privacy-protected off-chain transactions through payment channels built on Moreno. However, their solution requires a hard fork and significant changes to Monero's transaction scheme, and it is not backward-compatible. Hence, Thyagarajan et al. proposed a new Monero-specific payment channel protocol, PayMo [29], which requires no changes to Monero's transaction scheme nor additions to the scripting language. Furthermore, any PayMo-related transactions published on the blockchain are indistinguishable from any other regular transactions in Monero. Thus, the PayMo payment channel protocol is now usable within Monero, without system-wide modifications. However, compared to other payment channel schemes, PayMo and DLSAG are custom-designed proposals for Monero that support unidirectional off-chain small payments.

3. Preliminaries

3.1. Blockchain

The blockchain concept was first proposed in 2008 and in essence is a new type of distributed ledger. Blockchain is a decentralized infrastructure that uses encrypted blockchain structures to validate and store data, distributed node consensus algorithms to generate and update data, and smart contracts to program and manipulate data. It is widely used in finance, agriculture, healthcare, the charity sector, and the Internet of things. The blockchain system architecture generally includes six layers: a data layer, network layer, consensus layer, incentive layer, contract layer, and application layer. The data, network, and consensus layers are the three indispensable layers of the standard blockchain framework. Below, we describe the concept and function of each layer in detail.

- The application layer mainly encapsulates blockchain application scenarios, such as finance, supply chain, transportation, medical care, insurance, etc. Blockchain technology can solve some difficult points that cannot be solved using existing information technology. In addition, blockchain's empowerment of traditional industries can further enhance their competitiveness.
- The contract layer mainly encapsulates self-executing code scripts and smart contracts, so that the ledger has programmable features. The emergence of smart contracts has accelerated the application of blockchain technology in various industries and fields. At present, most blockchain applications are DAPPs based on smart contracts.
- The incentive layer integrates an issuance and distribution mechanism and encourages the nodes in the blockchain network to actively participate in the consensus and reward the nodes that verify the safety through the incentive mechanism.
- The consensus layer includes various consensus algorithms, such as PoW, PBFT, etc., which are used for packaging transactions into blocks and blockchains.

- The network layer includes the networking mode of nodes, network transmission protocol, data security transmission mechanism, etc.; the network layer is used to realize communication between nodes.
- The data layer includes data block storage, blockchain storage structure encryption technology, etc., to realize data storage and ensure data security.

3.2. zk-SNARKs

Zero-knowledge proofs can prove the correctness of statements without leaking any additional information, and they are a widely used privacy-preserving technique. As a kind of zero-knowledge proof, zk-SNARK's [30–32] non-interactive zero-knowledge proof scheme is the most widely used among the existing zero-knowledge proofs and can verify the validity of a transaction without the verifying node knowing the specific contents. Zk-SNARKs are often used in privacy protection scenarios and have excellent privacy protection effects.

The basic method of zk-SNARK can be described by a set of polynomial time algorithms Π = (Setup, KeyGen, Prove, Verify), specifically expressed as:

- $pp \leftarrow Setup(1^\lambda)$: Input a security parameter λ, the algorithm outputs a public parameter list pp. pp is published to the public and can be accessed by any user, and the algorithm is only executed once at the very beginning.
- $(pk, vk) \leftarrow KeyGen(\mathcal{C})$: Input a mathematical operation circuit $\mathcal{C}$, the algorithm uses public parameters pp and generates a key pair (pk, vk) for zero-knowledge proof, where pk is the proof key for generating zero-knowledge proof, and vk is the verification key for verifying a zero-knowledge proof key. The key pair are also exposed as a public parameter.
- $\pi \leftarrow Prove(pk, x, w)$: input circuit $\mathcal{C}$ zero-knowledge proof key, the normal input x of circuit $\mathcal{C}$, and the private input (auxiliary input) w of the circuit $\mathcal{C}$, the algorithm generates a x, w that satisfies the relationship $\mathcal{R}_\mathcal{C}$'s non-interactive proof π, constructed using the circuit $\mathcal{C}$. x, π are public.
- $b \leftarrow Verify(vk, x, \pi)$: input circuit $\mathcal{C}$ zero-knowledge proof key, circuit $\mathcal{C}$ normal input x, and the non-interactive proof π generated according to the circuit $\mathcal{C}$, the algorithm verifies the validity of the non-interactive proof π, and outputs the result b. If $b = 0$, the non-interactive proof π is verified valid. Otherwise, the non-interactive proof π is invalid.

Zk-SNARKs have the following properties:

Completeness: This means that an honest prover and a valid witness will always be able to convince a verifier; that is, for any λ, mathematical operation circuit $\mathcal{C}$ and any $(x, w) \in \mathcal{R}_\mathcal{C}$, we have

$$\Pr\left[\text{Verify}(vk, x, \pi) = 1 \middle| \begin{array}{l} (pk, vk) \leftarrow \text{KeyGen}(\mathcal{C}) \\ \pi \leftarrow \text{Prove}(pk, x, w) \end{array}\right] = 1.$$

Conciseness: An honest generated proof π has $\mathcal{O}_\lambda(1)$ bits, verifying that $\text{Verify}(vk, x, \pi)$ runs at time $\mathcal{O}_\lambda(|x|)$. (Here, $\mathcal{O}_\lambda$ hides a fixed polynomial factor in λ)

Non-interactive: The proof is a string that the prover sends to the verifier, and the verifier can directly verify the prover without any back-and-forth interaction.

Proof of knowledge: If the verifier accepts the proof π of the bounded prover, this shows that the prover recognizes the witness w of the given instance. For each PPT adversary $\mathcal{A}$, there is a PPT witness extractor $\mathcal{E}$, namely:

$$\Pr\left[\begin{array}{l} \mathcal{C}(x, w) \neq 0^l \\ \text{Verify}(vk, x, \pi) = 1 \end{array} \middle| \begin{array}{l} (pk, vk) \leftarrow \text{KeyGen}(\mathcal{C}) \\ (x, \pi) \leftarrow \mathcal{A}(pk, vk) \\ w \leftarrow \xi(pk, vk) \end{array}\right].$$

Zero-knowledge property: honestly generated proofs are perfectly zero-knowledge, since a $poly(\lambda)$-*size* simulator S exists such that, for all stateful $poly(\lambda)$-*size* delimiters $\mathcal{D}$, the following two probabilities are equal:

$$\Pr\left[\begin{array}{l|l} (x,w) \in \mathcal{R}_{\mathcal{C}} & (pk,vk) \leftarrow \mathrm{KeyGen}(\mathcal{C}) \\ \mathcal{D}(\pi) = 1 & (x,w) \leftarrow \mathcal{D}(pk,vk) \\ & \pi \leftarrow \mathrm{Prove}(pk,x,w) \end{array}\right],$$

$$\Pr\left[\begin{array}{l|l} (x,w) \in \mathcal{R}_{\mathcal{C}} & (pk,vk,trap) \leftarrow \mathcal{S}(\mathcal{C}) \\ \mathcal{D}(\pi) = 1 & (x,w) \leftarrow \mathcal{D}(pk,vk) \\ & \pi \leftarrow \mathcal{S}(trap,x) \end{array}\right].$$

4. The Concepts and Definitions

4.1. System Model

In this paper, there are four entities involved, namely the certificate authority (CA), users, miners, and trusted parties, and the descriptions of these entities are as follows:

- **Certificate Authority:** This is a trusted third party responsible for generating and managing identity certificates for users or trusted parties.
- **Users:** The main participants in the payment system are users, who can either be payers or receivers. Each user has an account comprising an address and a private key. The user's address serves as their identity and must be registered with the certificate authority before they can participate in the system. On the other hand, a private key is used to transfer coins from one address to another. Additionally, each user can have a long-term address and any number of anonymous addresses to ensure anonymity.
- **Miners:** Miners are responsible for verifying the correctness of transactions and maintaining the public ledger. If the transaction is valid and compliant with policies, they will add it to the blockchain ledger. Otherwise, the miner will discard the transaction, causing the transaction to fail.
- **Trusted parties:** Trusted parties are responsible for initializing the system and generating public parameters for users and miners.

Our system operates as follows: Certificate authorities are responsible for registering and issuing valid transactions to the parties involved. Trusted parties are responsible for initializing the system's public parameters. Miners are responsible for maintaining a public, append-only ledger L_T. Users have the options to engage in regular on-chain transactions or conduct off-chain transactions through payment channels.

4.2. Design Goal

Regarding the system model mentioned above, we primarily focus on the following security objectives:

- **Unlinkability:** The connection between the parties involved in transactions through the payment channel is unlinkable. This attribute ensures the public cannot link the fund sender (consumer) with the corresponding receiver (provider) within the payment channel.
- **Privacy Preserving:** The amount of funds transacted between parties within the payment channel should only be known to those involved; this remains hidden from the public. This means the public must be unaware of how much was spent or received by the participants.

4.3. Threat Model

Regarding the proposed system model and objectives, we define our threat model from the perspective of each entity in the zk-SNARK-based anonymous payment channel.

- Certificate Authority: We assume that the certificate authority is honest and trustworthy and does not disclose any information.
- Users: Since many transacting users are in the system, they are arbitrarily malicious. They will act in their own best interests and deviate from the intended protocol at will.
- Miners: We assume that miners implement a secure consensus algorithm to maintain their blockchain and that our scheme trusts the blockchain as a trusted intermediary to

properly process transactions and smart contracts. However, the blockchain is public to all entities and does not retain private data.

- Trusted party: We assume the trusted party is honest but curious. That is, the trusted party will honestly follow the deployed protocols, but it is also interested in inferring users' details, such as identity and data information.

5. Proposed Model

In order to address the privacy preservation problem faced by payment channels, this paper proposes a zk-SNARKs-based anonymous payment channel. The main idea of our zk-SNARKs-based anonymous payment channel is to use zk-SNARKs and a verifiable timed commitment scheme to ensure correctness and fairness, while preserving privacy and guaranteeing unlinkability. The zk-SNARKs-based anonymous payment channel consists of the following five operations: initialization, minting, payment channel establishment, payment channel update, and payment channel closure.

The initialization operation is performed before all other operations, in which the system's zk-SNARK public parameters, signature public parameters, and verifiable timed commitment scheme commitment public parameters are initialized. The minting operation transforms a certain amount of plaintext currency into zero-knowledge currency, and the payer performs this operation. The payer and the payee jointly perform the payment channel establishment operation to establish a payment channel between the payee and the payer, and this operation freezes a certain amount of zero-knowledge currency of the payer. The payer and the payee perform the payment channel update operation to realize the transfer of funds from the payer to the payee. During the payment channel closure phase, the payee will close the payment channel.

5.1. Phase I: Setup Phase

The system public parameter list pp will be initialized during the setup phase through the Algorithm 1 initialization algorithm. In the initialization algorithm, first, for a given security parameter λ, we will generate the public parameter pp_z using the initialization algorithm in zk-SNARKs. Then, the KenGen algorithm in zk-SNARKs is used to generate a key pair (pk_{zi}, vk_{zi}) for each specific circuit C_i required for zk-APC. These circuits are C_{Mint}, $C_{SetSend}$, C_{Update} and $C_{Transfer}$, which will be used for the generation and verification of transaction proofs in 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.

Algorithm 1 Initialization Algorithm

Input: Safety parameter λ
Output: The list of public parameters pp

1: Compute pp_z = zk-SNARKs.SetUp(1^λ)
2: **for** all i $\in$(Mint, SetSend, Update, Transfer) **do**
3: Construct the circuit C_i corresponding to the required
4: Compute the public-private key pair (pk_{zi}, vk_{zi}) = zk-SNARKs.KenGen(C_i)
5: **end for**
6: Set $PK_z = (pk_{zMint}, pk_{zSetSend}, pk_{zUpdate}, pk_{zTransfer})$ $VK_z = (vk_{zMint}, vk_{zSetSend}, vk_{zUpdate}, vk_{zTransfer})$
7: Compute pp_{BLS} = BLS.Setup(1^λ)
8: Choose a suitable prime order N and corresponding cyclic group G, and choose a suitable generator g. Set pp_{DLG} = (N,G,g)
9: Output $pp = \{pp_z, PK_z, VK_z, pp_{BLS}, pp_{DLG}\}$

The detailed procedure of Algorithm 1 is shown above.

5.2. Phase II: Minting Phase

Before engaging in transactions within the anonymous payment channel, the payer must possess a specific quantity of zero-knowledge currency acquired through the Algorithm 2 mint algorithm. When payer P_i intends to convert their currency using the minting algorithm, the following steps are followed: First, payer P_i must have at least one public address pk_i on the blockchain, with an adequate amount of plaintext currency at 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 minting transaction tx_{Mint} associated with user P_i's public address pk_i comprises the following variables:

- Address pk_i: this is the address of the transaction sender and the address of the transaction receiver.
- Value v_i: this is the value of minting transactions that need to be transformed from plaintext currency into 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.
- The zero-knowledge proof is a proof generated by zk-SNARKs.GenProof that the following conditions apply to the circuit of C_{Mint}:

  (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 minting algorithm is shown below.

Algorithm 2 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 zero-knowledge proof $x :=$ (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 $\sigma_{\text{Mint}} =$ 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}

5.3. Phase III: Payment Channel Establishment Phase

Once the payer of the payment channel P_i has a certain amount of zero-knowledge currency, they can enter the payment channel establishment phase. In this phase, the payer P_i needs to freeze a certain amount of zero-knowledge currency v_i. The purpose of freezing the currency is to prevent the payer P_i from using the same currency to transact with different receivers, which may result in unfair transactions. In order to lock the payer P_i's funds in the payment channel established with the payee P_j, it is necessary to convert the payer P_i's zero-knowledge currency into a new zero-knowledge currency, which is still owned by the payer P_i but in which a discrete logarithmic value Y corresponding to the secret value y chosen by the payee P_j is hidden and can be used only when the the payer P_i can spend the new zero-knowledge currency, and only if the correct secret value y is

provided. Thus, in this way the payer P_i cannot spend the funds frozen in the payment channel at will.

However, while protecting the rights and interests of the payee P_j, we need to consider the possibility that the payee P_j may go offline after establishing the payment channel. If this happens, it will result in the payer P_i's funds being locked in the payment channel forever. Therefore, to prevent such a situation, we must also guarantee that the payer P_i can retrieve the funds after a fixed period T (negotiated off-chain between the two parties). In order to unfreeze the funds locked in the payment channel by the payer P_i, the payee P_j generates a discrete logarithmic value Y and its corresponding verifiable timed commitment, so that the payer P_i unfreezes the funds after time T.

The payment channel establishment operation is performed by the Algorithm 3 payment channel establishment algorithm, through which a payment channel establishment transaction $tx_{SetSend}$ is generated. The payment channel establishment transaction locks the zero-knowledge currency of the payer P_i involved in the transaction into the payment channel. The $tx_{SetSend}$ transaction contains 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^{pc}: This value is generated by the commitment scheme COMM, and the content implied in the commitment includes the Payer's address pk_i, the Payee's address pk_j, the transferred value v_i, commitment serial number sn_i^{old}, and the serial number sn_i^{pc} associated with this commitment value generated by the PRF function and discrete logarithmic value Y.
- Zero-knowledge proof π_{SetSend}: This zero-knowledge proof is a proof generated by zk-SNARKs.GenProof that the following conditions apply to the circuit of C_{SetSend}:
  (1) $cm_i^{old} = \text{COMM}(pk_i, v_i, sn_i^{old}, r_i^{old})$
  (2) $sn_i^{old} = \text{PRF}(sk_i, r_i^{old})$
  (3) $cm^{pc} = \text{COMM}(pk_i, pk_j, v_i, sn_i^{old}, sn_i^{pc}, Y)$
  (4) $sn^{pc} = \text{PRF}(pk_i, r^{new})$
  (5) The path $path_i$ from cm_i^{old} to the rt saved on the ledger is correct

Algorithm 3 Payment Channel Establishment Algorithm

Input: The list of public parameters pp, Merkle root rt, path $path_i$, Zero-knowledge currency c_i^{old} and address pk_j
Output: Zero-knowledge currency c^{pc} and payment channel establishment transactions tx_{SetSend}

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

The detailed procedure of the payment channel establishment algorithm is shown above.

In the payment channel establishment algorithm, the payee P_j calculates the discrete logarithmic value Y by randomly selecting the secret value y and computing it using $Y = g^y$ mod N (where g and N are public parameters). Concurrently, P_j generates a verifiable timed commitment $(C_{\text{VTD}}, \pi_{\text{VTD}}) = \text{VTD.Commit}(y, T)$ based on the time T and y. P_j sends this verifiable time commitment and the discrete logarithm value Y to P_i. Subsequently, the payer P_i initiates the payment channel establishment operation after successful verification via $\text{VTD.Verify}(Y, C_{\text{VTD}}, \pi_{\text{VTD}})$. Meanwhile, P_i begins running $\text{VTD.ForceOp}(C_i)$ until time T has elapsed, computing y to unlock the currency in the payment channel. Upon successfully initiating the payment channel establishment transaction using P_i, this sends all the values concealed in the newly generated promise cm^{pc} to P_j to verify the correct discrete logarithmic value Y hidden within it. Consequently, these operations establish a payment channel between the payer P_i and the payee P_j.

5.4. Phase IV: Payment Channel Update Phase

After the formal establishment of the payment channel, the transacting parties P_i and P_j proceed to the subsequent phase: the payment transactions between them. In this phase, the payer P_i redistributes the zero-knowledge currency locked within the payment channel based on the transactions between both parties, reallocating it with each transaction until completion. The fund allocation within the payment channel is executed using the Algorithm 4 payment channel update algorithm, generating a payment channel update transaction tx_{Update}. The tx_{Update} transaction contains the following variables:

- Merkle tree root rt: the proof that the commitment cm^{pc} exists in the ledger;
- Serial numbers sn^{pc} : strings associated with commitment commitment cm^{pc};
- Commitment values cm_i^{new} and cm_j^{new}: these are also generated by the commitment scheme COMM. The contents implicit in the cm_i^{new} commitment are the address pk_i, the address pk_j, the explicit value v_i^{new}, serial number sn_i^{new} associated with this commitment value generated by the PRF function, and the serial number sn^{pc}; the contents implicit in the cm_j^{new} commitment are the address pk_i, the address pk_j, the explicit value v_j^{new}, serial number sn_j^{new} associated with this commitment value generated by the PRF function, and the serial number sn^{pc};
- Address pk_i: the address of the payer;
- Zero-knowledge proof π_{Update}: this zero-knowledge proof is a proof generated by zk-SNARKs.GenProof, which is suitable for the circuit of C_{Update}.

  (1) $cm^{pc} = \text{COMM}(pk_j, pk_j, v^{old}, sn^{old}, sn^{pc}, Y)$
  (2) $sn^{pc} = \text{PRF}(pk_i, r^{old})$
  (3) $cm_j^{new} = \text{COMM}(pk_i, pk_j, v_j^{new}, sn^{pc}, sn_j^{new})$
  (4) $sn_j^{new} = \text{PRF}(pk_j, r_j^{new})$
  (5) $cm_i^{new} = \text{COMM}(pk_j, pk_i, v_i^{new}, sn^{pc}, sn_i^{new})$
  (6) $sn_i^{new} = \text{PRF}(pk_i, r_i^{new})$
  (7) $v^{old} = v_j^{new} + v_i^{new}$
  (8) The path $path$ from cm^{pc} to the rt saved on the ledger is correct

- Signature σ_{Update}: signature of the above (rt, π_{Update}, cm_i^{new}, cm_j^{new},sn^{pc}) using the private key sk_i.
- Discrete logarithmic value Y and secret value y: $Y = g^y mod N$ (where g is the common parameter).

Algorithm 4 Payment Channel Update Algorithm

Input: The public parameter list pp, c^{pc}, Merkle tree root rt, path $path$, Private key sk_i owned by P_i, plaintext values v_j^{new} and v_i^{new}

Output: Zero-knowledge currency c_i^{new} and c_j^{new}, payment channel update transaction tx_{Update}

1: Parse c^{pc} :=$(cm^{pc}, pk_j, pk_i, v^{old}, sn^{old}, sn^{pc}, Y, r^{old})$
2: Compute the random numbers r_i^{new} and r_j^{new}, and compute serial number $sn_j^{new} = PRF\left(pk_j, r_j^{new}\right)$ and $sn_i^{\text{new}} = PRF\left(pk_i, r_i^{\text{new}}\right)$
3: Compute the new commitment value cm_i^{new} =COMM (pk_j, pk_i, v_i^{new}, sn^{pc}, sn_i^{new}) and cm_j^{new} = COMM (pk_i, pk_j, v_j^{new}, sn^{pc}, sn_j^{new})
4: Set c_i^{new} :=$(cm_i^{new}, pk_j, pk_i, v_i^{new}, sn^{pc}, sn_i^{new}, r_i^{new})$ and c_j^{new} := $(cm_j^{new}, pk_i, pk_j, v_j^{new}$, $sn^{pc}, sn_j^{new}, r_j^{new})$
5: Set the information that needs to be disclosed to generate a zero-knowledge proof x :=$(rt, cm_j^{new}, cm_i^{new}, pk_i, sn^{pc}, Y)$
6: Set hidden evidence w :=$(path, cm^{pc}, pk_j, v^{old}, v_j^{new}, v_i^{new}, sn_j^{new}, sn_i^{new}, sn^{old}, r_j^{new}, r_i^{new}$, $r^{old})$
7: Compute π_{Update} = zk-SNARKs.GenProof $(pk_{\text{zUpdate}}, x, w)$
8: Set $m = (rt, \pi_{Upadate}, cm_j^{new}, cm_i^{new}, sn^{pc}, Y)$
9: The signature $\sigma_{Upadate} = BLS.Sign(m, sk_i)$ is generated for message m using the private key sk_i.
10: Set $tx_{\text{Update}} = (m, pk_i, \sigma_{Upadate})$
11: Output c_i^{new}, c_j^{new} and $tx_{Upadate}$.

The detailed procedure of the payment channel update algorithm is shown above.

After the payer P_i completes the payment channel update algorithm, they are required to send the new zero-knowledge currency c_j^{new} and the payment channel update transaction $tx_{Upadate}$ to the payee P_j. Upon receiving this information, the payee P_j first verifies the correctness of the signature $\sigma_{Upadate}$ included in the payment channel update transaction $tx_{Upadate}$. Subsequently, utilizing the secret value y, they perform the complete payment channel update transaction $tx_{Upadate}$. Following this process, the payee P_j has the capability to close the payment channel at any time.

5.5. Phase V: Payment Channel Closure Phase

After both parties of a payment channel have completed their transaction, the payee can close the channel by posting the latest updated transaction without communicating with the payer . When the payee P_j posts the latest payment channel update transaction $tx_{Upadate}$, they can immediately receive the due currency. At the same time, the payer P_i can also obtain the remaining money immediately. However, since the zero-knowledge currency owned by the payee P_j that has been promised to be generated by the payer P_i, this also needs to be transferred immediately from the zero-knowledge currency to a new address, in order to protect the security of the payee's funds. This is accomplished using the Algorithm 5 transfer algorithm that generates a transfer transaction $tx_{Transfer}$ that transfers the zero-knowledge currency c_j^{old} received by the payee P_j to a new address pk_j^{new}, where the $tx_{Transfer}$ transaction consists of the following variables:

- Merkle tree root rt: This is the proof that the commitment cm_j^{old} exists in the ledger;
- Commitment serial number sn_j^{old}: A unique string associated with the commitment cm_j^{old};
- Commitment value cm_j^{new}: This value is generated by the commitment scheme COMM, and the content implied in the commitment includes the address pk_j^{new}, the transferred

value v_j, new random numbers r_j^{new}, and the serial number sn_j^{new} associated with this commitment value generated by the PRF function;

- Address pk_j: the address of user P_j ;
- Zero-knowledge proof π_{Transfer}: this zero-knowledge proof is a proof generated by zk-SNARKs.GenProof that the following conditions apply to the circuit of C_{Transfer}:
  (1) cm_j^{old} =COMM($pk_i, pk_j, v_j, sn^{pc}, sn_j^{old}, Y$)
  (2) sn_j^{old} =PRF(pk_j, r_j^{old})
  (3) cm_j^{new} =COMM($pk_j^{new}, v_j, r_j^{new}, sn_j^{new}$)
  (4) sn_j^{new} =PRF(sk_j^{new}, r_j^{new})
  (5) The path $path_j$ from cm_j^{old} to the rt saved on the ledger is correct
- Signature σ_{Transfer}:Signature of the above (π_{Transfer}, rt, cm_j^{new}, sn_j^{old}) using the payment channel private key sk_j.

Algorithm 5 Transfer Algorithm

Input: The public parameter list pp, c_j^{old}, Merkle tree root rt and path $path_j$, public key pk_j of P_j , new public key pk_j^{new} of P_j and new private key sk_j^{new} of P_j

Output: New zero-knowledge currency c_j^{new},transfer transaction tx_{Transfer}

1: Parse $c_j^{old} := (cm_j^{old}, pk_i, pk_j, v_j, sn^{pc}, sn_j^{old}, r_j^{old})$
2: Select new random number r_j^{new} and compute serial number sn_j^{new} = PRF(sk_j^{new}, r_j^{new})
3: Compute new commitment value cm_j^{new} = COMM(v_j, pk_j^{new}, sn_j^{new}, r_j^{new})
4: Set $c_j^{new} := (cm_j^{new}, pk_j^{new}, v_j, sn_j^{new}, r_j^{new}, sk_j^{new})$
5: Set the information that needs to be disclosed to generate a zero-knowledge proof x: = (rt, cm_j^{new}, pk_j, sn_j^{old},)
6: Set hidden evidence w: = ($path_j$, cm_j^{old}, pk_i, pk_j^{new}, v_j, sn^{pc}, sn_j^{new}, r_j^{old}, r_j^{new}, sk_j^{new})
7: Compute π = zk-SNARKs.GenProof($pk_{\text{zTransfer}}$,x, w)
8: Set $m := (\pi_{\text{Transfer}}, rt, cm_j^{new}, sn_j^{old})$
9: User P_i uses private key sk_j to generate signature σ_{Transfer} =BLS.Sign(m, sk_j) for m.
10: Set $tx_{\text{Transfer}} := (m, pk_j, \sigma_{\text{Transfer}})$
11: Output zero-knowledge currency c_j^{new},transfer transaction tx_{Transfer}

The detailed procedure of transfer algorithm is shown above.

6. Analysis

In this section, we analyze how this anonymous payment channel scheme based on zk-SNARKs achieves the design goals presented in Section 4.3.

Unlinkability: In Section 4.3, we designed unlinkability to address the lack of linkability between the two parties involved in the payment channel establishment send transaction, the payment channel update transaction, and the transfer transaction mentioned in the aforementioned scheme. To prove the unlinkability of this scheme, we conduct a formal security proof by executing the game $\mathcal{GUL}$.

We abstract the blockchain network within the payment channel as an oracle $\mathcal{X}$, providing an interface to interact with the operations defined in zk-APC. The blockchain ledger $\mathcal{L}$ that stores and manages all transactions is overseen by $\mathcal{X}$. Assume a probabilistic polynomial-time adversary $\mathcal{A}$ capable of querying $\mathcal{X}$ during the game $\mathcal{GUL}$. $\mathcal{X}$ responds to each query with $\mathcal{L}$ until $\mathcal{A}$ submits a transaction tuple $(tx, tx^*) \in \mathcal{L}$ satisfying conditions (a) tx and tx^* being of the same type, i.e., all three transaction types mentioned earlier, (b) $tx \neq tx^*$, and (c) $\mathcal{A}$ are not involved in either tx or tx^*. If one of the following holds: (a) for payment channel establishment transactions, both tx and tx^* involve the same sender and receiver; (b) for payment channel updating transactions or transfer-transactions, tx and tx^* involve the same sender and recipient; or (c) for payment channel renewal

transactions or transfer-transactions, tx and tx^* do not involve the same recipient, $\mathcal{X}$ outputs 1, indicating that $\mathcal{A}$ has won the game $\mathcal{GUL}$. Consequently, we can formalize the following proposition:

Theorem 1. *The zk-APC scheme preserves transaction unlinkability because, for any probabilistic polynomial time adversary $\mathcal{A}$, the following probability is negligible under security parameter λ:*

$$\Pr[\mathcal{GUL}(\lambda, \mathcal{A})] = 1$$

Proof. (a) Assume that $\mathcal{A}$ outputs a tuple of payment channel establishment transactions $(tx_{\text{SetSend}}, tx'_{\text{SetSend}})$ where tx_{SetSend} satisfies that

$$tx_{\text{SetSend}} := (rt, cm^{pc}, sn_i^{old}, \pi_{SetSend})$$
$$cm^{pc} = \text{COMM}(pk_i, pk_j, v_i, sn_i^{old}, sn^{pc}, Y)$$
$$cm_i^{old} = \text{COMM}(pk_i, v_i, sn_i^{old}, r_i^{old})$$

and $tx_{\text{SetSend}}{}'$ satisfies that

$$tx'_{\text{SetSend}} := (rt', cm^{pc\prime}, sn_i^{old\prime}, \pi'_{SetSend})$$
$$cm^{pc\prime} = \text{COMM}(pk'_i, pk'_j, v'_i, sn_i^{old\prime}, sn^{pc\prime}, Y')$$
$$cm_i^{old\prime} = \text{COMM}(pk_i{}', v_i{}', sn_i^{old\prime}, r_i^{old\prime})$$

To win the game $\mathcal{GUL}$ experiment, $\mathcal{A}$ finds a set of transactions $(tx_{\text{SetSend}}, tx'_{\text{SetSend}})$ for which the receiver $pk_j = pk'_j$ and the sender $pk_i = pk'_i$.

To achieve the goal of finding identical receivers, $\mathcal{A}$ can determine whether pk_j and pk'_j are equal in two ways:

(a) Obtaining the recipient's address pk_j (or pk'_j) from cm^{pc} (or $cm^{pc\prime}$).

(b) Extracting pk_j (or pk'_j) from zk-SNARK proof π_{SetSend} (or π'_{SetSend}).

For (a) $\mathcal{A}$ must distinguish the (pk_j,pk'_j) contained in (cm^{pc},$cm^{pc\prime}$) without knowing the secret values of the (cm^{pc},$cm^{pc\prime}$), which would imply that the hidden nature of the *COMM* is destroyed, something which corresponds to $\mathcal{A}$ is impractical. For (b), since $\mathcal{A}$ cannot break the zero-knowledge property of zk-SNARKs mentioned in Section 3, the receiver address cannot be obtained from the zero-knowledge proof either.

To achieve the objective of finding the same sender, $\mathcal{A}$ can also determine it in three ways: (a) distinguishing sender addresses from the zk-SNARKs proof; (b) $\mathcal{A}$ initially, searching for commitment values (cm_i^{old}, $cm_i^{old\prime}$) used in the SetSend transaction from its view, then distinguishing the sender utilizing the *Mint* transaction associated with *cm*; (c) distinguishing the sender addresses from cm^{pc} or ($cm^{pc\prime}$).

For mode (a), $\mathcal{A}$ has to distinguish (pk_i,$pk_i{}'$) from the different zero-knowledge proofs (π_{SetSend} ,π'_{SetSend}), which implies that *mathcalA* needs to destroy the zero-knowledge property of zk-SNARK.

For method (b), $\mathcal{A}$ can differentiate the sender (pk_i, $pk_i{}'$) without knowledge of the other secret values in (cm_i^{old}, $cm_i^{old\prime}$) through examining previous transactions (tx_{Mint}, tx'_{Mint}) that include (cm_i^{old}, $cm_i^{old\prime}$). However, as cm_i^{old} and $cm_i^{old\prime}$ are not visible in tx_{SetSend} and tx'_{SetSend}, $\mathcal{A}$ must obtain cm_i^{old} and $cm_i^{old\prime}$ through two means: (1) Merkle tree root; (2) zk-SNARK proof. For method (1), $\mathcal{A}$ needs to extract (cm_i^{old}, $cm_i^{old\prime}$) from the Merkle tree root (rt, rt'), requiring a breach in the collision-resistant hash (CRH) property. For method (2), $\mathcal{A}$ must extract (cm_i^{old}, $cm_i^{old\prime}$) from the zk-SNARKs proof (π_{SetSend}, π'_{SetSend}), which implies a breach in the zero-knowledge property of zk-SNARKs. Regarding method (c), for the same reasons mentioned, this violates the hiding property of COMM. Therefore, the

commitment scheme, hash function, and zk-SNARKs security features guarantee that the sender and receiver of a transaction cannot be distinguished in a tx_{SetSend} transaction.

(b) Assume that $\mathcal{A}$ outputs a tuple of $(tx_{\text{Update}}, tx'_{Update})$, where tx_{Update} satisfies that

$$\begin{aligned} tx_{\text{Update}} &= (rt, \pi_{\text{Update}}, cm_j^{\text{new}}, cm_i^{\text{new}}, pk_j, sn^{pc}, Y, \sigma_{Update}) \\ cm^{pc} &= \text{COMM}\Big(pk_j, pk_i, v^{\text{old}}, sn^{\text{old}}, sn^{pc}, Y\Big) \\ cm_j^{\text{new}} &= \text{COMM}\Big(pk_i, pk_j, v_j^{\text{new}}, sn^{pc}, sn_j^{\text{new}}\Big) \\ cm_i^{\text{new}} &= \text{COMM}\big(pk_j, pk_i, v_i^{\text{new}}, sn^{pc}, sn_i^{\text{new}}\big) \end{aligned}$$

and tx'_{Update} satisfies that

$$\begin{aligned} tx'_{\text{Update}} &= (rt', \pi'_{\text{Update}}, cm_j^{new'}, cm_i^{new'}, pk'_i, sn^{pc'}, Y', \sigma'_{\text{Update}}) \\ cm^{pc'} &= \text{COMM}\Big(pk'_j, pk'_i, v^{old'}, sn^{old'}, sn^{pc'}, Y'\Big) \\ cm_j^{new'} &= \text{COMM}\Big(pk'_i, pk'_j, v_j^{new'}, sn^{pc'}, sn_j^{new'}\Big) \\ cm_i^{new'} &= \text{COMM}\Big(pk'_j, pk'_i, v_i^{new'}, sn^{pc'}, sn_i^{new'}\Big) \end{aligned}$$

To win the $\mathcal{GUL}$ experiment, $\mathcal{A}$ must identify a set of transactions $(tx_{Update}, tx'_{Update})$ with the recipient $pk_j = pk'_j$. To achieve this goal, $\mathcal{A}$ can also make determinations through two methods: (a) Distinguishing the recipient's address from the zk-SNARKs proofs. (b) Discerning the recipient's address from cm_j^{new}, cm_i^{new} (cm_j^{new}), and ($cm_i^{new'}$).

For (a), $\mathcal{A}$ must distinguish (pk_j, pk'_j) from different zero-knowledge proofs (π_{Update}, π'_{Update}), which implies that $\mathcal{A}$ needs to break the zero-knowledge property of zk-SNARK.

For (b), for the same reasons mentioned earlier, this would violate the hiding property of the commitment scheme (COMM). Therefore, due to the security properties of the commitment scheme, hash functions, and zk-SNARK, this ensures that in the tx_{Update} transactions, it is impossible to determine the receiver of the transactions. The same applies to the corresponding transfer transactions of the same type.

Therefore, Theorem 1 is proved to be correct. □

Privacy Preserving: In the scheme proposed within this paper, each user involved in a transaction adopts anonymous addresses, and the verification only involves these anonymous addresses. This prevents the disclosure of users' real identities, as no relevant identity information can be obtained from transactions by anyone other than these anonymous addresses. Therefore, the scheme proposed in this paper ensures users' identity privacy. The zk-SNARKs-based anonymous payment channel also safeguards the privacy of transaction amounts for both parties involved in the payment channel establishment, sending transactions, payment channel updating transactions, and transfer transactions mentioned in this paper: on one hand, the substantial collision hash property of commitments keeps the transaction amount hidden from outsiders; on the other hand, due to the zero-knowledge nature of zk-SNARKs, attackers cannot extract any information about the transaction values from the zero-knowledge proofs. Additionally, this scheme protects the privacy of users' balance values. Each user's balance can be split into a plaintext balance and a zero-knowledge balance. Attackers could obtain the plaintext balance through ledger inspection. However, the zero-knowledge balance is secretly stored using hash key-value pairs on the blockchain's Merkle tree. Upon revealing its serial number, the zero-knowledge balance is consumed and replaced by a new zero-knowledge balance, consequently replacing its balance commitment with a new commitment. Due to the substantial collision resistance property of the balance commitment's hash function, attackers cannot deduce specific amounts from the balance commitment and the disclosed serial number. Therefore, attackers cannot obtain the user's current balance values.

7. Experiment and the Results

In this section, we describe our comprehensive evaluation of the zk-SNARK-based anonymous payment channel and present experimental results to demonstrate the feasibility and performance of the scheme.

7.1. Experiment Configuration

To benchmark the protocol's performance, we implemented our protocol using the C++ programming language, the Libsnark library, and the GMP library, where Libsnark is a library for implementing zk-SNARKs schemes in C++.

To implement zk-SNARKs in the zk-SNARKs-based anonymous payment channel, we utilized the Libsnark library [33] to develop functions for generating and verifying zk-SNARK proofs. For zk-SNARKs implemented based on the Libsnark library, we chose ALT_BN128 as the default elliptic curve and used various zero-knowledge proof schemes, including Groth16 [34], GM17 [35], and PGHR13 [36]. Each blockchain node generated and pre-installed the key pair (pk, vk) for zk-SNARK proof generation/verification. Additionally, the hash function COMM for constructing Merkle trees and generating commitments, the hash function, and the pseudo random function (PRF) we used were instantiated using the SHA-256 hash function. To implement BLS signatures [37], we developed and implemented a BLS signature algorithm based on the GMP library. According to the workflow of the zk-SNARKs-based anonymous payment channel, we conducted a comprehensive evaluation, primarily including the following components:

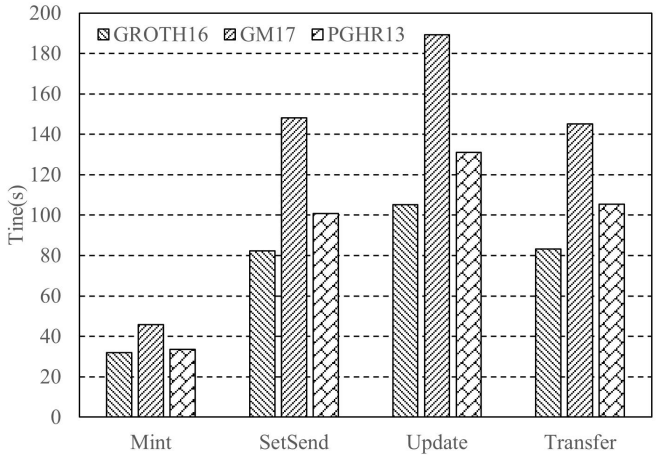
- We conducted a performance evaluation of the zk-SNARKs circuits for the $Mint$, $SetSend$, $Update$, and $Transfer$ aspects of the zk-SNARKs-based anonymous payment channel;
- We compared the zk-SNARKs-based anonymous payment channel with similar protocols, such as Blockmaze, Zerocash, and DMC [38];
- We evaluated the performance of the three phases of the payment channel, payment channel establishment, payment, and payment channel closure, and compared them with DMC.

All experiments in this paper were conducted on an Ubuntu Linux 16.04 LTS machine, equipped with an AMD Ryzen 7 5800H @ 3.20GHz CPU and 16 GB RAM.

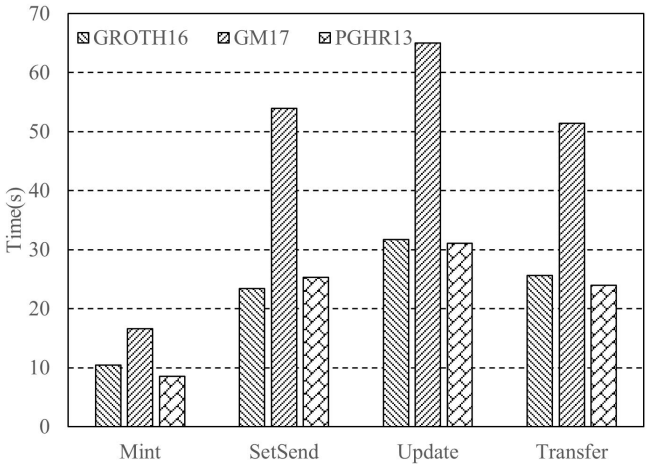
7.2. Experiment Results

7.2.1. zk-SNARKs Performance Evaluation

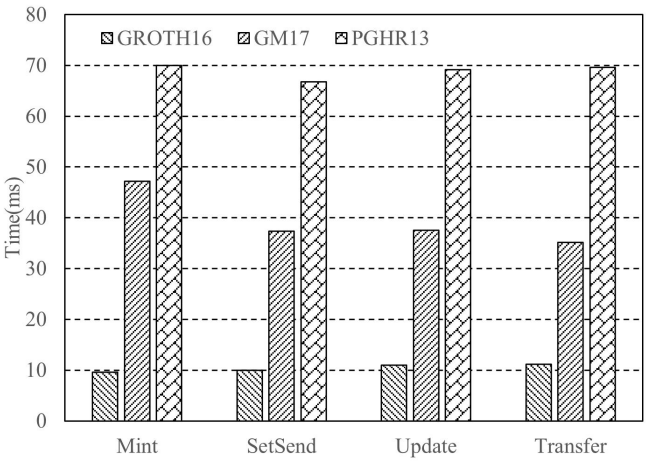
To determine the most suitable zk-SNARKs scheme for zk-APC, this paper evaluated the performance of PGHR13, Groth16, and GM17 schemes regarding computation and storage. Figure 1 shows a performance comparison of these three schemes for the $Mint$, $SetSend$, $Update$, and $Transfer$ circuits regarding five aspects: setup time, size of proof/verification keys, proof generation time, and verification time. As seen in Figure 1, Groth16 had significant advantages over PGHR13 and GM17, especially regarding the proof verification time, size of proof/verification keys, and setup time. In the $Mint$ circuit, the proof verification time for the GM17 scheme was 4.9-times that of Groth16, and for the PGHR13 scheme, it was 7.3-times that of Groth16. In the $Update$ circuit, the proof key size for GM17 was 2.4-times that of Groth16, and for PGHR13, it was 1.45-times that of Groth16. In the $SetSend$ circuit, the verification key size for GM17 was 2.4-times that of Groth16, and for PGHR13, it was 1.45-times that of Groth16. In the $Transfer$ circuit, the proof generation time for Groth16 and PGHR13 was almost the same, while for GM17, it was twice that of Groth16. Overall, the Groth16 scheme was the most suitable for the computation time and space requirements. Hence, all subsequent experiments in this paper utilized the Groth16 scheme.



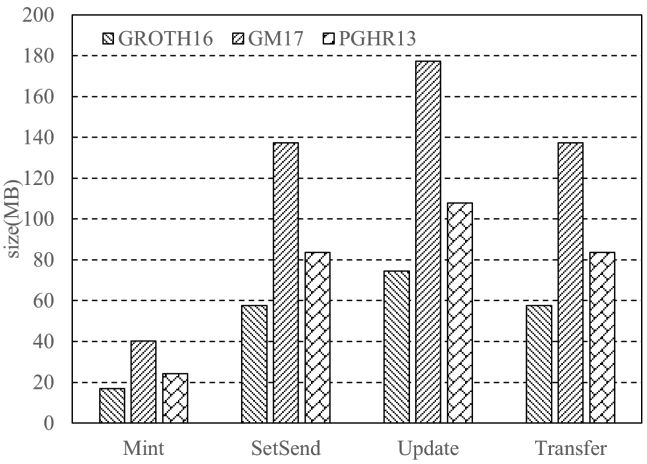
(**a**) Setup time



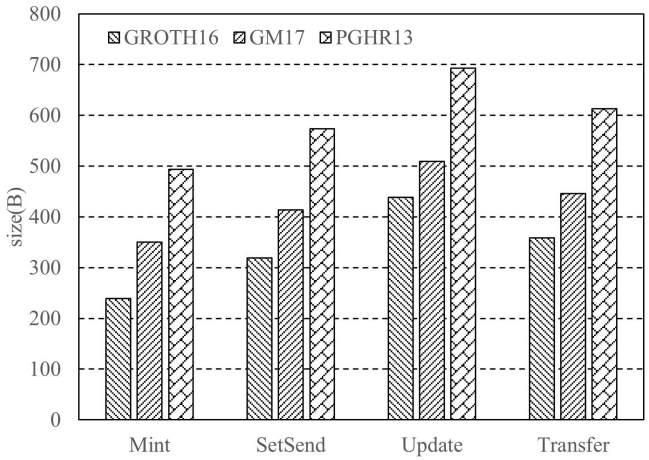
(**b**) Proof generation time



(**c**) Proof verification time



(**d**) Proving key size

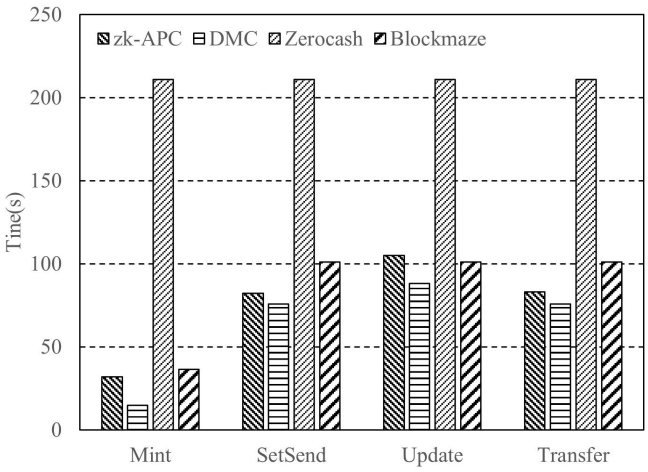


(**e**) Verification key size

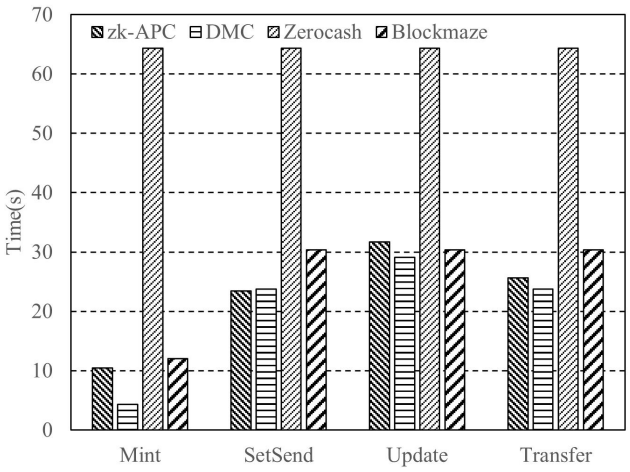
Figure 1. The performance of zk-SNARKs used in zk-APC.

7.2.2. Comparison with Blockmzae, Zerocash, and DMC

Figure 2 illustrates a comparison of our privacy protection scheme with other privacy protection schemes regarding five aspects: setup time, size of proof/verification keys, proof generation time, and verification time for zk-SNARKs. Although Blockmaze, Zerocash, and DMC differ from our scheme in certain scenarios, they all employ similar methods, specifically zk-SNARKs, for privacy protection. From an algorithmic functionality perspective, the *Mint* operation in zk-APC corresponds to Blockmaze's *Mint*, and zk-APC's *SetSend*, *Update*, and *Transfer* are similar to Blockmaze's *Deposit*. zk-APC's *Mint*, *SetSend*, *Update*, and *Transfer* are akin to Zerocash's *Pour*, and zk-APC's *Mint* is comparable to DMC's *Deposit*. zk-APC's *SetSend* and *Transfer* are analogous to DMC's *OpenChannel*, and zk-APC's *Update* is similar to DMC's *OffchainTransfer*. Compared to these three schemes, our zk-APC scheme demonstrated advantages over Blockmaze for *Mint* regarding setup time, size of proof/verification keys, and proof generation time. In the comparison of *SetSend*, *Update*, *Transfer*, and *Deposit*, *Update* and *Transfer* showed inferior performance for the proof verification time. Compared to Zerocash, our *Mint* required zero-knowledge operations, which were unnecessary in Zerocash. However, our *SetSend*, *Update*, and *Transfer* had significant computational and storage cost advantages over Zerocash's *Pour*, with the *Pour* circuit's proof generation time being 2.02-times that of the *Update* circuit. Compared to DMC, our *SetSend* circuit corresponding to DMC's *OpenChannel*, *Transfer* to DMC's *OpenChannel*, and *Update* to DMC's *OffchainTransfer* had minor differences in the five aspects mentioned above. However, the *Mint* circuit, corresponding to DMC's *Deposit*, required more time for setup, proof key size, and proof generation. Specifically, zk-APC's *Mint* circuit setup time was 2.13-times that of DMC's *Deposit*, and the proof generation time for zk-APC's *Mint* was 2.4-times that of DMC's *Deposit*, with the proof key size being double. Although there were certain disadvantages for these three aspects, they are typically only required for the initial setup or performed offline, having a minimal impact on the overall scheme. In summary, compared to BlockMaze, Zerocash, and DMC, zk-APC exhibited notable efficiency in the computational and storage costs of various zero-knowledge proof operations.

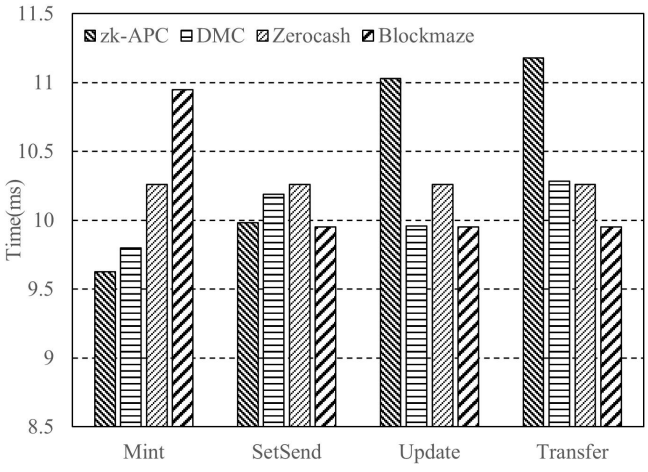


(**a**) Setup time

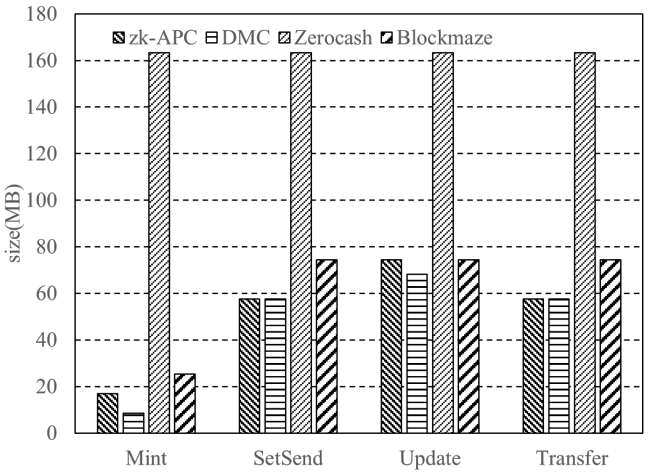


(**b**) Proof generation time

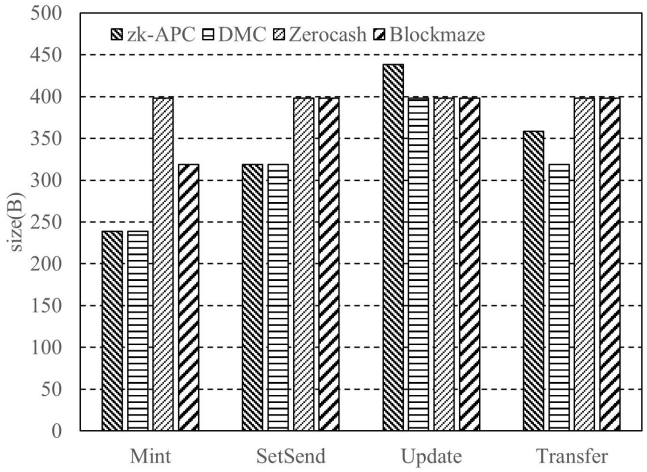
Figure 2. *Cont.*



(**c**) Proof verification time



(**d**) Proving key size



(**e**) Verification key size

Figure 2. Comparison with Blockmaze, Zerocash, and DMC.

7.2.3. Payment Channel Performance Evaluation

Comparison with DMC. Table 1 compares our scheme with other existing schemes in terms of on-chain storage overheads during the establishment and updating of payment channels, as well as off-chain communication costs. Compared to the similar scheme DMC, during the channel establishment process, the transaction size to be stored on-chain in our scheme was 0.78-times that of DMC. However, our scheme requires off-chain communication overheads, while DMC's scheme needs almost no off-chain communication. In the payment channel updating process, the off-chain communication cost of our scheme was 2.4-times that of DMC's.

Table 1. On-chain Storage Overhead and Off-chain Communication Overhead.

	Channel Opening (On-Chain)	**Channel Opening (Off-Chain)**	**Channel Update**
zk-APC	224B	5062B	634B
DMC	288B	-	256B

Computation cost. For the three stages of the payment channel, we evaluated the time consumed in each stage. In the zk-APC scheme, each stage incurred significant time costs due to zero-knowledge proofs and time-bound verifiable commitments. Our measurement of time disregarded communication time. According to our measurements, the initiator needed 36.1 seconds for the payment channel establishment process. In the payment channel updating process, each update took 31.5 s. Compared to similar Zcash shielded transactions with 6.6 tps, if our scheme was used, the transaction throughput per second could be increased to 0.032D tps, where D is the number of channels opened on the blockchain. If there are 10,000 payment channels open on the blockchain, our scheme can provide a transaction throughput of 32 tps per second.

8. Conclusions

In this paper, we proposed a zk-SNARKs-based anonymous payment channel, which solves the privacy problem during off-chain transactions between payers and payees through utilizing commitment schemes, zk-SNARKs, and timed verifiable commitments. Specifically, our approach achieves on-chain and off-chain amount privacy during payment channel transactions and unlinkable relationships between payers and payees, while ensuring the fairness of the transaction process. In this paper, we provided a concrete construction of the zk-SNARKs-based anonymous payment channel and performed security analysis and experimental verification.

Author Contributions: Conceptualization, Y.G.; methodology, Y.G. and K.G.; software, Y.G.; validation, Y.G.; formal analysis, H.L.; investigation, H.L.; resources, H.L.; writing — original draft, Y.G.; writing — review and editing, H.L., L.Z. and K.G.; supervision, L.Z. and K.G.; Funding acquisition, K.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Defense Basic Scientific Research program of China under grant number JCKY2020602B008.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: No new data were created or analyzed in this study. Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Bus. Rev.* **2008**, *PP*, 1.
2. Zheng, Z.; Xie, S.; Dai, H.N.; Chen, X.; Wang, H. Blockchain challenges and opportunities: A survey. *Int. J. Web Grid Serv.* **2018**, *14*, 352–375. [CrossRef]
3. Zhang, Y.; Gai, K.; Xiao, J.; Zhu, L.; Choo, K.K.R. Blockchain-empowered efficient data sharing in Internet of Things settings. *IEEE J. Sel. Areas Commun.* **2022**, *40*, 3422–3436. [CrossRef]
4. Monrat, A.A.; Schelén, O.; Andersson, K. A survey of blockchain from the perspectives of applications, challenges, and opportunities. *IEEE Access* **2019**, *7*, 117134–117151. [CrossRef]
5. Gao, W.; Hatcher, W.G.; Yu, W. A survey of blockchain: Techniques, applications, and challenges. In Proceedings of the 2018 27th international conference on computer communication and networks (ICCCN), Hangzhou, China, 30 July–2 August 2018; pp. 1–11.
6. Gai, K.; She, Y.; Zhu, L.; Choo, K.K.R.; Wan, Z. A blockchain-based access control scheme for zero trust cross-organizational data sharing. *ACM Trans. Internet Technol.* **2023**, *23*, 1–25. [CrossRef]
7. Gai, K.; Wang, S.; Zhao, H.; She, Y.; Zhang, Z.; Zhu, L. Blockchain-Based Multisignature Lock for UAC in Metaverse. *IEEE Trans. Comput. Soc. Syst.* **2022**, *10*, 2201–2213. [CrossRef]
8. Liang, H.; Guo, Y.; Gai, K. A Blockchain-Based Hierarchical Storage Method for Supply Chain Data. In Proceedings of the 2023 IEEE 8th International Conference on Smart Cloud (SmartCloud), Tokyo, Japan, 16–18 September 2023; pp. 105–110.
9. Tikhomirov, S. Ethereum: State of knowledge and research perspectives. In *Foundations and Practice of Security, Proceedings of the 10th International Symposium, FPS 2017, Nancy, France, 23–25 October, 2017, Revised Selected Papers 10*; Springer: Nancy, France, 2018; pp. 206–221.
10. Wood, G. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Proj. Yellow Pap.* **2014**, *151*, 1–32.
11. Yang, D.; Long, C.; Xu, H.; Peng, S. A review on scalability of blockchain. In Proceedings of the 2020 the 2nd International Conference on Blockchain Technology, Hilo, HI, USA, 12–14 March 2020; pp. 1–6.

12. Kim, S.; Kwon, Y.; Cho, S. A survey of scalability solutions on blockchain. In Proceedings of the 2018 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Republic of Korea, 17–19 October 2018; pp. 1204–1207.
13. Gai, K.; Hu, Z.; Zhu, L.; Wang, R.; Zhang, Z. Blockchain meets dag: A blockdag consensus mechanism. In *Algorithms and Architectures for Parallel Processing, Proceedings of the 20th International Conference, ICA3PP 2020, New York, NY, USA, 2–4 October 2020*; Proceedings, Part III 20; Springer: Copenhagen, Denmark, 2020; pp. 110–125.
14. Papadis, N.; Tassiulas, L. Blockchain-based payment channel networks: Challenges and recent advances. *IEEE Access* **2020**, *8*, 227596–227609. [CrossRef]
15. Malavolta, G.; Moreno-Sanchez, P.; Kate, A.; Maffei, M.; Ravi, S. Concurrency and privacy with payment-channel networks. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 455–471.
16. Gai, K.; Wu, Y.; Zhu, L.; Zhang, Z.; Qiu, M. Differential privacy-based blockchain for industrial internet-of-things. *IEEE Trans. Ind. Inform.* **2019**, *16*, 4156–4165. [CrossRef]
17. Gai, K.; Tang, H.; Li, G.; Xie, T.; Wang, S.; Zhu, L.; Choo, K.K.R. Blockchain-based privacy-preserving positioning data sharing for IoT-enabled maritime transportation systems. *IEEE Trans. Intell. Transp. Syst.* **2022**, *24*, 2344–2358. [CrossRef]
18. Sasson, E.B.; Chiesa, A.; Garman, C.; Green, M.; Miers, I.; Tromer, E.; Virza, M. Zerocash: Decentralized anonymous payments from bitcoin. In Proceedings of the 2014 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 18–21 May 2014; pp. 459–474.
19. Guan, Z.; Wan, Z.; Yang, Y.; Zhou, Y.; Huang, B. BlockMaze: An efficient privacy-preserving account-model blockchain based on zk-SNARKs. *IEEE Trans. Dependable Secur. Comput.* **2020**, *19*, 1446–1463. [CrossRef]
20. Wijaya, D.A.; Liu, J.K.; Steinfeld, R.; Liu, D.; Yu, J. On the unforkability of monero. In Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, Auckland, New Zealand, 9–12 July 2019; pp. 621–632.
21. Thyagarajan, S.A.K.; Bhat, A.; Malavolta, G.; Döttling, N.; Kate, A.; Schröder, D. Verifiable timed signatures made practical. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, 9–13 November 2020; pp. 1733–1750.
22. Zhou, X.; He, D.; Ning, J.; Luo, M.; Huang, X. Efficient Construction of Verifiable Timed Signatures and Its Application in Scalable Payments. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 5345–5358. [CrossRef]
23. Hearn, M.; Spilman, J. Bitcoin Contracts. 2015. Available online: https://en.bitcoin.it/wiki/Contracts (accessed on 8 October 2015).
24. Ying, N.; Wu, T.W. Xlumi: Payment channel protocol and off-chain payment in blockchain contract systems. *arXiv* **2021**, arXiv:2101.10621.
25. Xu, S.; Yuan, J.; Li, Y.; Liu, X.; Zhang, Y. Super payment channel for decentralized cryptocurrencies. In Proceedings of the 2019 IEEE Conference on Dependable and Secure Computing (DSC), Hangzhou, China, 18–20 November 2019; pp. 1–8.
26. Green, M.; Miers, I. Bolt: Anonymous payment channels for decentralized currencies. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 473–489.
27. Zhang, Y.; Long, Y.; Liu, Z.; Liu, Z.; Gu, D. Z-channel: Scalable and efficient scheme in zerocash. *Comput. Secur.* **2019**, *86*, 112–131. [CrossRef]
28. Moreno-Sanchez, P.; Blue, A.; Le, D.V.; Noether, S.; Goodell, B.; Kate, A. DLSAG: Non-interactive refund transactions for interoperable payment channels in monero. In *Financial Cryptography and Data Security, Proceedings of the 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, 10–14 February 2020*; Revised Selected Papers 24; Springer: Kota Kinabalu, Malaysia, 2020; pp. 325–345.
29. Thyagarajan, S.A.; Malavolta, G.; Schmidt, F.; Schröder, D. Paymo: Payment channels for monero. *Cryptol. ePrint Arch.* **2020**. Available online: https://eprint.iacr.org/2020/1441 (accessed on 28 December 2023).
30. Pinto, A.M. An introduction to the use of zk-SNARKs in blockchains. In *Mathematical Research for Blockchain Economy, Proceedings of the 1st International Conference MARBLE 2019, Santorini, Greece, 6–9 May 2019*; Springer: Santorini, Greece, 2020; pp. 233–249.
31. Groth, J.; Kohlweiss, M.; Maller, M.; Meiklejohn, S.; Miers, I. Updatable and universal common reference strings with applications to zk-SNARKs. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 19–23 August 2018; pp. 698–728.
32. Fuchsbauer, G. Subversion-zero-knowledge SNARKs. In *Public-Key Cryptography–PKC 2018, Proceedings of the 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, 25–29 March 2018*; Proceedings, Part I 21; Springer: Atlanta, GA, USA, 2018; pp. 315–347.
33. Ben-Saason, E.; Chiesa, A.; Genkin, D.; Kfir, S.; Tromer, E.; Virza, M. Libsnark: C++ Library for zkSNARK Proofs, 2014. Available online: https://github.com/clearmatics/libsnark (accessed on 28 December 2023).
34. Groth, J. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: Proceedings of the 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, 8–12 May 2016*; Proceedings, Part II 35; Springer: Lyon, France, 2016; pp. 305–326.
35. Groth, J.; Maller, M. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 20–24 August 2017; pp. 581–612.
36. Parno, B.; Howell, J.; Gentry, C.; Raykova, M. Pinocchio: Nearly practical verifiable computation. *Commun. ACM* **2016**, *59*, 103–112. [CrossRef]

37. Boneh, D.; Lynn, B.; Shacham, H. Short signatures from the Weil pairing. In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, 6–10 December 2001; pp. 514–532.
38. Liu, S.; Wang, J. DMC: Decentralized Mixer with Channel for Transaction Privacy Protection on Ethereum. In Proceedings of the CS & IT Conference Proceedings, Sydney, Australia, 24–25 December 2021; Volume 11.