ASOZ: Anonymous and Auditable Cryptocurrency with Enhanced Confidential Transaction

Tianjian Liu[1,2], Wei Wang[1,2],
Chang Chen[1,2], and Dawei Zhang[1,2]✉

[1] School of Computer and Information Technology, Beijing Jiaotong University
[2] Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing, China

Abstract. Decentralized payment systems have gradually received more attention in recent years. By removing the trusted intermediary used for accounting ledgers, those payment systems fundamentally empower users to control their assets. As privacy concerns grow, some cryptocurrencies are proposed to preserve the privacy of users. However, those cryptocurrencies also inadvertently facilitate illicit activities such as money laundering, fraudulent trading, etc. So it is necessary to design an auditing scheme. To solve this problem, many privacy-preserving and auditing schemes have been proposed. However, there exists no scheme that effectively solves the issue of privacy-preserving and auditing on both user identity and transaction value.

In this paper, we propose a design for a decentralized payment system named ASOZ. We use cryptographic accumulators based on Merkle trees for accounting and use a combination of Twisted ElGamal, Non-Interactive Zero-Knowledge(NIZK), Bulletproofs, and zk-SNARKs for privacy-preserving and auditing. Our scheme achieves full transaction audit in global mixing, while the additional cost introduced remains within an acceptable range, specifically an 8% increment in proof generation time and a 23% rise in verification time. Our scheme is capable of handling large-scale transaction scenarios such as designated contract markets, and offers the strongest privacy protection capabilities in coin mixer schemes.

Keywords: blockchain · cryptocurrencies · decentralized payment system· privacy-preserving· auditable· zero-knowledge proof

1 Introduction

With the development of blockchain, decentralized digital payment systems are gradually becoming the future direction of digital payment systems. Compared to centralized payment systems, decentralized digital payment systems are built on public ledgers and digital transaction schemes based on consensus algorithms. Therefore, they earn more public trust and show better development prospects.

Bitcoin [21] is the first decentralized digital payment system. It records plaintext transaction information on the public ledger, which can be easily analyzed, tracked, and monitored. Many cryptocurrencies and protocols are proposed to solve those problems, such as ZCash [4], Monero [19,20], Coinjoins [2], and so on. Essentially, those cryptocurrencies and protocols follow the concept of coin mixing. Coin mixing refers to using cryptology method or trust mechanism, to verify the correctness ofprotocol a transaction without leaking the information in a mixing list. Among them, the zerocash protocol of ZCash has the strongest mixing capability, we call it global mixing. Using membership proof of Merkle tree [23], zerocash protocol mixing sender address into a whole Merkle tree. This means if the depth of the Merkle tree is 32, then the mixing list size is 2^{32}.

However, the privacy-preserving approach mentioned above leads to certain issues. Due to the limited visibility of transaction information, those cryptocurrencies become susceptible to illegal transactions. according to the report of Chainalysis [16], the cryptocurrency received by mixers is $7.8 billion in 2022, 24% of which came from illicit addresses. In the year 2021, this percentage amounted to a mere 10%. Besides, auditing is necessary in real-world situations. For example, the Commodity Futures Trading Commission (CFTC) requires daily position reports from futures exchange members and brokers[14]. Therefore, to follow the law, decentralized payment system must find a balance between auditing and user privacy. So it is imperative to introduce audit mechanisms within the frameworks of privacy-preserving schemes. To solve those problems, many privacy-preserving and audit schemes were proposed [11,17,6,25,18]. [11] suggested a fundamental method to enhance the zerocash protocol by incorporating an auditing functionality; [6] proposed a transaction value auditing scheme based on Twisted ElGamal, but it does not support audit for identities; [25] use Pedersen Commitment and ElGamal to preserve transaction value and identities, but it lacks a design for recording transactions and relies on a conventional blockchain structure for recording; [17] builds on the design of transaction value and identities and uses cryptographic accumulators as a way to record transactions but employs a significant amount of zk-SNARKs, resulting in lower computational efficiency. Its sender address is sent in plaintext, the privacy-preserving ability is limited; [18] designs transaction records using a tabular structure based on the design of transaction values and identities, but this makes the scheme unsuitable for handling a large number of users, only effective in scenarios involving a small number of institutions within the defined consortium chain.

From above, we can see that there are some challenges in decentralized digital payment systems currently, including how to define audit capabilities, how to introduce effective audit mechanisms based on privacy-preserving schemes, how to balance the audit capabilities and scheme efficiency.

In this paper, we propose a design for a decentralized payment system with privacy-preserving and auditing. We summarize our contributions as follows:

- We propose a decentralized transaction privacy-preserving and auditing scheme called ASOZ for scenarios with large-scale users, supporting privacy-preserving

and auditing of both identities and transaction value. We formalize the system model and security properties of ASOZ, including correctness, auditability, and privacy. We outline the principles behind designing audit schemes, including offline audit, out-of-band cost, full transaction audit, minimal information disclosure, and security strength unchanged. We instantiate our scheme following the principles, and prove that the scheme achieves all the security requirements in the formalized security model.

- We use membership proof of Merkle tree to utilize a global mixing scheme for identity privacy-preserving, offering the strongest hiding capabilities among all cryptocurrency schemes. Our scheme's anonymity-set size is $2^{(n-1)}$, which n is the depth of the Merkle tree.
- We implement full transaction auditing utilizing Twisted ElGamal and ElGamal algorithms including user identity and transaction value, and combine zk-SNARK and NIZK to prove the audit reliability.

1.1 Organization

The rest of this paper is organized as follows:

In section 2, we introduce the basic concepts and notations used in this paper. In section 3, we give an overview of our scheme. In section 4, we describe the details of our scheme. In section 5, we give security proof. In section 6, we analyze the performance of our scheme. In section 7, we give further discussion.

2 Preliminary

Pedersen Commitment Below we recall Pedersen Commitment [22]:

$$comm(v, r) = g^v h^r \tag{1}$$

Where v is the value we aim to hide, r is a randomly generated number, g and h are two generators of a cyclic group G with $s = |G|$ elements and prime order p, $\mathbb{Z}_p = \{0, 1, \ldots, s-1\}, v \in \mathbb{Z}_p$, and $r \in \mathbb{Z}_p$. The Pedersen commitment provides perfect hiding and computational binding under the discrete logarithm assumption.

ElGamal Below we recall ElGamal encryption [8,9]:

Alice		**Bob**
pick a prime p		
find a primitive root g		
$\alpha \leftarrow\$ \mathbb{Z}_q \backslash \{1\}$		
$a \leftarrow g^{\alpha} \bmod$ p		
publish public key	$\xrightarrow{(p, g, a)}$	
		get messages $m \in \mathcal{M}$
		$r \leftarrow\$ \mathbb{Z}_q \mid 2 \leq r \leq p-2$
		$c_1 \leftarrow g^r \bmod$ p
		$c_2 \leftarrow m \cdot a^r \bmod$ p
	$\xleftarrow{(c_1, c_2)}$	send ciphertext
$m \leftarrow c_2 \cdot c_1^{p-1-\alpha}$		

ElGamal encryption algorithm is based on the security of Decisional Diffie-Hellman (DDH) assumption. This algorithm is IND-CPA secure. As for its friendliness to Sigma protocol, we use ElGamal encryption algorithm to encrypt the audit information in our scheme.

Twisted ElGamal Below we recall Twisted ElGamal encryption [6]:

Alice		**Bob**
pick a prime p		
find a primitive root g, h		
$\alpha \leftarrow\$ \mathbb{Z}_q \mid 2 \leq \alpha \leq p-1$		
$a \leftarrow g^{\alpha}$(mod p)		
publish public key	$\xrightarrow{(p,g,h,a)}$	
		get value $v \in \mathbb{Z}_q$
		$r \leftarrow\$ \mathbb{Z}_q$
		$c_1 \leftarrow a^r$mod p
		$c_2 \leftarrow g^r \cdot h^v$mod p
	$\xleftarrow{(c_1,c_2)}$	send ciphertext
$v \leftarrow \log_h(c_2/({c_1}^{\alpha^{-1}}))$		

In the above algorithm, the size of v is much smaller than p so that we can calculate the discrete logarithm of $c_2/({c_1}^{\alpha^{-1}})$. This algorithm supports homomorphic properties, so we directly use this algorithm to calculate transaction value in a black box manner. This algorithm is IND-CPA secure based on the divisible DDH assumption.

Sigma Protocol Sigma protocol [24] is an interactive proof protocol, it contains three parts: commitment, challenge, and response. And it is easy to convert to non-interactive proof by random oracle model. In general, Sigma protocol is more efficient than zk-SNARK.

Fiat-Shamir Heuristic The Fiat-Shamir heuristic [10] is a technique to convert an interactive protocol to a non-interactive proof in the random oracle model. This technique assumes pseudorandom function (PRF) as random oracle, and uses it to replace the random challenge from the verifier.

Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK) zk-SNARK [22] is an efficient variant of zero-knowledge proof of knowledge. It can prove an arithmetic circuit without revealing any information about the witness. In detail, it satisfies the following properties:

Completeness The honest prover can convince the verifier.

Succinctness An honestly-generated proof π has $O_\lambda(1)$ bits and the verify algorithm runs in time $O_\lambda(x)$, in which x is input language.

Soundness If the verifier accepts the proof, then the prover must know the witness.

Zero-Knowledge The verifier learn Zero-Knowledge about witness from the proof.

Bulletproofs Bulletproofs[5] is a non-interactive zero-knowledge proof protocol. It can prove inner-product argument and range proof based on Pedersen Commitment. Its range proof has a small proof size and fast speed, so it is widely used in many privacy coin such as ZCash [7], and Monero.

3 Solution Overview

Our solution is based on the zerocash framework. ZCash has created a separate blockchain for the zerocash scheme. We consider this scheme can be deployed as a smart contract on any platform, thereby forming an independent payment DApp. Our goal is to create such a system and then introduce an additional auditing structure. We want this auditing structure to comply with legal regulations, instead of the current practice in ZCash where users voluntarily choose whether to disclose transaction details.

3.1 System model

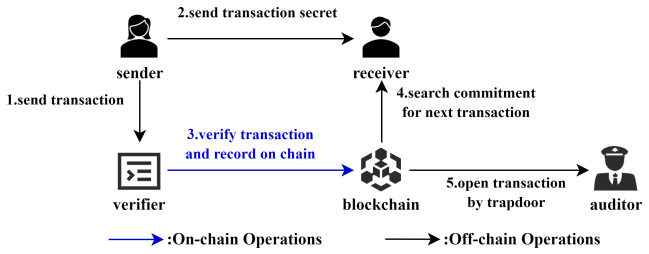


Fig. 1: System model of scheme

As shown in figure 1, five entities namely, sender, receiver, verifier, auditor, and blockchain are involved in our scheme. The base characters are defined as follows.

- **Sender**: The initiator of the transaction.
- **Receiver**: The recipient of the transaction.
- **Verifier**: The leader of updating public state in consensus algorithm, generally verifies a large number of transactions according to the protocol rules, then records the transactions in public ledger. We assume the verifier is honest but curious. In our scheme, the verification process does not require the involvement of auditors, and can be implemented as smart contracts.
- **Auditor**: As described above, we need a character to open transactions to get the identities and values. Such as the tax bureau and so on.

3.2 Challenge and Design Principle

Compared to other solutions, to meet the market's audit requirements, we wish our design to enable auditors to open all transactions without any interaction with validators. Therefore, we plan to design a scheme that uses zero-knowledge proofs to demonstrate that all transactions have left a trapdoor for auditors, and this rule is incorporated into the validators' verification process. Note that this rule does not require auditors to provide any information other than the audit key, hence it is interaction-free; and this trapdoor needs to be stored on the blockchain. However, the substantial storage and computational overhead brought by a full transaction audit is problematic, so we hope to shift the main computations to offline processing, and depend on existing privacy-preserving data structure designs to devise audit methods, reducing any additional storage introduced as much as possible. More specifically, we propose principles for the design of the auditing scheme:

Offline audit The auditing scheme should allow offline auditing, the regulatory entity is not required to participate in the consensus protocol.

Out-of-band Cost The auditing scheme should use the data structure provided by the privacy-preserve scheme whenever possible, with the aim of minimizing additional storage, computation, and interaction costs.

Full transaction audit In the auditing scheme, the auditor should have the ability to open all transactions and obtain information about all participants and value, we refer to this auditing scheme as a full transaction audit scheme. Under a full transaction audit scheme, the auditor can track the flow of all funds and the individuals involved.

Minimal information disclosure When the auditor opens private data for auditing, he should strive to minimize unnecessary disclosures of privacy.

Security Strength Unchanged The introduction of audit measures will not result in a diminution of the privacy-preserving strength in the original privacy-preserving schemes.

Below we recall the structure of zerocash scheme with our improvement.

3.3 How to Preserve Privacy

Zerocash is a decentralized anonymous payment system based on UTXO. As shown in Figure 2, zerocash uses zk-SNARK to prove the transaction on the Merkle tree[23], without revealing the specific path, thus confusing the commitment of a single transaction within the entire user set. To prevent double-spending, zerocash defines a 'nullifier' to identify coins that have already been spent. That means when a coin is spent, its owner needs to present the nullifier of that coin, and miners check if the nullifier of the coin appears in the list of already spent items. When a user makes a transfer, zerocash refers to it as a 'pour': The user signs the coin with their private key, generating a nullifier. he then generates a new coin using the recipient's public key and a random number, generating a commitment.

According to the Zerocash protocol, PRF^k and PRF^{cm} need to be pseudo-random functions, and in zerocash protocol they use sha-256 hash. Here, we will

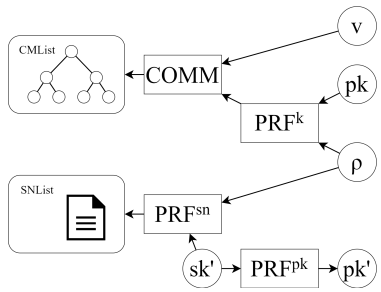


Fig. 2: zerocash scheme

present our specific construction.

$$\begin{aligned} k &= PRF^{k}(pk, \rho) = g^{pk}h^{\rho} \\ cm &= PRF^{cm}(v, k) = g^{k}h^{v} \end{aligned} \tag{2}$$

Where:

- v is the value of the coin.
- ρ provides the right to use the coin, and introduce randomness into commitment.
- pk provides ownership of the coin.

This simple construction has the following features:

- **Friendly to Range Proof:** When a user makes a payment, they need to prove their solvency, i.e., after spending coins, their balance should be greater than 0. 'cm' refers to a Pedersen commitment, which can be proved for a range using Bulletproofs[5].
- **Friendly to Balance Proof:** When the user spends old coins and pours new coins, the old coins' sum should equal the new ones' sum. This is easy to achieve. Suppose the old coins' commitment is $cm_{old} = g^{k_{old}}h^{v_{old}}$, the new coins' commitment is $cm_{new} = g^{k_{new}}h^{v_{new}}$, he only needs to calculate $r_{equation} = k_{old} - k_{new}$ and display ($r_{equation}$, cm_{old}, cm_{new}). Anyone can check its balance by verifying $cm_{old}^{-1} \cdot cm_{new} \cdot g^{r_{equation}}$.
- **Friendly to Value Transmitting:** Since cm is in the form of a Pedersen commitment, the amount of the transaction can be transmitted on the blockchain using the Twisted ElGamal encryption algorithm, without the need for an additional secure channel.

3.4 How to Audit

Now, we will provide the specific construction of PRF^{sn} and PRF^{pk}:

$$\begin{aligned} sn = PRF^{sn}(\rho, sk) &= g^{sk}h^{\rho} \\ pk &= g^{sk} \end{aligned} \tag{3}$$

The function PRF^{sn} needs to be collision-resistant, once the resistance exists, the spending of later coins will fail. We instantiate PRF^{sn} as the form of Pedersen commitment, because of its uniformly random distribution, the probability of collision is negligible.

We would like to emphasize here that, Although both CMList and SNList are membership lists that require public maintenance to ensure their immutability. However, the CMList needs to prove that the coin is in the Merkle tree structure in the commitment list when the specific content of the coin is not visible; SNList is simply a searchable list, which can be a structure such as B+ tree, Bloom Filter, etc.

According to the prior definition, our auditing refers to the process in which the verifier leaves a Trapdoor in the encrypted transaction for the auditor before the transaction is confirmed. The auditor can open the encrypted transaction and access the transaction value and identities.

The form of PRF^{cm} and PRF^{sn} is similar to the form of TwistElGamal and ElGamal output. (v, pk, sk) is the secret parameter in verification, and (v, pk) is what needs to be audited, so if we give the:

$$\begin{aligned} TwistElGamal(v) &= (upk^k, g^k h^v) \\ ElGamal(pk) &= (g^r, pk \cdot h^\rho) \\ upk &= g^{usk} \end{aligned} \tag{4}$$

We can hence audit v in CMList and pk in SNList, leaving us only to prove the consistency of the parameters involved in the computation. We give the Sigma protocol of those proofs later in 4. Unfortunately, we cannot provide a pure Sigma protocol to prove the consistency of pk in CMList. Instead, we mix the Sigma protocol with zk-SNARK to prove it.

Besides, by using ElGamal and TwistElGamal to encrypt transaction information, our introduction of audit is IND-CPA secure, which follows the Security Strength Unchanged principle. Since the auditor can open the encrypted data on the blockchain independently. Interaction with other characters is unnecessary. This follows the Offline Audit principle.

Finally, we summarized our coin structure in figure 3.

3.5 Scheme Definition

Below we propose the definition of the scheme. Firstly, we provide a rough definition of the symbols in table 1, with more precise definitions to be given in scheme design in chapter 4. Then we will formalize the notions of correctness, auditability, and privacy via security experiments, and capture the threat with oracles.

Our scheme is composed of the following algorithms:

- $Setup : pp \leftarrow 1^\lambda$
- $CreateAddress : (sk, pk) \leftarrow U \times pp$
- $CreateAuditKey : (usk, upk) \leftarrow A \times pp$
- $CreateTran : (\pi_t, cm, sn, \rho_n) \leftarrow U_s \times (pk_r, \rho_o, sk_s, v, r)$

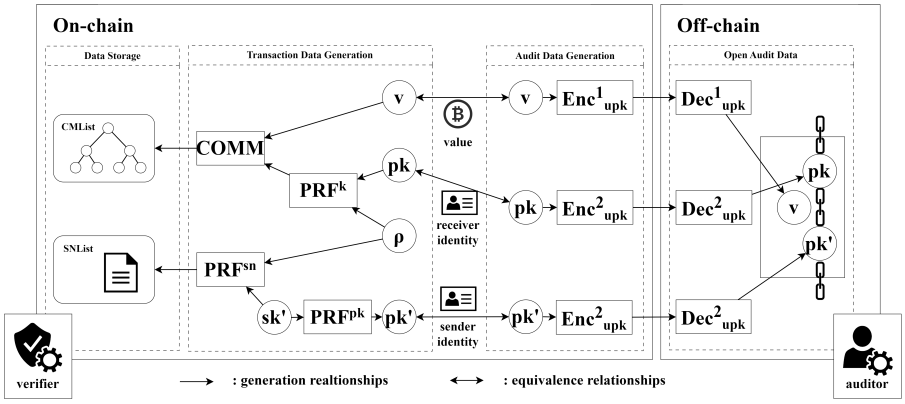


Fig. 3: Illustration of the structure of coin

- $CreateAudit : (\pi_a, s) \leftarrow U_s \times (cm, sn, upk)$
- $VerifyTran : 0/1 \leftarrow V \times (\pi_t, cm, sn)$
- $VerifyAudit : 0/1 \leftarrow V \times (\pi_a, s)$
- $AuditTran : (pk_s, pk_r, v) \leftarrow A \times (usk, cm, sn, s)$

3.6 Security Model

Let $\mathcal{A}$ be an adversary attacking our system. Formally, we capture attack behaviors as adversarial queries to oracles implemented by a challenger $\mathcal{CH}$. We list the oracles available to the adversary as below.

- $\mathcal{O}_P$: $\mathcal{A}$ queries this oracle to get a public key pk. The $\mathcal{CH}$ invoke $CreateAddress$ to get (sk, pk), then move sk, pk to the keys list T, return pk to $\mathcal{A}$. This oracle illustrates that $\mathcal{A}$ can get honest public keys.
- $\mathcal{O}_S$: $\mathcal{A}$ queries this oracle with a public key pk. $\mathcal{CH}$ first check if pk appears in corrupt list $T_{corrupt}$, if not, return a secret key sk, then $\mathcal{CH}$ move pk and sk to the corrupt list $T_{corrupt}$; if so, return $\perp$. This oracle illustrates that $\mathcal{A}$ can get a set of leaked public-secret key pairs.
- $\mathcal{O}_V$: $\mathcal{A}$ queries this oracle with $(pp, pk_r, \rho, sk_s, v, r)$, $\mathcal{CH}$ first check if pk_r or sk_s appears in $T_{corrupt}$, if so, return $\perp$; if not, the $\mathcal{CH}$ invoke $CreateTran$ to get (π_t, cm, sn), then $\mathcal{CH}$ invoke $CreateTran$ to get (π_a, s), finally $\mathcal{CH}$ return $(\pi_t, \pi_a, cm, sn, s)$ to $\mathcal{A}$. The oracle illustrates that $\mathcal{A}$ can use an honest public key to create a valid transaction.
- $\mathcal{O}_{RT}$: $\mathcal{A}$ queries this oracle with (cm, s) or (sn, s), $\mathcal{CH}$ first get (ρ, pk_r, v) or (ρ, sk_s) that generate cm or sn. Then $\mathcal{CH}$ check if pk_r or sk_s appears in $T_{corrupt}$, if not, return (ρ, pk_r, v) or (ρ, sk_s) above; if so, return $\perp$. This oracle illustrates a chosen plaintext attack (CPA) on transaction.

Then, we give the security model:

correctness Transactions that follow the rule of $CreateTran$ can pass $VerifyTran$.

Table 1: Symbols define

Symbol	Meaning
U	is the set of users. In this set, we define U_s as the sender, and U_r as the receiver in 3.1.
V	is the set of verifier in 3.1.
A	is the set of auditor in 3.1.
ρ	is the secret identity of coin. To ensure security, U_s needs to randomly update its value in a transaction. So we define ρ_o as the identifier of the old coins and ρ_n as the identifier of the coins to be poured. We want to point out that the effect of ρ is similar to the superposition of (ρ, r) in Zerocash [4].
v	is the transaction value, we define $\{v_{in,i}\}$ as input value from sneder, and $\{v_{out,i}\}$ as output value to receivers.
c	is the challenge parameter attached to the proof.
cm	is the commitment of the ownership.
sn	is the nullifier of the coin.
π_t	is the zero-knowledge proof (ZKP) of transaction information.
r	is the additional parameter used by sender U_s to calculate commitment cm.
(sk, pk)	is the signature key of users. And pk is the address of the users. (sk_r, pk_r) refer in particular to the key pair of receiver, and(sk_s, pk_s) refer in particular to the key pair of sender.
(usk, upk)	is the audit key pair.
s	is the additional parameter used by auditor A to decrypt transaction information. s is encrypted by upk.
π_a	is the ZKP of audit information.

$$Adv_{\mathcal{A}}^{C} = 1 - \Pr\left[\beta = 1 : \begin{array}{c} (\pi_t, cm, sn, \rho_n) \leftarrow CreateTran(pk_r, \rho_o, sk_s, v, r) \\ \beta \leftarrow VerifyTran(\pi_t, cm, sn) \end{array}\right] \tag{5}$$

auditability If transactions can pass $VerifyAudit$, then it can be opened by $AuditTran$ correctly.

We propose the following security experiment to describe the second half of the attribute above.

$$Adv_{\mathcal{A}}^{A} = 1 - \Pr\left[\begin{array}{c} pk_s = pk_s^* \\ pk_r = pk_r^* \\ v = v^* \end{array} : \begin{array}{c} (\pi_t, cm, sn, \rho_n) \leftarrow CreateTran(pk_r, \rho_o, sk_s, v, r) \\ 1 \leftarrow VerifyAudit(\pi_a, s) \\ (pk_s^*, pk_r^*, v^*, cm^*, sn^*) \leftarrow \mathcal{A}(\mathbb{Z}_p) \\ (pk_s^*, pk_r^*, v^*) \leftarrow AuditTran(usk, cm, sn, s) \end{array}\right] \tag{6}$$

privacy verifier cannot get transaction information from public data. Including transaction value and transaction identity.

Experiment 0. This experiment describe the advantage that adversary can get v, pk_r, pk_s from public data.

1. generation phase: on input a security parameter λ, run $pp \leftarrow Setup(1^\lambda)$

2. pre query phase: $\mathcal{A}$ adaptively makes queries to $\mathcal{O}_P, \mathcal{O}_S, \mathcal{O}_V, \mathcal{O}_{RT}$ finitely.
3. query phase: $\mathcal{CH}$ invoke $CreateAddress$ and get (pk_r, sk_s), $\mathcal{CH}$ random generate ρ and generate v from a smaller range, then invoke $CreateTran$, $CreateAudit$ to get $(\pi_t, \pi_a, cm, sn, s)$ and send to $\mathcal{A}$.
4. guess phase: $\mathcal{A}$ output ${pk_r}', {pk_s}', v'$

We define the advantage of $\mathcal{A}$ to guess v, pk_s, pk_r:
$Adv^v_{\mathcal{A}}(\lambda) = \Pr[v = v']$.
$Adv^s_{\mathcal{A}}(\lambda) = \Pr\big[pk_s = {pk_s}'\big]$.
$Adv^r_{\mathcal{A}}(\lambda) = \Pr\big[pk_r = {pk_r}'\big]$.
Then we define the advantage of $\mathcal{A}$:
$Adv^P_{\mathcal{A}}(\lambda) = \max\{Adv^v_{\mathcal{A}}(\lambda), Adv^s_{\mathcal{A}}(\lambda), Adv^r_{\mathcal{A}}(\lambda)\}$.

4 Scheme Design

4.1 Transaction Scheme

To explain our scheme, we take transaction with 2 input-1 output as an example. In fact, the scheme can be extended to m input-n output transactions. Sender has two coin record: $coin_1 = (\rho_1, v_1)$ and $coin_2 = (\rho_2, v_2)$, ρ is the secret identifier of coin, v is the value of coin. The sender wants to merge these two coins and send them to the receiver.

1. Sender calculate commitment $cm_1^{old} = g^{PRF^k(pk_s,\rho_1)}h^{v_1}$ and $cm_2^{old} = g^{PRF^k(pk_s,\rho_2)}h^{v_2}$
2. Sender calculate $cm' = cm_1^{old} \cdot cm_2^{old}$
3. Sender uses zk-SNARK to make membership proof: π_1, to prove commitment cm_1^{old} and cm_2^{old} in the Merkle tree of CMList and the correctness of cm'.
4. Sender calculate $sn_1 = PRF^{sn}(\rho_1, sk_s)$ and $sn_2 = PRF^{sn}(\rho_2, sk_s)$
5. Sender generates proof, to prove nullifier is generated by his private key: $\pi_2 = PoK\{(sk_s, pk_s, \rho_1, \rho_2) : pk_s = PRF^{pk}(sk_s) \bigwedge sn_1 = PRF^{sn}(\rho_1, sk_s) \bigwedge sn_2 = PRF^{sn}(\rho_2, sk_s)\}$. We use zk-SNARK to prove π_2.
6. Sender randomly generate ρ_3 and calculate new commitment of coin: $cm^{new} = g^{PRF^k(pk_r,\rho_3)}h^{v_3}$
7. Sender calculate $r_{equation}$, make it satisfy $PRF^k(pk_s, \rho_1) + PRF^k(pk_s, \rho_2) - PRF^k(pk_r, \rho_3) + r_{equation} = 0$
8. Sender use Bulletproofs to generate range proof $\pi_3 = PoK\{(v_3, cm^{new}, r) : cm^{new} = g^r h^{v_3} \bigwedge v_3 \in [0, 2^n]\}$
9. Sender uses Twist ElGamal to encrypt transaction value with receiver's key: $C_{value} = (pk_r^{r_2}, g^{r_2}h^{v_3})$
10. Sender uses Twist ElGamal to encrypt transaction value with audit key: $X = upk^{PRF^k(pk_r,\rho_3)}, Y = cm^{new}$ We define $C_{content} = Enc^1_{upk}(v_3) = (X, Y)$
11. Sender generates ZKP, prove that the values encrypted by audit key are correct $\pi_4 = PoK\{(r_3) : X = upk^{r_3} \bigwedge Y = g^{r_3}h^{v_3}\}$, where $r_3 = PRF^k(pk_r, \rho_3)$
    Sigma protocol of π_4(P for prover and V for verifier):
    (a) P randomly generates a,b. send $A = upk^a, B = g^a h^b$ to V
    (b) V randomly choose e, send e to P as challenge

(c) P calculate $z_1 = a + er_3, z_2 = b + ev_3$, send z_1, z_2 to V, V check the following equation:

$$upk^{z_1} = AX^e \tag{7}$$

$$g^{z_1}h^{z_2} = BY^e \tag{8}$$

We use Fiat-Shamir Transform to convert the above Sigma protocol into a NIZK. We want to point out that this proof can be easily extended to m input-n output transactions. And the proof of security properties can be seen in [6].

12. Sender uses audit key encrypt identity information: $C_{sender} = Enc^2_{upk}(pk_s)$, $C_{receiver} = Enc^2_{upk}(pk_r)$
13. Sender generates ZKP, prove that the identities of sender encrypted with audit key are correctly $\pi_5 = PoK\{(pk_s, sk_s) : C_{sender} = Enc^2_{upk}(pk_s) \bigwedge pk_s = PRF^{pk}(sk_s) \bigwedge sn_1 = PRF^{sn}(\rho_1, sk_s) \bigwedge sn_2 = PRF^{sn}(\rho_2, sk_s)\}$
Then we expand PRF and Enc function, π_5 can be written as:

$$\begin{aligned} &\pi_5 = \pi_7 \circ \pi_8, C_{sender} = (X_1, Y_1) \\ &\pi_7 = PoK\{(r_4) : X_1 = g^{r_4} \bigwedge Y_1 = upk^{r_4} g^{sk_s}\} \\ &\pi_8 = PoK\{(sk_s) : Y_1 = upk^{r_4} g^{sk_s} \bigwedge sn_1 = g^{sk_s} h^{\rho_1} \bigwedge sn_2 = g^{sk_s} h^{\rho_2}\} \end{aligned} \tag{9}$$

Where $Enc^2_{upk}(pk_s) = (X_1, Y_1)$, and r_4 is random generated. The proof of π_7 is similar with Sigma protocol of π_4, so we do not prove them in there. The proof of π_8 is below:

Sigma protocol of π_8:

(a) P random generate a, b_1, b_2, b_3. send $A = g^a, B_1 = upk^{b_1}, B_2 = h^{b_2}, B_3 = h^{b_3}$ to V

(b) V random generate e, send e to P as challenge

(c) P calculate $z = a + e \cdot sk_s, z_1 = b_1 + e \cdot r_4, z_2 = b_2 + e \cdot \rho_1, z_3 = b_3 + e \cdot \rho_2$, send z, z_1, z_2, z_3 to V, V check the following equation:

$$g^z upk^{z_1} = A(Y_1)^e B_1 \tag{10}$$

$$g^z h^{z_2} = A(sn_1)^e B_2 \tag{11}$$

$$g^z h^{z_3} = A(sn_2)^e B_3 \tag{12}$$

We use Fiat-Shamir Transform to convert the above Sigma protocol into a NIZK. We want to point out that this proof can be easily extended to m input-n output transactions. We will give the proof of security properties later in A.1.

14. Sender generates ZKP, prove that the identities of receiver encrypted with audit key are correct: $\pi_6 = PoK\{(pk_r) : C_{receiver} = Enc^2_{upk}(pk_r) \bigwedge cm^{new} = g^{PRF^k(pk_r, \rho_3)} h^{v_3}\}$.
Then we expand PRF and Enc function, π_6 can be written as:

$$\begin{aligned} &\pi_6 = \pi_9 \circ \pi_{10}, C_{receiver} = (X_2, Y_2) \\ &\pi_9 = PoK\{(r_5) : X_2 = g^{r_5} \bigwedge Y_2 = upk^{r_5} pk_r\} \\ &\pi_{10} = PoK\{(pk_r) : Y_2 = upk^{r_5} pk_r \bigwedge cm^{new} = g^{g^{pk_r} h^{\rho_3}} h^{v_3}\} \end{aligned} \tag{13}$$

Where $Enc^2_{upk}(pk_r) = (X_2, Y_2)$, and r_5 is random generated. We use zk-SNARK to prove π_9, π_{10}.

15. Sender send $\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, cm', sn_1, sn_2, r_{equation}, C_{content}, C_{sender}, C_{receiver}$ to verifier.
16. Sender send ρ_3 to receiver through a separate channel.

4.2 Transaction Verify Scheme

1. Verifier verify $\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6$.
2. Verifier verify $\pi_{11} : cm' \cdot g^{r_{equation}} \cdot cm^{-new} = 0, (cm^{new} = C_{content}[1])$.
3. Verifier verify if sn_1, sn_2 already in SNList, if not, then record nullified sn_1, sn_2 into SNList (we call it π_{12}).
4. Verifier record cm^{new} in Merkle tree node of CMList.
5. Verifier send $C_{content}, C_{receiver}, C_{sender}$ to auditor.

4.3 Transaction Receiving Scheme

1. Receiver checks blockchain to get C_{value}, then encrypts it with sk_r to get v_3.
2. Receiver get ρ_3 and checks blockchain to get cm^{new}, then use ρ_3,v_3,cm^{new} for next transaction.

4.4 Transaction Audit Scheme

1. The auditor obtains (X, Y) from $C_{content}$, calculate $v_3 = \log_h(\frac{Y}{X^{usk^{-1}}})$, get the transaction value v_3, this needs brute-forced calculation, but it is an acceptable calculate. Firstly, the transaction value is not a large number. Secondly, this computation will be carried out locally by the auditor, rather than smart contracts.
2. Auditor calculate $Dec(C_{sender}), Dec(C_{receiver})$ to get public key of sender and receiver.

To sum up, we give the overview on figure 4.

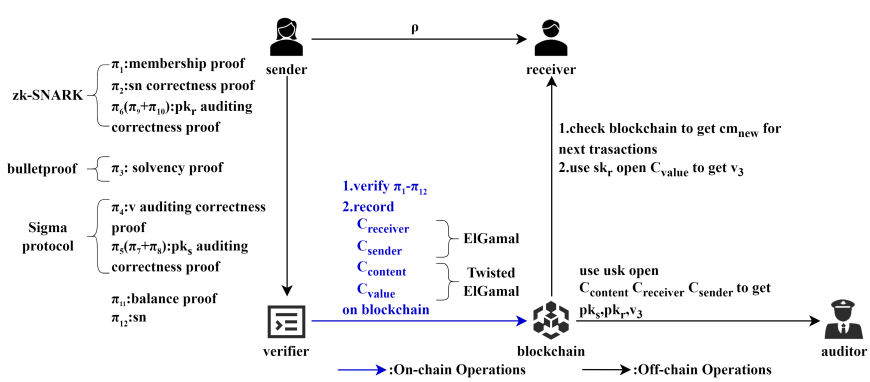


Fig. 4: Scheme design overview

5 Security

Theorem 5.1. Assuming the security of ZKP, the IND-CPA security of ElGamal, and Twisted ElGamal, the aforementioned scheme is secure.

5.1 Correctness

The correctness of the ZKP is included in the assumption. And other parts of the scheme are easy to verify by simple calculations.

5.2 Privacy

Lemma 5.2. Assume the adaptive zero-knowledge property of NIZK and the IND-CPA security of ElGamal and Twisted ElGamal, the above scheme satisfies the privacy property.

Proof. We proceed via a sequence of games. Let S_i be the probability that $\mathcal{A}$ wins in Game i.

Game 0. This game corresponds to the Game 0 in Experiment 0.

Game 1. Game 1 is the same as Game 0 except in the challenge phase, the difference is that zero-knowledge proof $\pi_1, \pi_2, \pi_3, \pi_4$ is generated by simulator simulation. If the advantage of $\mathcal{A}$ winning in Game 0 and Game 1 has a difference, then we can build a scheme to break the zero-knowledge property. So we have:

$$|\Pr[S_1] - \Pr[S_0]| \leq negl(\lambda) \tag{14}$$

Similar to Game 0, we use$\Pr[S_1^v], \Pr[S_1^s], \Pr[S_1^r]$ to describe the advantage of $\mathcal{A}$ to guess v, pk_s, pk_r. We know that $\Pr[S_1] = \max\{\Pr[S_1^v], \Pr[S_1^s], \Pr[S_1^r]\}$

Game 2. Game 2 is a part of Game 1. In Game 2, $\mathcal{CH}$ makes a random guess for the index of target pk, i.e., randomly picks an index $j \in (T - T_{corrupt})$. If $\mathcal{A}$ makes an extraction query of pk_j in the pre query stage, or picks $pk \neq pk_j$ in the query stage, then $\mathcal{CH}$ abort. Let W be the event that $\mathcal{CH}$ does not abort. We can get $\Pr[W] = 1/(T - T_{corrupt})$. $\mathcal{A}$'s view in Game 0 is identical to that in Game 1. Then we have:

$$\begin{aligned} \Pr[S_2^s] &= \Pr[S_1^s] \cdot \Pr[W] \\ \Pr[S_2^r] &= \Pr[S_1^r] \cdot \Pr[W] \end{aligned} \tag{15}$$

Game 3. Game 3 is a part of Game 1. The difference is that Game 3 random generate r_1 to replace $cm = g^{g^{pk} h^{\rho}} h^v$.

We know that the ρ are random generated in $cm = g^{g^{pk} h^{\rho}} h^v$, so cm is random distributed as r_1,so cm is indistinguishable with r_1, we have:

$$|\Pr[S_3] - \Pr[S_1^v]| \leq negl(\lambda) \tag{16}$$

Game 4. Game 4 is a part of Game 2. The difference is that Game 4 random generate r_4 to replace $cm = g^{g^{pk_r} h^{\rho}} h^v$. Similar to Game 3, r_4 is indistinguishable with cm because of the randomness of ρ. So we have:

$$|\Pr[S_4] - \Pr[S_2^r]| \leq negl(\lambda) \tag{17}$$

Game 5. Game 5 is a part of Game 2. The difference is that Game 5 random generate r_4 to replace $sn = pk_s h^\rho$. Similar to Game 3 and 4, r_4 is indistinguishable with sn because of the randomness of ρ. So we have:

$$|\Pr[S_5] - \Pr[S_2^s]| \leq negl(\lambda) \tag{18}$$

Game 6. Game 6 is the sum of Game 3,4,5. Considering the message sent to Verifier. In $SumPara = (\pi_1, \pi_2, \pi_3, \pi_4, cm', sn_1, sn_2, r_{equation}, C_{content}, C_{sender}, C_{receiver})$, we have already use simulation value to replace ZKP π, and we use random value to replace commitment and nullifier cm, sn, then $r_{equation}$ is randomly generated. Now we remain the privacy of $C_{content}, C_{sender}, C_{receiver}$ to be proven.

Lemma 5.3. If the encryption schemes ElGamal and Twisted ElGamal are IND-CPA secure, then we can get $\Pr[S_6] \leq negl(\lambda)$.

Proof. If $\mathcal{A}$ wins in Game 6 in a negligible advantage, then we can build a simulator $\mathcal{B}$ to break the IND-CPA security of ElGamal or Twisted ElGamal scheme. The simulator simulation runs Game 6 as follows.

1. generation phase: $\mathcal{B}$ run $pp \leftarrow Setup(1^\lambda)$
2. pre query phase: $\mathcal{A}$ adaptively makes queries to below oracles finitely:
   $\mathcal{O}_P$: $\mathcal{B}$ invoke $CreateAddress$ to get (sk, pk), then move sk, pk to the keys list T, then return undisclosed pk to $\mathcal{A}$.
   $\mathcal{O}_S$: $\mathcal{B}$ move sk, pk to $T_{corrupt}$ and return sk to $\mathcal{A}$.
   $\mathcal{O}_V$: $\mathcal{B}$ check if pk in $T_{corrupt}$, then invoke $CreateTran$ and return $SumPara$ to $\mathcal{A}$.
   $\mathcal{O}_{RT}$ $\mathcal{B}$ get (cm, s) or (sn, s) and return correspond (ρ, pk_r, v) or (ρ, sk_s) to $\mathcal{A}$.
3. query phase: $\mathcal{B}$ invoke $CreateAddress$ and get (pk_r, sk_s), then random generate ρ and generate v, finally invoke $CreateTran, CreateAudit$ to get $SumPara$ and send to $\mathcal{A}$.
4. guess phase: $\mathcal{A}$ output ${pk_r}', {pk_s}', v'$, $\mathcal{B}$ send pk_r', pk_s' to ElGamal encryption scheme, and send v' to Twisted ElGamal encryption scheme.

We know that $\mathcal{A}$'s observation in Game 6 is equivalence distribute with $\mathcal{B}$, so the probability of $\mathcal{A}$ succeeding in Game 6 is the same as the probability of $\mathcal{A}$ succeeding in $\mathcal{B}$. Game 6 and simulation of $\mathcal{B}$ are PPT algorithms, so the advantage of $\mathcal{A}$ win in Game 6 is the same as $\mathcal{B}$ win in ElGamal encryption scheme or Twisted ElGamal encryption scheme. This prove Lemma 5.3.

Now we can get:

$$\Pr[S_6] \leq negl(\lambda) \tag{19}$$

To sum up, we prove Lemma 5.2.

5.3 Auditability

If Sigma protocol and zk-SNARK are sound, then we can extract v, pk_r, pk_s from the proofs π_6, π_5, π_4, and the ciphertexts $C_content$, C_sender, $C_receiver$ using their respective knowledge extractors. Therefore, by decrypting with usk,

the auditor can audit the transaction. We now need to prove that the adversary cannot forge false knowledge.

Lemma 5.4. If PRF is collision-resistant, the scheme satisfies auditability.

In our scheme, if $\mathcal{A}$ do not follow the rule of $CreateTran$ want to pass $VerifyTran$, he must generate $cm^*, sn^*(cm^* \neq cm, sn^* \neq sn)$ satisfy game in 6. Consider the auditability of cm:

if the $\mathcal{A}$ can forge cm, sn, then we can build a simulator $\mathcal{B}$ to break the collision-resistant of PRF:

1. generation phase: $\mathcal{B}$ run $pp \leftarrow Setup(1^\lambda)$
2. query phase: $\mathcal{B}$ invoke $CreateTran$, calculate $cm = g^{PRF^k(pk,\rho)}h^v$ and send cm to A
3. guess phase: $\mathcal{A}$ send cm^* to $\mathcal{B}$.

In the security proof in privacy, we prove that $\mathcal{A}$ can not forge $cm^* = cm$, so $cm^* \neq cm$, we know that the Pedersen Hash function is collision-resistant for fixed-length input, so we get $PRF^k(pk^*, \rho^*) = PRF^k(pk, \rho)$. So if $\mathcal{A}$ can win the game in 6 in a non-negligible probability, $\mathcal{B}$ can break the collision-resistant of PRF.

The auditability proof of sn is similar to the cm above. Then we prove Lemma 5.4.

To sum up, we prove Theorem 5.1.

6 Performance

In table 2, we give the comparison of security attributes for different schemes:

Table 2: Comparison of Security Attributes for Different Schemes

scheme	**Identity Auditing**	**Value Auditing**	**Independent Auditing**	**Global Mixing**	**Full Transaction Auditing**
PGC[6]	Yes	Yes	Yes	No	Yes
Traceable Monero[20]	Yes	No	Yes	No	Yes
ASOZ	Yes	Yes	Yes	Yes	Yes

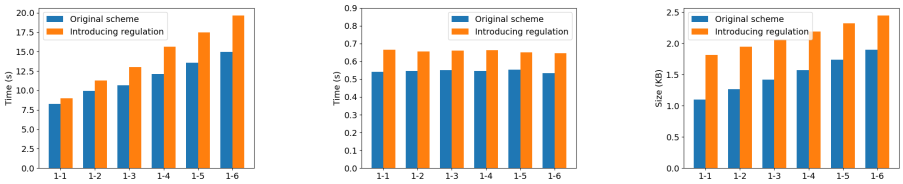
We now give a prototype implementation of ASOZ in JavaScript and circom mainly based on circomlibjs [15], and collect the benchmarks on Intel i7-12700H CPU (2.30GHz) and 16GB of RAM. The source code of ASOZ is publicly available at GitHub [1].

Our scheme is implemented on Jubjub curve [7], and the length of sk are 253 bits. We choose Poseidon [12] as the hash function for the Merkle tree, Groth16 [13] as our zk-SNARK's proving system. Additionally, we use linearly aggregated Sigma protocol to reduce overhead.

We show the zk-SNARK cost of our scheme in Table 3. The proof generation time, proof verification time, and transmission cost of our scheme are illustrated

in Figures 5. After surveying the ZCash explorer [3], we noted that cryptocurrency transactions commonly manifest as either 1 input to n outputs or n inputs to 1 output, with the former being more prevalent. The most common scenario is the 1 input-1 output transaction. According to these observations. We test the performance of 1-1 to 1-6 transactions. In our experiments, In 1-1 transactions, the auditing functionality introduces 8% increment in proof generation time, 23% rise in verification time compared to the original scheme, and 65% rise in transmission cost. The results indicate that the auditing cost introduced by our scheme falls within an acceptable range, which follows the Out-of-band Cost principle.

Because our implementation language is JavaScript, which is relatively inefficient, there is considerable room for improvement in the computational speed of the Sigma protocol.



(a) Comparison of proof generating time (b) Comparison of verify time (c) Comparison of transmission cost

Fig. 5: Comparison of execution cost before and after audit introduction

Table 3: zk-SNARK Cost

input-output	**Constraints**	**Wires**	**Labels**
1-1	17413	17442	148447
1-2	22141	22169	209901
1-3	26869	26896	271355
1-4	31597	31623	332809
1-5	36325	36350	394263
1-6	41053	41077	455717

7 Further Discussion

How to balance privacy and auditability remains a big challenge for decentralized payment systems. In this paper, we propose our solution through involves the use of an auditor. The auditor can open the transaction and trace illegal transactions.

Furthermore, we propose security properties and design principles to refine our solution. We leave the following problems as future works.

Is there a better way to construct commitment and nullifier? In our scheme, we design a sigma-protocol-friendly construction to reduce out-of-band cost, but we still use zk-SNARK in some parts, leading to some necessary computational costs. We believe that, except for the membership proof in the Merkle tree, employing the relatively inefficient zk-SNARK is unnecessary in the remaining portions.

Using key agreement and key derivation. In our scheme, user and auditor reuse the public keys, which may lead to a decrease in security strength. Drawing upon the Sapling and Orchard versions of ZCash, a solution is incorporating key agreement and key derivation mechanisms to avoid the reuse of keys.

Rethinking audit capabilities. In our scheme, we define a trusted auditor, which follows the Full transaction audit principle. This design, however, undermines the decentralized idea to some extent. One reasonable enhancement is to decentralize auditor authority through the use of secret sharing mechanisms.

References

1. https://github.com/AwakeLithiumFlower/ASOZ
2. Coinjoins - learn about bitcoin collaborative transactions, https://www.coinjoins.org/
3. nighthawk apps: Zcash explorer - search the zcash blockchain, https://zcashblockexplorer.com/
4. Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474 (2014). https://doi.org/10.1109/SP.2014.36
5. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 315–334 (2018). https://doi.org/10.1109/SP.2018.00020
6. Chen, Y., Ma, X., Tang, C., Au, M.H.: Pgc: Decentralized confidential payment system with auditability. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) Computer Security – ESORICS 2020. pp. 591–610. Springer International Publishing, Cham (2020)
7. Daira Hopwood, Sean Bowe, T.H.N.W.: Zcash protocol specification, version 2022.3.8 [nu5] (2022), https://zips.z.cash/protocol/canopy.pdf
8. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (1976). https://doi.org/10.1109/TIT.1976.1055638
9. Elgamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**(4), 469–472 (1985). https://doi.org/10.1109/TIT.1985.1057074
10. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology — CRYPTO' 86. pp. 186–194. Springer Berlin Heidelberg, Berlin, Heidelberg (1987)

11. Garman, C., Green, M., Miers, I.: Accountable privacy for decentralized anonymous payments. In: Grossklags, J., Preneel, B. (eds.) Financial Cryptography and Data Security. pp. 81–98. Springer Berlin Heidelberg, Berlin, Heidelberg (2017)
12. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for Zero-Knowledge proof systems. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 519–535. USENIX Association (Aug 2021), https://www.usenix.org/conference/usenixsecurity21/presentation/grassi
13. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016. pp. 305–326. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
14. Headquarters, C.: Large trader reporting program — cftc, https://www.cftc.gov/IndustryOversight/MarketSurveillance/LargeTraderReportingProgram/index.htm
15. iden3: iden3, https://github.com/iden3
16. Inc, C.: The chainalysis 2023 crypto crime report, https://go.chainalysis.com/2023-crypto-crime-report.html
17. Jeong, G., Lee, N., Kim, J., Oh, H.: Azeroth: Auditable zero-knowledge transactions in smart contracts. IEEE Access **11**, 56463–56480 (2023). https://doi.org/10.1109/ACCESS.2023.3279408
18. Kang, H., Dai, T., Jean-Louis, N., Tao, S., Gu, X.: Fabzk: Supporting privacy-preserving, auditable smart contracts in hyperledger fabric. In: 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 543–555 (2019). https://doi.org/10.1109/DSN.2019.00061
19. Lab, M.R.: Monero research lab (mrl) — monero - secure, private, untraceable, https://www.getmonero.org/resources/research-lab/
20. Li, Y., Yang, G., Susilo, W., Yu, Y., Au, M.H., Liu, D.: Traceable monero: Anonymous cryptocurrency with enhanced accountability. IEEE Transactions on Dependable and Secure Computing **18**(2), 679–691 (2021). https://doi.org/10.1109/TDSC.2019.2910058
21. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized business review (2008)
22. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) Advances in Cryptology — CRYPTO '91. pp. 129–140. Springer Berlin Heidelberg, Berlin, Heidelberg (1992)
23. Sander, T., Ta-Shma, A.: Auditable, anonymous electronic cash. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. pp. 555–572. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
24. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (jan 1991). https://doi.org/10.1007/BF00196725, https://doi.org/10.1007/BF00196725
25. Wüst, K., Kostiainen, K., Capkun, V., Capkun, S.: Prcash: Fast, private and regulated transactions for digital currencies. Cryptology ePrint Archive, Paper 2018/412 (2018), https://eprint.iacr.org/2018/412

A Missing Proofs and Protocols

A.1 The Proof of Sigma Protocol

Sigma protocol of π_8 is an example of 2 input-1 output transaction. In order to give a general proof, we define the following problem in n input-m output transaction:

$$\pi_{11} = PoK\{(sk_s) : Y_1 = upk^{r_4} g^{sk_s} \bigwedge sn_1 = g^{sk_s} h^{\rho_1} \bigwedge \ldots \bigwedge sn_n = g^{sk_s} h^{\rho_n}\} \tag{20}$$

The Sigma protocol of π_{11} is similar to π_8.

Sigma protocol of π_{11}:

1. P random generate $a, a_2, b_1, \ldots, b_n$. send $A = g^a, A_2 = upk^{a_2}, B_i = h^{b_i}$ to V
2. V random generate e, send e to P as challenge
3. P calculate $y = a + e \cdot sk_s, y_2 = a_2 + e \cdot r_4, z_i = b_i + e \cdot \rho_i (i = 1, \ldots, n)$, send $y, y_2, z_1, \ldots, z_n$ to V, V check the following equations:

$$\begin{aligned} g^y upk^{y_2} &= A(Y_1)^e A_2 \\ g^y h^{z_i} &= A(sn_i)^e B_i \end{aligned} \tag{21}$$

Then we give the security proof of the protocol above:

Prefect Completeness This is obvious from simple calculation.

Special Soundness Fix the initial message $(A, A', B_1, \ldots, B_n)$, suppose there are two accepting transcripts (e, y, y_2, z_i) and (e', y', y_2', z_i'). We have $sk_s = (y - y')/(e - e'), r_4 = (y_2 - y_2')/(e - e'), \rho_i = (z_i - z_i')/(e - e')$. So if P can answer with probability greater than $1/(2^t)$ then P is consider to know sk_s, r_4, ρ_i

Special HVZK For a fixed challenge e, the simulator $\mathcal{S}$ works as below: picks y, y_2, z_i randomly, we can calculate equation 21, let A to be equal to k ,then $A_2 = g^y upk^{y_2}/(k \cdot Y_1^e), B_i = g^y h^{z_i}/(k \cdot sn_i^e)$. Which means that an accepting transcript $(A, A_2, B_i, e, y, y_2, zi)$ is distributed exactly like a real execution where V sends e.