

PACChain: Private, Authenticated & Auditable Consortium Blockchain and its Implementation

Tsz Hon Yuen

The University of Hong Kong, Hong Kong

Abstract

Blockchain provides a distributed ledger recording a globally agreed, immutable transaction history, which may not be suitable for Fintech applications that process sensitive information. This paper aims to solve three important problems for practical blockchain applications: privacy, authentication and auditability.

Private transaction means that the transaction can be validated without revealing the transaction details, such as the identity of the transacting parties and the transaction amount. Auditable transaction means that the complete transaction details can be revealed by auditors or law enforcement agencies. Authenticated transaction means that only authorized parties can be involved in the transaction. Auditability prevents a malicious user from abusing the privacy property. Authenticity helps the auditor to trace the identity of the malicious user in the system. In this paper, we present a private, authenticated and auditable consortium blockchain, using a number of cryptographic building blocks. Our scheme can be integrated with the architecture of consortium blockchain.¹

Keywords: Blockchain, privacy.

1. Introduction

Blockchain is a fast-growing field of technology since it is recognized as the core component of Bitcoin [2]. Blockchain can serve as a distributed ledger in a peer-to-peer network to provide publicly verifiable data. The

Email address: `thyuen@cs.hku.hk` (Tsz Hon Yuen)

¹This is the full version of the paper in CANS 2019 [1].

public blockchain is an open, permissionless system such that every one is allowed to join as a user or miner freely. Cryptocurrencies like Bitcoin is an example of public blockchain. They usually have limited throughput. The *consortium blockchain* is a partly private blockchain. The submission of transactions can be performed by many (authorized) users, but the verification of transactions is only permitted by a few predetermined parties. Consortium blockchain provides a higher efficiency (more than 10K transactions per second (tps)) than the public blockchain. Consortium blockchain is suitable for organizational collaboration. Hyperledger, an umbrella project of open source consortium blockchain, has 130 members including members from the IT industry (e.g., IBM, Intel) and the financial industry (e.g., JP Morgan, American Express).

Privacy in Blockchain. Privacy is important for commercial system, especially in the financial sector where money is transferred from one party to another. No one would like to have his bank account transaction history posted on a public blockchain. We define three key privacy properties that we want to achieve in this paper:

1. **Sender Privacy:** the sender's identity is not known by any third party and two valid transactions of the same sender should not be linked.
2. **Recipient Privacy:** the recipient's identity is not known by any third party and two valid transactions of the same recipient should not be linked.
3. **Transaction Privacy:** the content of the transaction is not known by any third party. General transaction privacy for smart contract is difficult to achieve without using general zero-knowledge proof of circuit or fully homomorphic encryption, which are both not quite practical. In this paper, we only consider the privacy for *transaction amount*.

The above conditions should hold for any third party (including the parties running the consensus algorithm). There are also a number of academic and industrial solutions for privacy-preserving blockchain. Details will be discussed in §3.1.

Our Contributions: Private, Authenticated & Auditable in Consortium Blockchain. Auditability is essential for financial blockchain applications and unconditional anonymity may not be desirable. Financial institution has to check if there is any money laundering or terrorist-related activities. In case of court order, the institute has to provide the complete

information of a particular transaction to the court. For the above situations, the privacy of certain transaction has to be revoked if necessary. For simplicity, we denote the party to legally revoke privacy as the *auditor*.

Authentication is important for consortium blockchain in two aspects. First, the consortium companies need to ensure that the user is authenticated to use the system (e.g., he has paid/subscribed for the blockchain service). The consortium companies do not earn from the “mining” process and no new coin is generated from consortium blockchain. Second, authentication is useful for tracing real user identity during the auditing process. If users can transact in the consortium blockchain without registration, then the auditor can only discover the self-generated public key after opening the transaction. The real world identity can only be recovered if the user was registered before and is authenticated during the transaction.

In this paper, we show how to construct a private, authenticated and auditable consortium blockchain: PACHain. We give the sender privacy, recipient privacy and transaction privacy by three separate modules. Auditability is provided for all three modules. Authentication is analyzed for the sender privacy and recipient privacy modules. It allows us to analysis the security of each module clearly. It gives the flexibility for system architects to choose the properties according to the business requirements. In our construction, we use a number of cryptographic techniques (e.g., anonymous credential, zero-knowledge range proof, additive homomorphic encryption) and modify them for higher efficiency in consortium blockchain. Table 1 gives the comparison of our paper and related works described in §3.1.

Finally, we give the system architecture for integrating our PACHain with Hyperledger Fabric, the open source consortium blockchain platform. We demonstrate how to incorporate various parties in PACHain (e.g., peers, auditors, certificate authorities) into the concrete framework of Hyperledger Fabric. We highlight the changes in each individual steps, such as user registration, transaction proposal and transaction endorsement.

2. PACHain Overview

In this paper, we show the high level overview of how to achieve privacy and auditability in consortium blockchain.

		SP	RP	TP	Aud	Auth	Eff
Public blockchain	Monero	◐	●	●			◐
	Zcash	●	●	●			◐
	DAP [3]	●	●	●	●		◐
Consortium/ Private blockchain	R3 Corda, [4], Hyperledger Fabric	◐	◐	◐		●	●
	Fabric experiment	◐	◐	◐	◐	◐	◐
	PRCash [5]	◐	◐	●	◐		◐
	This paper	◐	●	●	●	●	◐

Table 1: Comparison with existing privacy-preserving blockchain schemes. SP, TP, TP, Aud, Auth and Eff stands for sender privacy, recipient privacy, transaction privacy, auditability, authentication and efficiency respectively.

2.1. System Model

In blockchain system, clients can submit transactions (Tx) to nodes running the consensus algorithms. The nodes validate the Tx and add it to a block. In this paper, we extend the system model of Hyperledger Fabric 1.0², which is designed for consortium blockchain. The system hosts smart contracts (called *chaincode* in Hyperledger Fabric) that comprise the application logic of the system. Chaincode holds state and ledger data, and transactions are operations invoked on the chaincode. Transactions have to be *endorsed* by endorsers and only endorsed Tx's can be committed.

There are five types of nodes in PACHain:

1. **Client:** A client invokes a Tx to the endorsers, and also broadcasts the transaction-proposals to the orderer.
2. **Peer:** There are two types of peers. *Endorser* checks the validity of the Tx submitted by the client. *Committing peers* commits Tx's and maintains the ledger. Note that a peer can play the role of endorser and committing peers at the same time.
3. **Orderer:** A node running the service of ordering Tx's. Consensus algorithm is run between many orderers.
4. **Auditor:** The auditor can recover the sender identity, recipient identity and/or the Tx amount of any transaction.

²Hyperledger Fabric Architecture Explained. <http://hyperledger-fabric.readthedocs.io/en/latest/arch-deep-dive.html>

<http://hyperledger-fabric.readthedocs.io/en/latest/arch-deep-dive.html>

5. **Certificate Authority(CA):** CA issues certificates for the public keys of clients. Only authorized parties can be involved in a Tx.

Transaction Workflow. Assume that the client obtains a certificate from the CA. The basic workflow of a transaction (Tx) is as follows:

1. The client creates a signed Tx and sends it to endorser(s) of its choice.
2. Each endorser validates a Tx and produces an endorsement signature.
3. The submitting client collects endorsement(s) for a Tx and broadcasts it to the orderers.
4. The orderers deliver the block of ordered Txs to all peers.
5. The committing peer checks if every ordered Tx is endorsed correctly according to some policy. It also removes double-spending Tx endorsed by different endorsers concurrently. If the checking is correct, it commits Txs and maintains the state and a copy of the ledger.

The auditor can recover the sender identity, recipient identity and the transaction amount of the Tx if necessary.

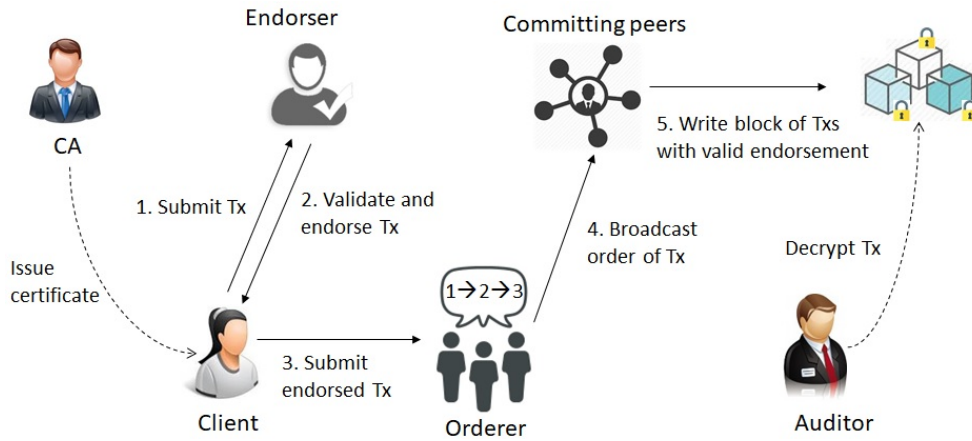


Figure 1: Nodes and Workflow of PACChain

UTXO Model for Digital Assets. The model of Unspent Transaction Outputs (UTXO) is used in many blockchain systems, such as Bitcoin. If a user wants to spend his digital assets in a Tx, he has to refer to the specific assets that he wants to spend. If he spends the same asset twice, the verifier will notice it and will reject the Tx. In this paper, we consider

the general UTXO model for digital assets and provides anonymity for their transactions.³

2.2. High-Level Description of PACHain

We provide three privacy properties in three different modules. It accommodates the plug-and-play design principle of Hyperledger Fabric, where system architects can choose the modules according to their concrete requirement. Generally speaking, we use the semi-trusted setting of consortium blockchain to set up system parameters and user credentials. Then, we can achieve a more efficient solutions for privacy and auditability, as compared to public blockchain (where all nodes may have Byzantine faults).

Sender Privacy: The sender Alice uses anonymous credential in Step 1 of the workflow. The credential is issued by the endorser of the last corresponding Tx received by Alice (Step 2), to ensure that she is authenticated. We have to add a tag (which is a deterministic function of the user secret key) to the signature in Step 1 in order to detect double spending. We take advantage of having some semi-trusted endorsees in our framework, which can act as the group manager of anonymous credential. It provides sender anonymity for all users using the same endorser (see §6.2). This anonymous credential approach is more efficient the ring signature-based approach in Monero and zk-SNARK-based approach in Zcash.

Recipient Privacy: The sender generates a one-time ephemeral key for the recipient in Step 1 of the workflow. However, we face the challenge that only authorized recipients are allowed in consortium blockchain. Therefore, we have to embed a zero-knowledge proof that the recipients are authorized into the generation of one-time ephemeral key (see §5.2). Recipient authorization is a new security requirement which does not exist in the public blockchain. This requirement is a major difference between PACHain and [3].

Auditability fo Sender and Recipient: Auditability for sender and recipient identity can be achieved by encrypting their public keys to the auditor, followed by a zero-knowledge proof of the correctness of such encryption.

Transaction Privacy with Auditability: The sender uses additive homomorphic encryption and zero-knowledge proof in Step 1 of the workflow. The proof shows that the Tx output amount is encrypted correctly, falls within a valid

³Hyperledger Fabric 1.0 currently uses the account balance model by default, but it also supports the UTXO model.

range, and the total Tx input and output amount are balanced. The major difficulty is to allow an auditor to decrypt the transaction amount. Monero used the additive homomorphic Pedersen commitment which does not allow decryption. Simply using Paillier encryption with a zero-knowledge range proof is not efficient: encrypting 2048-bit plaintext and performing the binary-decomposition range proof is an overkill for 64-bit of transaction amount (see §4). By taking advantage of the consortium blockchain framework, we propose the use of modified ElGamal encryption with a signature-based range proof (see §4.2). The size of the zero knowledge proof is independent to the size of the range.

2.3. Threat Model

We assume that the system parameters are honestly generated. We consider the following attacker model for privacy:

- The attacker can create malicious client or corrupt any client.
- The peer and CA are assumed to be honest-but-curious: it tries to break privacy passively by recording all inputs, outputs and randomness used, but it still follows the protocols.
- All keys and data used by the orderers are known to the attacker.
- The auditor is assumed to be honest for privacy. Compromising the auditor trivially breaks all privacy.⁴

We do not consider network-level privacy issues, such as tracing the sender's IP address or analyzing meta-data in network packets.

Consider the example of a consortium of banks. It is reasonable to assume that the banks jointly generate system parameters (e.g., by multi-party computation). Each bank acts as a peer and follows the protocol (if it ignores Tx or endorses invalid Tx, it will be discovered by other banks and will be handled by other means outside the blockchain system). However, the bank may be curious to view the Tx details of other banks. Our privacy model

⁴One may argue that it gives too much power for auditor. However in most companies, internal auditor should always be able to control and governance business operations. In some industries, laws require that information must be provided to the court when requested (e.g., anti-money laundering in banks and lawful interception in telecommunication industry).

captures this scenario. As a result, we allow a larger degree of decentralization by allowing multiple endorsers to validate a Tx, without causing extra privacy leakage.

3. Backgrounds

3.1. Related Works

Public Blockchain. Monero is a cryptocurrency providing privacy by linkable ring signature, stealth address and ring confidential transactions [6]. The major disadvantage of Monero is the size of the linkable ring signature, which is proportional to the size of the ring (related to the level of sender anonymity).

Zcash offers privacy and selective transparency of TxS by using zero-knowledge proofs (zk-SNARK) on special *shielded* transactions [7]. The major disadvantage of Zcash is the large signing key size of 896MB, long key generation time of 8 minutes and long signing time of 3 minutes. As a result, only around 7% of TxS are shielded in Zcash as of March 2017. Recently, Zcash proposed a new Sapling update which reduced the proving time to a few seconds. However, it is still far less efficient than the Monero’s approach (and also this paper’s approach) which is about 100ms.

A decentralized anonymous payment (DAP) with the support of accountability is proposed in [3]. They tackle the accountability problem in the public blockchain by using the zk-SNARK approach. Hence, it is also not efficient.

Recently, some privacy preserving public blockchain specialized in different area are proposed, such as IoT [8], health care [9] and thin client authentication [10]. No authentication is provided for all solutions in the public blockchain.

Consortium Blockchain. For consortium blockchain, the major platforms provide limited support for privacy. In R3’s Corda and Hyperledger Fabric, privacy is maintained by the system’s access control policy. The level of privacy is lower since system administrators are trusted. In addition, Hyperledger Fabric uses *VKey* to provide transaction privacy by symmetric encryption. The problem of key distribution between all parties is a severe challenge for a global deployment of such system. There is an experiment to integrate identity mixer [11] with Fabric⁵. The identity mixer provides a

⁵<https://jira.hyperledger.org/browse/FAB-2005>

better sender anonymity (by preventing the system administrator to link all transactions from the same user), and provides auditability for the sender identity.

Private Blockchain. A private industrial blockchain is proposed in [4], where multiple distributed private ledgers are maintained by a subset of stakeholders in the network. Each private ledger is maintained by its stakeholder only. This approach only provides privacy for users outside the private ledger. To provide higher level of privacy, the system must be divided into smaller private ledgers. However, more expensive cross ledger asset transfer is needed if there are some Txs between private ledgers.

PRCash[5] is a centrally-issued cryptocurrency with privacy and auditability. They provide anonymity of the sender and recipient identity by using mixing. However, there are still certain linkability between Txs. For auditability, PRCash provides perfect transaction amount privacy and no auditability for small amount Tx. For Txs with large amount, it cannot provide privacy for these Txs.

3.2. Mathematical Backgrounds

Bilinear Groups. \mathcal{G} is an algorithm, which takes as input a security parameter λ and outputs a tuple $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e})$, where \mathbb{G}_1 , \mathbb{G}_2 and \mathbb{G}_T are multiplicative cyclic groups with prime order p , and $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a map, which has the following properties: (1) **Bilinearity:** $\hat{e}(g^a, \hat{g}^b) = \hat{e}(g, \hat{g})^{ab}$ for $\forall g \in \mathbb{G}_1, \hat{g} \in \mathbb{G}_2$ and $\forall a, b \in \mathbb{Z}_p$. (2) **Non-degeneracy:** There exists $g \in \mathbb{G}_1, \hat{g} \in \mathbb{G}_2$ such that $\hat{e}(g, \hat{g}) \neq 1_{\mathbb{G}}$. (3) **Computability:** There exists an efficient algorithm to compute $\hat{e}(g, \hat{g})$ for $\forall g \in \mathbb{G}_1, \hat{g} \in \mathbb{G}_2$.

4. Transaction Privacy

One of the challenging part for privacy in blockchain is the confidentiality of the transaction amount. The major difficulty is how to verify the Tx that (1) the total committed input amount is equal to the total committed output amount; (2) all committed amounts fall within a valid range, e.g., from 0 to 2^{64} . This requirement is commonly known as the *confidential transaction*. Theoretically, it can be achieved by combining additive homomorphic commitment with zero-knowledge range proof. One example is Monero [6], which uses Pedersen commitment with 1-out-of-2 ring signature-based binary decomposition range proof.

In PACHain, we require auditability of transaction amount. It is common in business use cases that all Txns can be audited by internal auditors or some external regulators. Therefore, it is necessary to replace additive homomorphic commitment with additive homomorphic encryption in confidential transaction, such that the decryption key is held by the auditor. However, technical difficulties arise when combining the additive homomorphic Paillier encryption [12] with existing zero-knowledge range proof.

Range Proof with Encryption. Range proof is one essential element in transaction privacy. Without a range proof, the attacker can create a transaction with one input \$0 and two outputs of \$100 and -\$100. The sum of input and output amount is still balanced. If the output amount is encrypted, the attacker can create \$100 out of an input \$0. Therefore range proof is needed to prevent any negative amount or overflow amount.

In order to prove a secret value x lies between $[0, R]$, there are three families of range proof in the literature.

- **Square Decomposition.** Boudot [13] used a mathematical theorem that any positive integer can be written as a sum of four squares. However, it can only be used in a group of unknown order (since for group with known order p , a negative number $-n$ is equivalent to $-n + p$). If this method is used in blockchain, it would result in a much larger signature size due to the use of composite order group.
- **Multi-Base Decomposition.** Bellare and Goldwasser [14] used the binary decomposition of x to perform a bit-by-bit range proof in $[0, R]$. Later, it is further generalized to a multi-base decomposition. However, the space and time complexities depend on the size of the range. Recently, Bulletproof [16] is proposed to reduce the proof size to $O(\log R)$.
- **Signature-based.** Camenisch *et al.* [15] proposed a range proof based on a set Φ of public signatures on every element in $[0, R]$ by a designated authority. The range proof consists of proving the knowledge of a signature in Φ on the secret x without revealing x nor the signature.

Problems of Using Paillier Encryption with Range Proof. Paillier encryption [12] is the most common additive homomorphic encryption to date. However, combining Paillier encryption with the existing range proof is not trivial in blockchain applications. There is no simple and efficient solution for using Paillier encryption with range proof in blockchain.

4.1. Security Model of Transaction Privacy Protocol

We give the notion and security model for transaction privacy.

Transaction Privacy Protocol. A transaction privacy protocol consists of a tuple of poly-time algorithms as described follows:

1. $(\mathbf{param}, \mathbf{ask}_{\text{tp}}) \leftarrow \text{Setup}(1^\lambda, R)$. On input a security parameter 1^λ and the range parameter R , it outputs the auditor secret key \mathbf{ask}_{tp} and the system parameter \mathbf{param} (which includes the auditor public key). Suppose \mathbf{param} is the input of all other algorithms, and hence is omitted for simplicity.
2. $(\{C_{\text{out},j}, r_{\text{out},i}\}_{j \in [1,n']}, \pi_{\text{tp}}) / \perp \leftarrow \text{TxPrivacySpend}(\{M_{\text{out},j}\}_{j \in [1,n']}, \{C_{\text{in},i}, M_{\text{in},i}, r_{\text{in},i}\}_{i \in [1,n]})$. On input n' output transaction amount $M_{\text{out},j}$, n transaction input ciphertext $C_{\text{in},i}$, amount $M_{\text{in},i}$ and randomness $r_{\text{in},i}$, it outputs \perp if $C_{\text{in},i}$ is not a valid ciphertext for $(M_{\text{in},i}, r_{\text{in},i})$ for some $i \in [1, n]$, or $\sum_{i=1}^n M_{\text{in},i} \neq \sum_{j=1}^{n'} M_{\text{out},j}$ ⁶. For each $M_{\text{out},j}$, it runs the following sub-protocol:
 - $(C_{\text{enc}}, r_{\text{enc}}, \pi_{\text{enc}}) \leftarrow \text{EncProof}(M)$. On input a message $M \in [0, R]$, it outputs a ciphertext C_{enc} encrypted to the auditor, the encryption randomness r_{enc} and a zero-knowledge proof π_{enc} that C_{enc} encrypts M and $M \in [0, R]$.
 It obtains $(C_{\text{out},j}, r_{\text{out},j}, \pi_{\text{out},j}) \leftarrow \text{EncProof}(M_{\text{out},j})$. It computes π_{tp} which includes all $\pi_{\text{out},j}$ and the zero-knowledge proof of $\sum_{i=1}^n M_{\text{in},i} = \sum_{j=1}^{n'} M_{\text{out},j}$. It outputs $(\{C_{\text{out},j}, r_{\text{out},i}\}_{j \in [1,n']}, \pi_{\text{tp}})$.
3. $1/0 \leftarrow \text{TxPrivacyVerify}(\{C_{\text{in},i}\}_{i \in [1,n]}, \{C_{\text{out},j}\}_{j \in [1,n']}, \pi_{\text{tp}})$. On input n input ciphertext $C_{\text{in},i}$, n' output ciphertext $C_{\text{out},j}$ and a proof π_{tp} , it outputs 1 if π_{tp} is valid, and outputs 0 otherwise. It includes running the following sub-protocol each $C_{\text{out},j}$ and $\pi_{\text{out},j} \in \pi_{\text{tp}}$:
 - $1/0 \leftarrow \text{EncVerify}(C_{\text{out}}, \pi_{\text{out}})$. On input a ciphertext C_{out} and a proof π_{out} , it outputs 1 if π_{out} is valid, and outputs 0 otherwise.
4. $M \leftarrow \text{Decrypt}(\mathbf{ask}_{\text{tp}}, C)$. On input the auditor secret key \mathbf{ask}_{tp} and a ciphertext C , it outputs the decrypted transaction amount M .

Security Model. We define the security requirements for transaction privacy:

⁶For coinbase transaction where money is created, input checking is not needed. The verification of coinbase transaction does not need to check for input ciphertext. We omit the description for coinbase transaction for simplicity.

1. No output transaction amount is outside the range.
2. The total input transaction amount is equal to the total output transaction amount.
3. No one can learn the output transaction amount, except the auditor.

The security of transaction privacy is formalized as follows.

Definition 1 (Soundness). *A transaction privacy protocol is sound if for all PPT adversary \mathcal{A} , it holds that given $(\text{param}, \text{ask}_{\text{sp}}) \leftarrow \text{Setup}(1^\lambda, R)$; \mathcal{A} outputs $(\{C_{\text{in},i}\}_{i \in [1,n]}, \{C_{\text{out},j}\}_{j \in [1,n']}, \pi_{\text{tp}})$, then*

$$\Pr \left[\begin{array}{l} \text{TxPrivacyVerify}(\{C_{\text{in},i}\}_{i \in [1,n]}, \{C_{\text{out},j}\}_{j \in [1,n']}, \pi_{\text{tp}}) = 1, \\ \exists j \in [1, n'] : M_{\text{out},j} \leftarrow \text{Decrypt}(\text{ask}_{\text{tp}}, C_{\text{out},j}), M_{\text{out},j} \notin [0, R] \end{array} \right] \leq \text{negl}(\lambda).$$

Definition 2 (Balance). *A transaction privacy protocol is balance if for all PPT adversary \mathcal{A} , it holds that given $(\text{param}, \text{ask}_{\text{sp}}) \leftarrow \text{Setup}(1^\lambda, R)$, \mathcal{A} outputs $(\{C_{\text{in},i}\}_{i \in [1,n]}, \{C_{\text{out},j}\}_{j \in [1,n']}, \pi_{\text{tp}})$, then:*

$$\Pr \left[\begin{array}{l} \text{TxPrivacyVerify}(\{C_{\text{in},i}\}_{i \in [1,n]}, \{C_{\text{out},j}\}_{j \in [1,n']}, \pi_{\text{tp}}) = 1, \\ \forall j \in [1, n'], M_{\text{out},j} \leftarrow \text{Decrypt}(\text{ask}_{\text{tp}}, C_{\text{out},j}), \\ \forall i \in [1, n], M_{\text{in},i} \leftarrow \text{Decrypt}(\text{ask}_{\text{tp}}, C_{\text{in},i}), \\ \sum_{i=1}^n M_{\text{in},i} \neq \sum_{j=1}^{n'} M_{\text{out},j} \end{array} \right] \leq \text{negl}(\lambda),$$

where \mathcal{A} can query the Spend oracle to request generating an honest run of TxPrivacySpend. We additionally require that $\{C_{\text{in},i}\}$ are from the output of the Spend oracle.

Definition 3 (Privacy). *A transaction privacy protocol is private if for all PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$, it holds that given when \mathcal{A}_1 is given param generated from $\text{Setup}(1^\lambda, R)$, \mathcal{A}_1 outputs $(\{M_{\text{out},j}^{(0)}, M_{\text{out},j}^{(1)}\}_{j \in [1,n']}, \{C_{\text{in},i}^*, M_{\text{in},i}^*, r_{\text{in},i}^*\}_{i \in [1,n]})$, and when \mathcal{A}_2 is given $(\{C_{\text{out},j}^*\}_{j \in [1,n']}, \pi_{\text{tp}}^*)$, where $(\{C_{\text{out},j}^*, r_{\text{out},j}^*\}_{j \in [1,n']}, \pi_{\text{tp}}^*) \leftarrow \text{TxPrivacySpend}(\{M_{\text{out},j}^{(b)}\}_{j \in [1,n']}, \{C_{\text{in},i}^*, M_{\text{in},i}^*, r_{\text{in},i}^*\}_{i \in [1,n]})$ for some random bit b , \mathcal{A}_1 outputs its guess b' , then:*

$$\Pr [b = b'] \leq \text{negl}(\lambda).$$

We additionally require that $\sum_{i=1}^n M_{\text{in},i} = \sum_{j=1}^{n'} M_{\text{out},j}^{(0)} = \sum_{j=1}^{n'} M_{\text{out},j}^{(1)}$.

4.2. Transaction Privacy for PAChain

We give our efficient transaction privacy solution for consortium blockchain. We overcome the problem for effectively combining additive homomorphic encryption and range proof due to two properties: (1) consortium blockchain allows trusted setup; (2) the transaction amount is short.

The first observation is that encrypting a “short” transaction amount (e.g., 51-bit can represent all 21M Bitcoins in terms of satoshi, or 64-bit can represent trillions of dollars with sixth decimals) with Paillier encryption of message space of 2048-bit is superfluous. Therefore, we propose to use the additive homomorphic ElGamal encryption instead. However, decrypting such ciphertext requires the computation of discrete logarithm, which is not feasible for large message space. Hence, we decompose the K -bit transaction amount M into ℓ segment $\mu_0, \dots, \mu_{\ell-1}$ which are smaller than the message space \mathbf{u} by $M = \sum_{j=0}^{\ell-1} \mu_j \mathbf{u}^j$. As a result, we encrypt each μ_j by additive homomorphic ElGamal encryption. The small message space of \mathbf{u} guarantees efficient decryption. In our implementation, we consider 64-bit transaction amount and the auditor uses a pre-computation table of $(g, g^2, \dots, g^{\mathbf{u}-1})$ for efficient decryption. We can choose $\ell = 4$, $\mathbf{u} = 65536$ and the pre-computation table is about 2MB. The ciphertext size is 2048-bit, which is still less than the 4096-bit ciphertext size of Paillier encryption.

Another advantage of using ElGamal encryption is the ease to combine with range proof. By using ECC ElGamal encryption, it can be combined with the Boneh-Boyen signature-based range proof [15]. Note that this solution is only suitable for consortium blockchain since the non-interactive version of such range proof requires a trusted setup.

Our Construction. Our transaction privacy (TP) protocol is described below.

- **Setup.** On input a security parameter 1^λ and the range parameter $R = \mathbf{u}^\ell$, it generates the bilinear group by $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}) \leftarrow \mathcal{G}(1^\lambda)$. It randomly picks generators $g, g_0 \in \mathbb{G}_1$, $\hat{g} \in \mathbb{G}_2$ and $\mathfrak{X} \in \mathbb{Z}_p$. It computes $\hat{\mathfrak{Y}} = \hat{g}^{\mathfrak{X}}$, $\mathfrak{A}_i = g^{\frac{1}{\mathfrak{X}+i}}$ for $i \in [0, \mathbf{u} - 1]$. Suppose $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ is a collision resistant hash function. In addition, suppose the auditor picks a random secret key ask_{tp} in \mathbb{Z}_p and outputs its public key $h_{\text{tp}} = g^{\text{ask}_{\text{tp}}}$. It outputs $\text{param} = (g, g_0, h_{\text{tp}}, \hat{g}, \hat{e}(g, \hat{g}), \hat{\mathfrak{Y}}, \mathbf{u}, \ell, H, \mathfrak{A}_0, \dots, \mathfrak{A}_{\mathbf{u}-1})$.

- **TxPrivacySpend.** On input n' output transaction amount $M_{\text{out},j}$, n input ciphertext $C_{\text{in},i}$, amount $M_{\text{in},i}$ and randomness $r_{\text{in},i}$, it outputs \perp if $C_{\text{in},i}$ is

not a valid ciphertext for $(M_{\text{in},i}, r_{\text{in},i})$ for some $i \in [1, n]$, or $\sum_{i=1}^n M_{\text{in},i} \neq \sum_{j=1}^{n'} M_{\text{out},j}$.

For each $M_{\text{out},j}$, it obtains $(C_{\text{out},j}, r_{\text{out},j}, \pi_{\text{out},j})$ by running the sub-protocol $\text{EncProof}(M_{\text{out},j})$.

EncProof. Suppose that the prover wants to prove some M lies in $[0, \mathbf{u}^\ell - 1]$. It runs as follows:

1. It first decomposes M into $\mu_k \in [0, \mathbf{u} - 1]$ such that $M = \sum_{k=0}^{\ell-1} \mu_k \mathbf{u}^k$.
2. For each μ_k , the prover computes the ElGamal ciphertext $(C_k = g_0^{\mu_k} h_{\text{tp}}^{r_k}, B_k = g^{r_k})$ for some random $r_k \in \mathbb{Z}_p$. Denote $C_{\text{enc}} = \{C_k, B_k\}_{k \in [0, \ell-1]}$ and $r_{\text{enc}} = \sum_{k=0}^{\ell-1} r_k \mathbf{u}^k$.
3. For each μ_k , it proves in zero-knowledge that the encrypted μ_k corresponds to some \mathfrak{A}_{μ_k} :

$$\pi_{\text{enc}} \leftarrow \text{PoK}\left(\left\{\left\{\mu_k, r_k, \mathfrak{A}_{\mu_k}\right\}_{k \in [0, \ell-1]}\right\} : \hat{e}(g, \hat{g}) = \hat{e}(\mathfrak{A}_{\mu_k}, \hat{g}^{\mu_k} \cdot \hat{\mathfrak{Q}}) \wedge C_k = g_0^{\mu_k} h_{\text{tp}}^{r_k} \wedge B_k = g^{r_k}\right).$$

Details of the zero-knowledge proof is as follows. The prover randomly picks $v_k, s_k, t_k, \nu \in \mathbb{Z}_p$ for $k \in [0, \ell - 1]$ and computes:

$$V_k = \mathfrak{A}_{\mu_k}^{v_k}, \quad a_k = \hat{e}(V_k, \hat{g})^{-s_k} \cdot \hat{e}(g, \hat{g})^{t_k}, \quad E_k = g_0^{s_k} h_{\text{tp}}^{\nu_k}, \quad D_k = g^{\nu_k}.$$

It computes $\tilde{c} = H(\text{param}, \{V_k, a_k, B_k, C_k, D_k, E_k\}_{k \in [0, \ell-1]})$ and for $k \in [0, \ell - 1]$:

$$z_{\mu_k} = s_k - \tilde{c} \mu_k, \quad z_{v_k} = t_k - \tilde{c} v_k, \quad z_{r_k} = \nu_k - \tilde{c} r_k.$$

Then $\pi_{\text{enc}} = (\{V_k, z_{\mu_k}, z_{v_k}, z_{r_k}\}_{k \in [0, \ell-1]}, \tilde{c})$.

4. It outputs $(C_{\text{enc}}, r_{\text{enc}}, \pi_{\text{enc}})$.

Observe that $C_{\text{out},j} = \{C_{j,k}, B_{j,k}\}_{k \in [0, \ell-1]}$. For simplicity, denote $C'_{\text{out},j} = \prod_{k=0}^{\ell-1} C_{j,k}^{\mathbf{u}^k}$ and $B'_{\text{out},j} = \prod_{k=0}^{\ell-1} B_{j,k}^{\mathbf{u}^k}$. The same definition applies for input ciphertext.

Next, it proves that the total input Tx amount is equal to the total output Tx amount. It is equivalent to know $x_{\text{tp}} = \sum_{j=1}^{n'} r_{\text{out},j} - \sum_{i=1}^n r_{\text{in},i}$, such that

$$\prod_{j=1}^{n'} C'_{\text{out},j} / \prod_{i=1}^n C'_{\text{in},i} = h_{\text{tp}}^{x_{\text{tp}}}.$$

The zero knowledge proof of x_{tp} is as follows. It picks some random $r_{\text{tp}} \in \mathbb{Z}_p$ and computes $R_{\text{tp}} = h_{\text{tp}}^{r_{\text{tp}}}$, $\tilde{R}_{\text{tp}} = g^{r_{\text{tp}}}$, $\tilde{c}' = H(\text{param}, R_{\text{tp}}, \tilde{R}_{\text{tp}}, \{C_{\text{in},i}\}_{i \in [1,n]}, \{C_{\text{out},j}, \pi_{\text{out},j}\}_{j \in [1,n']})$, $z_{\text{tp}} = r_{\text{tp}} + \tilde{c}' x_{\text{tp}}$. Denote $\pi_{\text{tp}} = (z_{\text{tp}}, \tilde{c}', \{\pi_{\text{out},j}\}_{j \in [1,n']})$.

The algorithm outputs $(\{C_{\text{out},j}, r_{\text{out},i}\}_{j \in [1,n']}, \pi_{\text{tp}})$.

- TxPrivacyVerify. On input n input ciphertext $C_{\text{in},i}$, n' output ciphertext $C_{\text{out},j}$ and a proof $\pi_{\text{tp}} = (z_{\text{tp}}, \tilde{c}', \{\pi_{\text{out},j}\}_{j \in [1,n']})$. For each $\pi_{\text{out},j}$, it runs the following sub-protocol:

EncVerify. On input the ciphertext $C_{\text{out}} = \{C_k, B_k\}_{k \in [0,\ell-1]}$ and the proof $\pi_{\text{out}} = (\{V_k, z_{\mu_k}, z_{v_k}, z_{r_k}\}_{k \in [0,\ell-1]}, \tilde{c})$, it validates the proof by computing for all $k \in [0, \ell - 1]$:

$$D_k = B_k^{\tilde{c}} g^{z_{r_k}}, \quad E_k = C_k^{\tilde{c}} g_0^{z_{\mu_k}} h_{\text{tp}}^{z_{r_k}}, \quad a_k = \hat{e}(V_k, \hat{\mathcal{Y}}^{\tilde{c}} \hat{g}^{-z_{\mu_k}}) \cdot \hat{e}(g, \hat{g})^{z_{v_k}}.$$

It outputs 1 if $\tilde{c} = H(\text{param}, \{V_k, a_k, B_k, C_k, D_k, E_k\}_{k \in [0,\ell-1]})$ or 0 otherwise.

It computes $R'_{\text{tp}} = h_{\text{tp}}^{z_{\text{tp}}} \left(\frac{\prod_{i=1}^n C'_{\text{in},i}}{\prod_{j=1}^{n'} C'_{\text{out},j}} \right) \tilde{c}'$, $\tilde{R}'_{\text{tp}} = g^{z_{\text{tp}}} \left(\frac{\prod_{i=1}^n B'_{\text{in},i}}{\prod_{j=1}^{n'} B'_{\text{out},j}} \right) \tilde{c}'$. It returns 1 if and only if all EncVerify outputs 1, and $\tilde{c}' = H(\text{param}, R'_{\text{tp}}, \tilde{R}'_{\text{tp}}, \{C_{\text{in},i}\}_{i \in [1,n]}, \{C_{\text{out},j}, \pi_{\text{enc},j}\}_{j \in [1,n']})$.

- Decrypt. On input the auditor's secret key ask_{tp} and a ciphertext $C_{\text{enc}} = \{C_k, B_k\}_{k \in [0,\ell-1]}$, it computes $g_0^{\mu_k} = \frac{C_k}{B_k^{\text{ask}_{\text{tp}}}}$ for $k \in [0, \ell - 1]$. The auditor uses a pre-computation table containing $(g_0^0, g_0^1, \dots, g_0^{u-1})$ to find out the value of μ_k . Finally, the auditor recovers $M = \sum_{k=0}^{\ell-1} \mu_k \mathbf{u}^k$.

Security of Transaction Privacy. We give the security theorem of our TP protocol. The proofs are given in the appendix.

Theorem 1. *Our TP protocol is sound if the \mathbf{u} -Strong Diffie-Hellman (SDH) assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$ in the random oracle model. Our TP protocol is private if the decisional Diffie-Hellman (DDH) assumption holds in \mathbb{G}_1 in the random oracle model. Our TP protocol is balance if the discrete logarithm (DL) assumption holds in \mathbb{G}_1 in the random oracle model.*

5. Recipient Privacy

In blockchain, the user address is the hash of his public key, and hence it represents his identity. If we want to preserve the recipient privacy, we

can always use a new public key for each Tx. However, this approach is problematic in some consortium blockchain which only allows Tx's between authenticated users. It means that all recipient (and sender) address should be authenticated. A straightforward approach is to associate each address with a certificate issued by a CA. The key challenge is how to validate the certificate while hiding the public key/address at the same time.

Previous Works. Dash's PrivateSend is a coin-mixing service based on CoinJoin [17]. Dash requires combining identical input amounts from multiple senders at the time of mixing, and thus it restricts the mixing to only accept certain denominations (e.g. \$0.1, \$1, \$10, etc.). The level of anonymity is related to number of Tx's mixed. In Monero, the recipient uses *stealth address* [6], which is a one-time DH-type public key computed from the recipient's public key and some randomness included in the transaction block. The corresponding one-time secret key is only computable by the recipient. In Zcash, it uses the general zk-SNARK to provide zero knowledge for all Tx details including UTXO used [7].

5.1. Security Model of Recipient Privacy Protocol

The formal security notion and security models are defined as follows.

Recipient Privacy Protocol. A recipient privacy protocol consists of a tuple of poly-time algorithms as described follows:

1. $(\text{param}, \text{ask}_{\text{rp}}) \leftarrow \text{Setup}(1^\lambda)$. On input a security parameter 1^λ , it outputs the auditor secret key ask_{rp} and the system parameter param (which includes the auditor public key). Suppose param is the input of all other algorithms, and hence is omitted for simplicity.
2. $(\text{usk}, \text{upk}) \leftarrow \text{UserKeyGen}()$. The user outputs his long-term secret key usk and long-term public key upk .
3. $(\text{otpk}, R_{tx}) \leftarrow \text{OneTimePkGen}(\text{upk})$. On input upk , it outputs a one-time public key otpk and a transaction randomness R_{tx} .
4. $\text{otsk}/\perp \leftarrow \text{OneTimeSkGen}(\text{otpk}, R_{tx}, \text{usk})$. On input upk , R_{tx} and usk , it outputs a one-time secret key otsk or \perp if otpk is not related to usk .
5. $(\text{cask}, \text{capk}) \leftarrow \text{CAKeyGen}()$. The CA outputs his secret key cask and public key capk .
6. $\text{cert} \leftarrow \text{CertIssue}(\text{capk}, \text{upk})$. This is an interactive algorithm runs between the CA and a user, with common input capk and upk . Each party additionally takes its own secret key as private input. After the interaction, the CA returns the certificate cert to the user.

7. $(\pi_{\text{otsk}}, \text{otpk}_r, R_{tx,r}, C_{rp}, \pi_{rp}) \leftarrow \text{RecPrivacySpend}(\text{capk}, \text{otpk}_s, R_{tx,s}, \text{usk}, \text{upk}_r, \text{cert}_r)$. On input capk , the sender's input UTXO (including his otpk_s and $R_{tx,s}$), the sender's usk ⁷, the recipient's long term public key upk_r and his certificate cert_r , the sender runs as follows:
 - (a) It first runs $\text{otsk} \leftarrow \text{OneTimeSkGen}(\text{otpk}_s, R_{tx,s}, \text{usk})$. It outputs a zero-knowledge proof π_{otsk} of knowing otsk .
 - (b) It also outputs $(\text{otpk}_r, R_{tx,r}) \leftarrow \text{OneTimePkGen}(\text{upk}_r)$.
 - (c) It also outputs a ciphertext C_{rp} encrypting upk_r to the auditor, and a proof π_{rp} showing that C_{rp} encrypts upk_r and $\text{cert}_r \leftarrow \text{CertIssue}(\text{capk}, \text{upk}_r)$.

The sender outputs $(\pi_{\text{otsk}}, \text{otpk}_r, R_{tx,r}, C_{rp}, \pi_{rp})$.⁸

8. $1/0 \leftarrow \text{Verify}(\text{capk}, \pi_{\text{otsk}}, \text{otpk}_s, \text{otpk}_r, R_{tx,r}, C_{rp}, \pi_{rp})$. On input capk , the sender's otpk_s with the proof of corresponding one-time secret key π_{otsk} , the recipient's otpk_r , the transaction randomness $R_{tx,r}$, a ciphertext C_{rp} and a proof π_{rp} , it outputs 1 if π_{otsk} and π_{rp} are valid, and outputs 0 otherwise.
9. $\text{upk}_r \leftarrow \text{Decrypt}(\text{ask}_{rp}, C_{rp})$. On input the auditor secret key ask_{rp} and a ciphertext C_{rp} , it outputs the decrypted public key upk_r .

Security Model. We first define the security requirements for recipient privacy:

1. No unauthorized recipient can spend their received money. (Note that in this model, we allow the adversary to transfer money to arbitrary unauthorized address. However, the corresponding secret key is unknown to the adversary and hence the adversary has no motivation to transfer money in this way.)
2. No one can learn the identity of the recipient, except the auditor.

Definition 4 (Soundness). *A recipient privacy protocol is called sound if for all PPT adversary \mathcal{A} , it holds that given $(\text{param}, \text{ask}_{rp}, \text{capk})$, where $(\text{param},$*

⁷For coinbase transaction, the input UTXO and the sender's key are not needed. The verification of coinbase transaction also does not need to check the input UTXO. We omit the description for coinbase transaction for simplicity.

⁸In this section, we model the recipient privacy as a single input, single output transaction. It can be easily generalized into multiple input, multiple output transaction. We use the simplified model for the ease of understanding.

$\text{ask}_{\text{rp}} \leftarrow \text{Setup}(1^\lambda)$; $(\text{cask}, \text{capk}) \leftarrow \text{CAKeyGen}()$, \mathcal{A} outputs $(\pi_{\text{otsk}}, \text{otpk}_s, \text{otpk}^*, R_{tx}^*, C_{\text{rp}}^*, \pi_{\text{rp}}^*)$, $(\pi_{\text{otsk}}^*, \text{otpk}^*, \text{otpk}_r, R_{tx,r}, C_{\text{rp}}, \pi_{\text{rp}})$, then:

$$\Pr \left[\begin{array}{l} \text{Verify}(\text{capk}, \pi_{\text{otsk}}, \text{otpk}_s, \text{otpk}^*, R_{tx}^*, C_{\text{rp}}^*, \pi_{\text{rp}}^*) = 1 \\ \wedge \text{Verify}(\text{capk}, \pi_{\text{otsk}}^*, \text{otpk}^*, \text{otpk}_r, R_{tx,r}, C_{\text{rp}}, \pi_{\text{rp}}) = 1 \end{array} \right] \leq \text{negl}(\lambda),$$

where \mathcal{A} can query the oracles defined below:

- **KeyGen()**: it runs $(\text{usk}, \text{upk}) \leftarrow \text{UserKeyGen}()$, stores (usk, upk) in a list \mathcal{L} (which is initially empty) and outputs upk .
- **Corrupt(upk)**: on input upk , it searches $(\text{usk}, \text{upk}) \in \mathcal{L}$ and returns usk . It returns \perp if no such key is found in \mathcal{L} .
- **Issue(upk)**: on input upk , it runs as the CA of the $\text{CertIssue}(\text{capk}, \text{upk})$ protocol with private input cask . It outputs cert .
- **Spend($\text{upk}_s, \text{otpk}_s, R_{tx,s}, \text{upk}_r, \text{cert}_r$)**: on input a sender public key upk_s , the $(\text{otpk}_s, R_{tx,s})$ from the previous transaction, an authorized recipient upk_r with certificate cert_r , it firstly retrieves $(\text{usk}_s, \text{upk}_s) \in \mathcal{L}$. Then it outputs $(\pi_{\text{otsk}}, \text{otpk}_r, R_{tx,r}, C_{\text{rp}}, \pi_{\text{rp}}) \leftarrow \text{RecPrivacySpend}(\text{capk}, \text{otpk}_s, R_{tx,s}, \text{usk}_s, \text{upk}_r, \text{cert}_r)$.

We additionally require that $\text{upk}^* \leftarrow \text{Decrypt}(\text{ask}_{\text{rp}}, C_{\text{rp}}^*)$, upk^* is never queried to the **Issue** oracle, and $(\text{upk}^*, \text{otpk}^*, R_{tx}^*, \cdot, \cdot)$ is never queried to **Spend** oracle.

Definition 5 (Anonymity). A recipient privacy protocol is called anonymous if for all PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$, it holds that when \mathcal{A}_1 is given $\text{param}, \text{cask}, \text{capk}$, where $(\text{param}, \text{ask}_{\text{rp}}) \leftarrow \text{Setup}(1^\lambda)$, $(\text{cask}, \text{capk}) \leftarrow \text{CAKeyGen}()$ and outputs $(\text{otpk}_s, R_{tx,s}, \text{usk}, \text{upk}_{r,0}, \text{cert}_{r,0}, \text{upk}_{r,1}, \text{cert}_{r,1})$; and \mathcal{A}_2 is given $(C_{\text{rp}}^*, \pi_{\text{rp}}^*)$, where $(\pi_{\text{otsk}}^*, \text{otpk}^*, R_{tx,r}^*, C_{\text{rp}}^*, \pi_{\text{rp}}^*) \leftarrow \text{RecPrivacySpend}(\text{capk}, \text{otpk}_s, R_{tx,s}, \text{usk}, \text{upk}_{r,b}, \text{cert}_{r,b})$ for some random bit b , \mathcal{A}_2 outputs its guess b' such that:

$$\Pr [b' = b] \leq \text{negl}(\lambda).$$

The adversary $(\mathcal{A}_1, \mathcal{A}_2)$ is given the **KeyGen**, **Corrupt** oracles.

5.2. Recipient Privacy for PACchain

Stealth address [6] appears to be the most efficient approach for recipient privacy. However in consortium blockchain, only the recipient's public key is authenticated by the CA, but not the one-time public key. Therefore, the

sender additionally needs to show that the one-time public key is computed from an authenticated public key, without revealing the public key itself.

Our Construction. The recipient's certificate is signed by the CA using BBS+ signature [18], which allows efficient zero-knowledge proof. In addition, we encrypt the long-term public key in the zero-knowledge proof, such that the auditor can decrypt the real address (long-term public key) of the recipient.

Our recipient privacy (RP) protocol is described below.

- **Setup.** On input a security parameter 1^λ , the setup algorithm generates the bilinear group by $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}) \leftarrow \mathcal{G}(1^\lambda)$. It picks some random generators $g, g_2, g_3, h_2 \in \mathbb{G}_1$ and $\hat{g}_2 \in \mathbb{G}_2$. Suppose $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$, $H' : \mathbb{G}_1 \rightarrow \mathbb{Z}_p$ are collision resistant hash functions. In addition, suppose the auditor picks a random secret key $\text{ask}_{\text{rp}} \in \mathbb{Z}_p$ and outputs its public key $h_{\text{rp}} = g^{\text{ask}_{\text{rp}}}$. It outputs the public parameters $\text{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, g, h_{\text{rp}}, g_2, g_3, h_2, \hat{g}_2, H, H')$.

- **UserKeyGen.** The user randomly picks a long-term secret key $x_1, x_2 \in \mathbb{Z}_p$ and computes a long-term public key $Y_1 = g_2^{x_1}, Y_2 = g_2^{x_2}$. It outputs the user key pair $(\text{usk} = (x_1, x_2), \text{upk} = (Y_1, Y_2))$.

- **OneTimePkGen.** On input $\text{upk} = (Y_1, Y_2)$, the sender randomly picks $r_{tx} \in \mathbb{Z}_p$ and outputs $(R_{tx} = g_2^{r_{tx}}, \text{otpk} = Y_1 g_2^{H'(Y_2^{r_{tx}})})$.

- **OneTimeSkGen.** On input otpk, R_{tx} and $\text{usk} = (x_1, x_2)$, the recipient computes a one-time secret key $\text{otsk} = x_1 + H'(R_{tx}^{x_2})$, and it outputs $\text{otsk} = g_2^{\text{otsk}}$.

- **CAKeyGen.** The CA randomly picks $\beta \in \mathbb{Z}_p$ and computes $\hat{W}_2 = \hat{g}_2^\beta$. It outputs the CA key pair $(\text{cask} = \beta, \text{capk} = \hat{W}_2)$.

- **CertIssue.** On input CA's public key capk and the user long-term public key $\text{upk} = (Y_1, \cdot)$, the user first performs a zero-knowledge proof of discrete logarithm: $x_1 = \log_{g_2} Y_1$. Denote this proof as π_{ca} . After the CA validates π_{ca} , the CA picks some random $s, w \in \mathbb{Z}_p$ and uses his private key $\text{cask} = \beta$ to compute: $F = (h_2 \cdot Y_1 \cdot g_3^s)^{\frac{1}{\beta+w}}$. The CA returns the certificate (F, w, s) to the user.

- **RecPrivacySpend.** On input $\text{param}, \text{capk} = \hat{W}_2$:

1. The sender with usk decides one or more UTXOs that he wants to spend. For simplicity, assume he picks one UTXO with $(\text{otpk}_s, R_{tx,s})$. He runs $\text{otsk}_s \leftarrow \text{OneTimeSkGen}(\text{param}, \text{otpk}_s, R_{tx,s}, \text{usk})$. It runs a

zero-knowledge proof of discrete logarithm: $\text{otks}_s = \log_{g_2} \text{otpk}_s$. Denote this proof as π_{otsk} .

2. The sender chooses the set of recipients. For simplicity, assume there is only one recipient with long-term public key $\text{upk}_r = (Y_1, Y_2)$. The sender generates the recipient's one-time public key by running **One-TimePkGen**. The sender obtains $\text{otpk}_r, R_{tx,r}$ and the randomness r_{tx} . Denote $h_{tx} = H'(Y_2^{r_{tx}})$.
3. The sender encrypts Y_1 to the auditor by picking a random $r_{\text{cert}} \in \mathbb{Z}_p$ and computing $C_{\text{rp}} = (C_{\text{cert}} = Y_1 \cdot h_{\text{rp}}^{r_{\text{cert}}}, B_{\text{cert}} = g^{r_{\text{cert}}})$.
4. The sender runs the following proof of knowledge for showing that (1) otpk_r is computed from a public key, (2) the public key has a valid certificate (F, w, s) , (3) the public key is encrypted to the auditor:

$$\begin{aligned} \pi_{\text{rp}} \leftarrow \text{PoK}\{(F, w, s, Y_1, h_{tx}, r_{\text{cert}}) : \hat{e}(F, \hat{g}_2^w \cdot \hat{W}_2) = \hat{e}(h_2 \cdot Y_1 \cdot g_3^s, \hat{g}_2) \\ \wedge \text{otpk}_r = Y_1 g_2^{h_{tx}} \wedge B_{\text{cert}} = g^{r_{\text{cert}}} \wedge C_{\text{cert}} = Y_1 h_{\text{rp}}^{r_{\text{cert}}}\}. \end{aligned}$$

The details of the zero knowledge proof π_{rp} is as follows.

- (a) **ZKCommit**: It picks some random $\rho, r_\tau, r_\omega, r_\sigma, r_\rho, r_{\text{cert}}, r_c, r_s \in \mathbb{Z}_p$, computes $\Theta = F^\rho$ and

$$\begin{aligned} R_{\text{cert},1} &= \hat{e}((h_2 C_{\text{cert}})^{r_\rho} g_3^{r_s} \Theta^{-r_\omega} h_{\text{rp}}^{-r_\sigma}, \hat{g}_2), \\ R_{\text{cert},2} &= g^{r_c}, \quad R_{\text{cert},3} = B_{\text{cert}}^{r_\rho} g^{-r_\sigma}, \quad R_{\text{cert},4} = h_{\text{rp}}^{r_c} g_2^{-r_\tau}. \end{aligned}$$

- (b) **ZKChallenge**: It computes $c = H(\text{CertAuth.mpk}, C_{\text{rp}}, \Theta, R_{\text{cert},1}, R_{\text{cert},2}, R_{\text{cert},3}, R_{\text{cert},4})$.
- (c) **ZKResponse**: It computes:

$$\begin{aligned} z_\omega &= r_\omega + c \cdot w, & z_\tau &= r_\tau + c \cdot h_{tx}, & z_\rho &= r_\rho + c \cdot \rho, \\ z_c &= r_c + c \cdot r_{\text{cert}}, & z_\sigma &= r_\sigma + c \cdot r_{\text{cert}} \cdot \rho, & z_s &= r_s + c \cdot \rho \cdot s. \end{aligned}$$

It outputs $\pi_{\text{rp}} = (c, \Theta, z_\omega, z_\tau, z_\rho, z_c, z_\sigma, z_s)$.

5. Output $\pi_{\text{otsk}}, \text{otpk}_s, \text{otpk}_r, R_{tx,r}, C_{\text{rp}}$ and π_{rp} .

- **RecPrivacyVerify**. On input $\text{param}, \text{capk}, \pi_{\text{otsk}}, \text{otpk}_s, \text{otpk}_r, R_{tx,r}, C_{\text{rp}}$ and π_{rp} , it outputs 1 if π_{otsk} and π_{rp} are valid zero knowledge proofs.

The details of verifying the zero knowledge proof $\pi_{\text{rp}} = (c, \Theta, z_\omega, z_\tau, z_\rho, z_c, z_\sigma, z_s)$ is as follows.

1. ZKReconstruct: Denote $C_{\text{rp}} = (C_{\text{cert}}, B_{\text{cert}})$. It computes:

$$\begin{aligned} R_{\text{cert},1} &= \hat{e}((h_2 \cdot C_{\text{cert}})^{z_\rho} g_3^{z_s} \Theta^{-z_\omega} h_{\text{rp}}^{-z_\sigma}, \hat{g}_2) \cdot \hat{e}(\Theta, \hat{W}_2)^c, \\ R_{\text{cert},2} &= g^{z_c} B_{\text{cert}}^{-c}, \quad R_{\text{cert},3} = B_{\text{cert}}^{z_\rho} g^{-z_\sigma}, \quad R_{\text{cert},4} = h_{\text{rp}}^{z_c} g_2^{-z_\tau} (\text{otpk}_r / C_{\text{cert}})^c. \end{aligned}$$

2. ZKCheck: It computes $c' = H(\text{CertAuth.mpk}, C_{\text{rp}}, \Theta, R_{\text{cert},1}, R_{\text{cert},2}, R_{\text{cert},3}, R_{\text{cert},4})$. If $c = c'$, then π_{rp} is a valid zero knowledge proof.

Security of Recipient Privacy. We give the security theorem of our RP protocol. The proofs are given in the appendix.

Theorem 2. *Our RP protocol is sound if the q -SDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$ in the random oracle model, where q is the maximum number of issue oracle query. Our RP protocol is anonymous if the DDH assumption holds in \mathbb{G}_1 in the random oracle model.*

6. Sender Privacy

In the UTXO model, the sender has to specify the UTXOs that he wants to use. The UTXOs include the information of the owner's address as well as the transaction amount. The linkage between the current transaction and UTXOs guarantees the validity of the transaction and ensures that there is no double spending. However, this linkage violates the privacy of the sender (no matter the address is used for one time only and the transaction amount is encrypted). It is a dilemma to preserve the transaction correctness and to protect the sender privacy at the same time.

Previous Works. The sender privacy for Dash and Zcash are achieved as the same way as the recipient privacy. In Monero, it uses linkable ring signature for hiding the real UTXOs used with other UTXOs. The level of anonymity is related to number of UTXOs (denote as L) included. However, the number of computation used in signing and the signature size are both $O(L)$. Recently, Sun *et al.* reduced the signature size to $O(1)$ [19], at the price of using trusted setup.

6.1. Security Model of Sender Privacy Protocol

The formal security notion and models are given as follows.

Sender Privacy Protocol. A sender privacy protocol consists of a tuple of poly-time algorithms as described follows:

1. $(\mathbf{param}, \mathbf{ask}_{rp}) \leftarrow \mathbf{Setup}(1^\lambda)$. On input a security parameter 1^λ , it outputs the auditor secret key \mathbf{ask}_{sp} and the system parameter \mathbf{param} (which includes the auditor public key). Suppose \mathbf{param} is the input of all other algorithms, and hence is omitted for simplicity.
2. $(\mathbf{usk}, \mathbf{upk}) \leftarrow \mathbf{UserKeyGen}()$. The user outputs his long-term secret key \mathbf{usk} and long-term public key \mathbf{upk} .
3. $(\mathbf{esk}, \mathbf{epk}) \leftarrow \mathbf{EndorserKeyGen}()$. The endorser outputs his secret key \mathbf{esk} and public key \mathbf{epk} .
4. $\mathbf{cred} \leftarrow \mathbf{CredIssue}(\mathbf{epk}, \mathbf{upk}, C_{tp})$. This is an interactive algorithm runs between the endorser and a user, with common input $\mathbf{epk}, \mathbf{upk}$ and the ciphertext of the transaction amount C_{tp} . Each party additionally takes its own secret key as private input. After the interaction, the endorser returns the credential \mathbf{cred} to the user.
5. $(C_{sp}, T, \pi_{sp}) \leftarrow \mathbf{CredSign}(\mathbf{usk}, \mathbf{upk}, \mathbf{cred}, m, r)$. On input a sender's key $\mathbf{usk}, \mathbf{upk}$, the credential \mathbf{cred} corresponding to the input transaction amount m encrypted with randomness r , it runs as follows:
 - (a) It encrypts \mathbf{upk} to the auditor and obtains the ciphertext C_{sp} .
 - (b) It computes a linking tag T from \mathbf{usk} .
 - (c) It generates a zero-knowledge proof π_{sp} for (1) the correctness of C_{sp}, T above; (2) knowing $\mathbf{cred}, m, r, \mathbf{upk}$ such that $(C_{tp}, r) = \mathbf{EncProof.Enc}(m)$ and $\mathbf{cred} = \mathbf{CredIssue}(\mathbf{epk}, \mathbf{upk}, C_{tp})$.

It outputs (C_{sp}, T, π_{sp}) .

6. $1/0 \leftarrow \mathbf{Link}(T_1, T_2)$. On input two tags T_1 and T_2 , it outputs 1 if they are linked (computed by the same secret key) or 0 otherwise.
7. $1/0 \leftarrow \mathbf{Verify}(\mathbf{epk}, C_{sp}, T, \pi_{sp})$. On input \mathbf{epk} , a ciphertext C_{sp} , a tag T , and a proof π_{sp} , it outputs 1 if π_{sp} is valid, and outputs 0 otherwise.
8. $\mathbf{upk}_s \leftarrow \mathbf{Decrypt}(\mathbf{ask}_{sp}, C_{sp})$. On input the auditor secret key \mathbf{ask}_{rp} and a ciphertext C_{sp} , it outputs the decrypted public key \mathbf{upk}_s .

Security Model. We define the security requirements for sender privacy:

1. No adversary can spend without credential, even with colluding auditor.
2. No adversary can spend money of honest user, even with colluding endorser and auditor.
3. No one can learn the identity of the sender, except the auditor.

Definition 6 (Soundness). *A sender privacy protocol is sound if for all PPT adversary \mathcal{A} , it holds that:*

$$\Pr \left[\begin{array}{l} \text{Verify}(\text{epk}, C_{\text{sp}}^*, \\ T^*, \pi_{\text{sp}}^*) = 1 \end{array} : \begin{array}{l} (\text{param}, \text{ask}_{\text{sp}}) \leftarrow \text{Setup}(1^\lambda); \\ (\text{esk}, \text{epk}) \leftarrow \text{EndorserKeyGen}(); \\ (C_{\text{sp}}^*, T^*, \pi_{\text{sp}}^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\text{param}, \text{ask}_{\text{sp}}, \text{epk}) \end{array} \right] \leq \text{negl}(\lambda),$$

where $\mathcal{O} = (\text{Issue}_e, \text{Sign})$. *The issue and sign oracle are defined as below:*

- $\text{Issue}_e(\text{upk})$: *on input upk , it runs as the endorser of the $\text{CredIssue}(\text{epk}, \text{upk})$ protocol with private input esk and it outputs cred .*
- $\text{Sign}(\text{upk}, \text{usk}, m, r)$: *on input a sender's public key upk , secret key usk , the input transaction amount m encrypted with randomness r , it generates the credential cred corresponding to upk, m, r . Then it outputs $(C_{\text{sp}}, T, \pi_{\text{sp}}) \leftarrow \text{CredSign}(\text{usk}, \text{upk}, \text{cred}, m, r)$.*

We additionally require that $\text{upk}^ \leftarrow \text{Decrypt}(\text{ask}_{\text{sp}}, C_{\text{sp}})$ and upk^* is never queried to the Issue_e oracle, and $(C_{\text{sp}}^*, T^*, \pi_{\text{sp}}^*)$ is not the output of Sign oracle.*

Definition 7 (Unforgeability). *A sender privacy protocol is unforgeable if for all PPT adversary \mathcal{A} , it holds that:*

$$\Pr \left[\begin{array}{l} \text{Verify}(\text{epk}, C_{\text{sp}}^*, \\ T^*, \pi_{\text{sp}}^*) = 1 \end{array} : \begin{array}{l} (\text{param}, \text{ask}_{\text{sp}}) \leftarrow \text{Setup}(1^\lambda); \\ (\text{esk}, \text{epk}) \leftarrow \text{EndorserKeyGen}(); \\ (C_{\text{sp}}^*, T^*, \pi_{\text{sp}}^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\text{param}, \text{ask}_{\text{sp}}, \text{epk}, \text{esk}) \end{array} \right] \leq \text{negl}(\lambda),$$

where $\mathcal{O} = (\text{KeyGen}, \text{Corrupt}, \text{Issue}_u, \text{Sign})$. *The issue and sign oracle are defined as below:*

- $\text{Issue}_u(\text{upk})$: *on input upk , it runs as the user of the $\text{CredIssue}(\text{epk}, \text{upk})$ protocol with private input usk , where $(\text{usk}, \text{upk}) \in \mathcal{L}$.*
- $\text{Sign}(\text{upk}, \text{cred}, m, r)$: *on input a sender's public key upk , its credential cred corresponding to the input transaction amount m encrypted with randomness r , it firstly retrieves $(\text{usk}, \text{upk}) \in \mathcal{L}$. Then it outputs $(C_{\text{sp}}, T, \pi_{\text{sp}}) \leftarrow \text{CredSign}(\text{usk}, \text{upk}, \text{cred}, m, r)$.*

We additionally require that $\text{upk}^ \leftarrow \text{Decrypt}(\text{ask}_{\text{sp}}, C_{\text{sp}})$ and upk^* is the output of the KeyGen oracle but is never queried to the Corrupt oracle, and $(C_{\text{sp}}^*, T^*, \pi_{\text{sp}}^*)$ is not the output of Sign oracle.*

Definition 8 (Anonymity). *A sender privacy protocol is anonymous if for all PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, it holds that:*

$$\Pr \left[\begin{array}{l} (\text{param}, \text{ask}_{rp}) \leftarrow \text{Setup}(1^\lambda); \\ (\text{esk}, \text{epk}) \leftarrow \text{EndorserKeyGen}(); b \leftarrow \{0, 1\}; \\ b' = b : (\{\text{upk}_i^*, \text{cred}_i^*, m_i^*, r_i^*\}_{i \in [0,1]}) \leftarrow \mathcal{A}_1^\mathcal{O}(\text{param}, \text{epk}, \text{esk}); \\ (C_{sp}^*, T^*, \pi_{sp}^*) \leftarrow \text{CredSign}(\text{usk}_b^*, \text{upk}_b^*, \text{cred}_b^*, m_b^*, r_b^*), \\ b' \leftarrow \mathcal{A}_2^\mathcal{O}(\text{param}, \text{esk}, \text{epk}, C_{sp}^*, T^*, \pi_{sp}^*) \end{array} \right] \leq \text{negl}(\lambda).$$

where $\mathcal{O} = (\text{KeyGen}, \text{Corrupt})$, $\text{upk}_0^*, \text{upk}_1^*$ are the output of the **KeyGen** oracle, they are not queried to the **Corrupt** oracle, and their corresponding secret key are denoted as $\text{usk}_0^*, \text{usk}_1^*$.

6.2. Sender Privacy for PACHain

We give our efficient sender privacy solution for consortium blockchain. By the semi-trusted property of consortium blockchain, we can use the anonymous credential approach to achieve sender privacy. By using the semi-trusted endorser as the group manager (in the honest-but-curious security model), we provide an efficient solution which has the signing time, verification time and signature size independent to the number of UTXO included in the group. At the same time, the sender can be revealed by the auditor. Note that similar to group signature, the endorser (who issued credentials) cannot link the transaction by the credential he issued. Credential is issued to the recipient when the endorser approve the transaction. The endorser does not have any advantage in breaking anonymity in the UTXO model.

Our Construction. Our construction differs from traditional group signatures in two ways: (1) we have to hide both the sender's public key as well as the Tx amount, (2) we have to add a linkability tag to avoid double spending. For the first requirement, we use the BBS group signature [20], since the underlying credential is signed by Boneh-Boyen signature [21], which can be modified to sign on multiple committed values [18]. For the second requirement, we use the tag structure used in linkable ring signatures.

There are two possible constructions: the Tx amount is in plaintext or in ciphertext. For simplicity, we only give the Tx amount ciphertext version here and the plaintext version can be constructed similarly.

- **Setup.** On input a security parameter 1^λ , the setup algorithm generates the bilinear group by $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}) \leftarrow \mathcal{G}(1^\lambda)$. It picks some random generators $g, g_1, g_2, u_1, h_s, f \in \mathbb{G}_1$ and $\hat{g}_2 \in \mathbb{G}_2$. Suppose $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$

is a collision resistant hash function. Denote h_{tp} as the public key of the auditor in transaction privacy. Suppose the auditor picks a random secret key ask_{sp} in \mathbb{Z}_p and outputs its public key $h_{\text{sp}} = g^{\text{ask}_{\text{sp}}}$. It outputs the public parameters $\text{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, g, g_1, g_2, u_1, h_s, f, h_{\text{tp}}, h_{\text{sp}}, \hat{g}_2, H)$.

- **UserKeyGen.** The user secret key is $\text{usk} = x_1 \in \mathbb{Z}_p$ and the public key is $\text{upk} = Y_1 = g_2^{x_1}$.⁹

- **EndorserKeyGen.** The endorser randomly picks $\alpha \in \mathbb{Z}_p$ and computes $W_{\text{sp}} = \hat{g}_2^\alpha$. It outputs the endorser key pair ($\text{esk} = \alpha, \text{epk} = W_{\text{sp}}$).

- **CredIssue.** On input endorser public key epk , the user public key Y_1 and the Tx amount ciphertext C , the user first performs a zero-knowledge proof of secret key: $x_1 = \log_{g_2} Y_1$. Denote this proof as π_{ci} . The user sends π_{ci} and π_{tp} to the endorser, where π_{tp} is the zero-knowledge transaction privacy proof (showing the knowledge of (m, r_{tp}) such that $C = g^m h_{\text{tp}}^{r_{\text{tp}}}$).

After the endorser validates the proofs π_{ci} and π_{tp} , the endorser picks some random $v, z \in \mathbb{Z}_p$ and uses his secret key $\text{esk} = \alpha$ to compute: $A = (h_s \cdot g_1^v \cdot C \cdot Y_1)^{\frac{1}{\alpha+z}}$. The endorser returns the credential $\text{cred} = (A, v, z)$ to the user.

- **CredSign.** On input param , and private input tuples $x'_{\text{in}}, \text{cred}_{\text{in}}, m'_{\text{in}}, r'_{\text{in}}, Y'_{\text{in}}$ (such that $C_{\text{in}} = g^{m'_{\text{in}}} h_{\text{tp}}^{r'_{\text{in}}}$), it runs the following:

1. It computes the tag for detecting double spending: $T = f^{x'_{\text{in}}}$.
2. It encrypts the public key Y'_{in} to the auditor, by randomly choosing $r_{\text{cred}} \in \mathbb{Z}_p$ and computing $C_{\text{sp}} = (C_{\text{cred}} = Y'_{\text{in}} \cdot h_{\text{sp}}^{r_{\text{cred}}}, B_{\text{cred}} = g^{r_{\text{cred}}})$.
3. It computes the zero knowledge proof π_{sp} for: (1) the credential $\text{cred}_{\text{in}} = (A, v, z)$ corresponds to $Y'_{\text{in}} = g_2^{x'_{\text{in}}}$ and $C_{\text{in}} = g^{m'_{\text{in}}} h_{\text{tp}}^{r'_{\text{in}}}$; (2) $T = f^{x'_{\text{in}}}$; (3) Y'_{in} is encrypted to the auditor.

$$\pi_{\text{sp}} = \text{PoK} \{ (x'_{\text{in}}, m'_{\text{in}}, r'_{\text{in}}, A, v, z, r_{\text{cred}}) : \hat{e}(A, W_{\text{sp}} \hat{g}_2^z) = \hat{e}(h_s g_1^v g^{m'_{\text{in}}} h_{\text{tp}}^{r'_{\text{in}}} g_2^{x'_{\text{in}}}, \hat{g}_2) \wedge T = f^{x'_{\text{in}}} \wedge C_{\text{cred}} = g_2^{x'_{\text{in}}} \cdot h_{\text{sp}}^{r_{\text{cred}}} \wedge B_{\text{cred}} = g^{r_{\text{cred}}} \}.$$

The output signature $\sigma = (\pi_{\text{sp}}, C_{\text{sp}}, T)$. Details of the zero-knowledge proof is shown as follows.

⁹This public key Y_1 can be a long term public key if recipient anonymity is not protected in the previous transaction. Otherwise, it can be a one-time public key.

- (a) **ZKCommit**: It picks some random $a, r_\psi, r_k, r_a, r_b, r_z, r_m, r_r, r_v \in \mathbb{Z}_p$. It computes:

$$\begin{aligned} S &= A \cdot u_1^a, \quad \Xi = g_1^a, \\ R_{\text{cred},1} &= \hat{e}(u_1^{r_b} S^{-r_z} g_1^{r_v} g^{r_m} h_{\text{tp}}^{r_r} g_2^{r_k}, \hat{g}_2) \cdot \hat{e}(u_1, W_{\text{sp}})^{r_a}, \quad R_{\text{cred},2} = g_1^{r_a}, \\ R_{\text{cred},3} &= \Xi^{r_z} g_1^{-r_b}, \quad R_{\text{cred},4} = g^{r_\psi}, \quad R_{\text{cred},5} = g_2^{r_k} h_{\text{sp}}^{r_\psi}, \quad R_{\text{cred},6} = f^{r_k}. \end{aligned}$$

- (b) **ZKChallenge**: It computes $c = H(\text{CredAuth.mpk}, C_{\text{sp}}, T, S, \Xi, R_{\text{cred},1}, R_{\text{cred},2}, R_{\text{cred},3}, R_{\text{cred},4}, R_{\text{cred},5}, R_{\text{cred},6})$.
(c) **ZKResponse**: It computes:

$$\begin{aligned} z_k &= r_k + c \cdot x'_{\text{in}}, & z_a &= r_a + c \cdot a, & z_z &= r_z + c \cdot z, \\ z_b &= r_b + c \cdot a \cdot z, & z_v &= r_v + c \cdot v, & z_m &= r_m + c \cdot m'_{\text{in}}, \\ z_r &= r_r + c \cdot r'_{\text{in}}, & z_\psi &= r_\psi + c \cdot r_{\text{cred}}. \end{aligned}$$

It outputs the proof $\pi_{\text{sp}} = (c, S, \Xi, z_k, z_a, z_z, z_b, z_v, z_m, z_r, z_\psi)$.

- **Verify**. On input **param**, the endorser public keys W_s , a signature $\sigma = (\pi_{\text{sp}}, C_{\text{sp}} = (C_{\text{cred}}, B_{\text{cred}}), T)$, it checks the validity of the proof $\pi_{\text{sp}} = (c, S, \Xi, z_k, z_a, z_z, z_b, z_v, z_m, z_r, z_\psi)$:

1. **ZKReconstruct**: It computes:

$$\begin{aligned} R'_{\text{cred},1} &= \hat{e}(u_1^{z_b} S^{-z_z} g_1^{z_v} g^{z_m} h_{\text{tp}}^{z_r} g_2^{z_k} h_s^c, \hat{g}_2) \cdot \hat{e}(u_1^{z_a} S^{-c}, W_{\text{sp}}), \\ R'_{\text{cred},2} &= g_1^{z_a} \Xi^{-c}, \quad R'_{\text{cred},3} = \Xi^{z_z} g_1^{-z_b}, \quad R'_{\text{cred},4} = g^{z_\psi} B_{\text{cred}}^{-c}, \\ R'_{\text{cred},5} &= g_2^{z_k} h_{\text{sp}}^{z_\psi} C_{\text{cred}}^{-c}, \quad R'_{\text{cred},6} = f^{z_k} T^{-c}. \end{aligned}$$

2. **ZKCheck**: It computes $c' = H(\text{CredAuth.mpk}, C_{\text{sp}}, T, S, \Xi, R_{\text{cred},1}, R_{\text{cred},2}, R_{\text{cred},3}, R_{\text{cred},4}, R_{\text{cred},5}, R_{\text{cred},6})$.

It outputs 1 if $c = c'$; and outputs 0 otherwise.

- **Link**. On input **param** and two tags T_1, T_2 in signatures σ_1, σ_2 , such that $T_1 = T_2$, it outputs 1. Otherwise it outputs 0.

- **Decrypt**. On input a ciphertext $(C_{\text{cred}}, B_{\text{cred}})$ and ask_{sp} , it computes $Y' = C_{\text{cred}} / B_{\text{cred}}^{\text{ask}_{\text{sp}}}$.

Security of Sender Privacy. We give the security theorem of our sender privacy (SP) protocol. The proofs are given in the appendix.

	SP	RP	TP	Aud	Auth	Tx Overhead (bytes)	Sender Running Time	Verifier Running Time
Monero	●	●	●			12704	300ms	300ms
Zcash	●	●	●			576	120s	10ms
Hyperledger Fabric	○	○	○		●	628	10ms	10ms
This paper	●	●	●	●	●	2720	100ms	100ms

Table 2: Comparison of privacy-preserving blockchain schemes, for a standard 2-input-2-output transaction.

	Setup Time	C_{tp} Enc Time	C_{tp} Dec Time	π_{tp} Proof Time	π_{tp} Verify Time
Our Scheme	53.8s	2.8ms	3.0ms	27.1ms	25.6ms
Paillier Encryption	402.6ms	27.1ms	7.4ms		

Table 3: Comparison for transaction privacy for a single output

Theorem 3. *The SP protocol is sound if the q -SDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$ in the random oracle model, where q is the maximum number of Issue_e oracle query. The SP protocol is unforgeable if the DL assumption holds in \mathbb{G}_1 in the random oracle model. The SP protocol is anonymous if the DDH assumption holds in \mathbb{G}_1 in the random oracle model.*

7. Performance Analysis

We analyze our PACchain in terms of throughput and latency, two of the most important metrics for analyzing the performance of a blockchain system. The latency of our PACchain is affected by the running time of the modules. The throughput of our PACchain is affected by both the running time of our three modules, and the size of each transaction.

7.1. Transaction Overhead

In this paper, we consider 128-bit security. The transaction amount is represented by a 64-bit positive integer (the same setting as Bitcoin and Monero). For PACchain’s transaction privacy, 64-bit of transaction amount implies that the range $R = 2^{64}$. We can take $\mathbf{u} = 2^{16} = 65536$, $\ell = 4$.

The public parameters for transaction privacy is about 2MB. The size of the ciphertext is 256 bytes. For each transaction output amount, the size of the range proof π_{enc} is 544 bytes¹⁰. The size of π_{tp} is 64 bytes plus all π_{enc} for all transaction outputs. For recipient privacy, the size of C_{rp} is 64 bytes, π_{rp} is 256 bytes for each recipient. The block randomness $R_{\text{tx},r}$ is 32 bytes. (The 32 bytes of otpk_r replaces the output address and hence it is not viewed as an overhead). For sender privacy, the size of C_{sp} is 64 bytes, π_{rp} is 352 bytes and T is 32 bytes for each sender.

Considering a classical transaction of 2 inputs and 2 outputs, the overhead for privacy-enhancing consortium blockchain is 2720 bytes. We compare our PACHain with other schemes in Table 2:

- For consortium blockchain (e.g., Fabric or Corda), the classical transaction of 2 inputs and 2 outputs includes 2 ECDSA signatures from two inputs (128 bytes) and two X.509 certificates for 2 outputs' ECDSA public keys (about 500 bytes). The overhead is 628 bytes.
- For the public blockchain Monero, even if we consider the minimum ring size for ring signature as 3 (i.e., the real sender is one-out-of-three public keys. Hence the anonymity is very limited.), the total overhead is 12704 bytes for 2 inputs and 2 outputs.
- For Zcash, all the proofs can be combined to a single 288 bytes zk-SNARK proof. The total proof size becomes 576 bytes. However, the time for generating the proof will be much longer (> 120 sec) and it requires a lot of RAM ($> 3\text{GB}$). It causes a long latency in the blockchain system.

7.2. Module Implementation

We implemented our modules in a server with Intel Core i5 3.4GHz, 8GB RAM, running on Linux. Our implementation is by Golang, using BN256 pairing library.

Transaction Privacy. For transaction privacy, the running time for a single output is shown in table 3. We compare our scheme with the additive homomorphic Paillier encryption with the same security level. When comparing with the encryption and decryption part only, our scheme is about 9

¹⁰A 64-bit range proof by the recent Bulletproof [16] is about 800 bytes.

times and 2 times more efficient than the Paillier encryption. For the prover side, the complete transaction privacy is almost as efficient as a single Paillier encryption. Comparatively, our scheme takes a longer time for Setup, mainly for the generation of system parameters for the range proof.

Recipient Privacy. For recipient privacy for a single output, the Setup time is 4.6ms, the CertIssue time is 1.4ms, the Spend Time is 11.2ms and the Verify time is 10.6ms.

Sender Privacy. For sender privacy for a single input, the Setup time is 7.8ms, the CredIssue time is 1.5ms, the CredSign Time is 15.0ms and the Verify time is 16.3ms.

For a standard 2-input 2-output transaction, the total running time of our scheme (achieving all three properties) is 112ms for the prover and 105ms for the verifier side.

7.3. Testing Transaction Privacy with Hyperledger Fabric

We integrate the transaction privacy protocol in Hyperledger Fabric 1.0, in order to demonstrate our modulus can be consolidated into real world consortium blockchain. There are a few technical obstacles to implement our scheme. The first obstacle is that Fabric does not support optimization code of BN256 pairing written in C language. It results in > 10 times slower exponentiation and pairing computation. We expect future version of Fabric to allow optimization for pairing-based computation.

The second difficulty is to implement the verification logic into the smart contract (chaincode) of Fabric. We built a complete flow of transaction, including the creation of money (deposit), normal transaction, balance query and the destroy of money (withdraw). The chaincode has 2223 lines of codes. The extra codes for server side and client are 575 lines and 1061 lines respectively. The common module has 823 lines. (Comparatively, the core transaction privacy protocol has 2143 lines of codes.)

Transaction Privacy. In our current implementation for a 2-input 2-output transaction in Hyperledger Fabric 1.0, the signing time is 988ms and the verification time is 1.35s. Our implementation shows that other processing time for the transaction packet is negligible when compared to cryptographic operations. We expect that if optimization code of pairing is allowed, the signing and verification time can be about 100ms.

The consensus algorithm is the current bottleneck of most consortium blockchain systems. The PBFT consensus algorithm used in Hyperledger

Fabric 1.0 allows about 2000 transactions per second and has about 1 second of latency. If optimization is allowed in Fabric, our scheme has a running time of 100ms for both the prover and verifier side, for a standard 2-input 2-output transaction. Therefore, our scheme is practical and will not become the bottleneck of the consortium blockchain system.

8. Building PAChain with Hyperledger

We show how to integrate PAChain with the Hyperledger fabric framework. We first describe how the entities in PAChain is mapped to the entities in Hyperledger fabric. Then we give some modifications for combining our cryptographic building blocks. Finally, we demonstrate the system setup followed by the transaction flow in our privacy-preserving auditable consortium blockchain. The complete system architecture described in this section is shown in Figure 2.

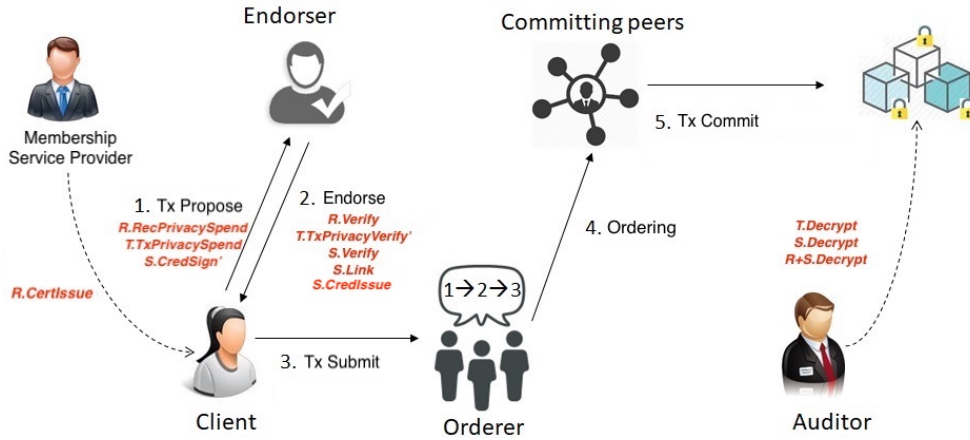


Figure 2: Nodes and Workflow in Hyperledger Fabric

8.1. Role Mapping

In Hyperledger, the sender of a transaction (a client in Hyperledger) will run the spend algorithms in the TP, SP and RP protocols. The endorser will run as the verifier in the TP, SP and RP protocols. In addition, the endorser will also issue anonymous credential to the recipient of the transaction in the SP protocol.

The auditor is an external party running the decrypt in the TP, SP and RP protocols. It is not included in the Hyperledger framework.

Besides the endorsement and consensus services, Hyperledger Fabric also support membership service. It can restrict clients that must be authorized by a *membership service provider* in order to create a transaction. Hence, we map the role of CA to the membership service provider in Hyperledger Fabric.

8.2. Combining Our Building Blocks

We use the transaction privacy protocol **T**, the recipient privacy protocol **R** and the sender privacy protocol **S** to build a secure privacy-enhancing blockchain system. Some non-trivial modifications are needed when combining our building blocks.

8.2.1. Combining Transaction Privacy with Sender Privacy

A small change has to be made for **T.TxPrivacyVerify** and **S.CredSign**. It is because $C_{in,i} = g^{M_{in,i}} h_{tp}^{r_{in,i}}$ and $B_{in,i} = g^{r_{in,i}}$ used for verification in **T.TxPrivacyVerify**, while it should be kept private for sender privacy. If we want to achieve both transaction privacy and sender privacy, we require that they use the same hash output for the 3-move zero-knowledge proof. Then, we have an extra output in each **S.CredSign**:

$$R_{C,i} = g^{r_{m,i}} h_{tp}^{r_{r,i}}, \quad R_{B,i} = g^{r_{r,i}}.$$

For **T.TxPrivacyVerify**, the verification equation involving $(C_{in,i}, B_{in,i})$ is changed to:

$$R'_{tp} = h_{tp}^{z_{tp}} \cdot \prod_{i=1}^n \frac{g^{z_{m,i}} h_{tp}^{z_{r,i}}}{R_{C,i}} \cdot \left(\prod_{j=1}^{n'} C_{out,j} \right)^{-c'}, \quad \tilde{R}'_{tp} = g^{z_{tp}} \cdot \prod_{i=1}^n \frac{g^{z_{r,i}}}{R_{B,i}} \cdot \left(\prod_{j=1}^{n'} B_{out,j} \right)^{-c'}.$$

It is easy to show the soundness and the zero-knowledge property for the above proof of $(C_{in,i}, B_{in,i})$. We omit it for simplicity. Denote the modified algorithms as **S.CredSign'** and **T.TxPrivacyVerify'**.

8.2.2. Combining Recipient Privacy with Sender Privacy

The auditability for sender privacy **S** allows the decryption of the sender's public key in the UTXO. However, recipient privacy **R** requires that this public key is a one-time public key. Therefore, the auditability for sender privacy is indirectly achieved: auditor runs **S.Decrypt** and gets a one-time public key. From this one-time public key, the auditor retrieves the UTXO, and runs **R.Decrypt** to obtain the long-term public key of the sender.

8.3. System Setup and User Registration

This section outlines the system setup and user registration of PACChain. in the Hyperledger Fabric framework [22]. The modified parts of our scheme are *italic* and are marked by red colour. It shows how our modulus can be consolidated into a consortium blockchain.

System Setup. *On input a security parameter 1^λ , it runs $\text{param}_T \leftarrow \text{T.Setup}(1^\lambda)$, $\text{param}_R \leftarrow \text{R.Setup}(1^\lambda)$ and $\text{param}_S \leftarrow \text{S.Setup}(1^\lambda)$. Suppose the system parameter $\text{param} = (\text{param}_T, \text{param}_S, \text{param}_R)$ is known to all users in the system.*

Endorser Setup. The endorser generates a pair of signing and verification ssk, svk . *The endorser obtains a key pair for sender privacy by running $(\text{esk}, \text{epk}) \leftarrow \text{S.EndorserKeyGen}()$.*

CA Setup. The CA obtains a key pair by running $(\text{cask}, \text{capk}) \leftarrow \text{R.CAKeyGen}()$.

User Registration. The user generates his long-term key pair *by running $(\text{usk}, \text{upk}) \leftarrow \text{R.UserKeyGen}()$. The user runs $\text{R.CertIssue}(\text{capk}, \text{upk})$ with the CA and the algorithm outputs cert .* If the user is authorized, the CA returns a certificate cert to the user.

8.4. Transaction Flow

We outline the modified transactional mechanics that take place during a standard transaction. Our protocols are mainly added to the client side (TxPropose) and endorser side (Endorse).

Tx Propose. Consider the case that Alice want to sends $\$M_{\text{out},j}$ to user $\text{upk}_{\text{out},j}$ for $j \in [1, n']$. Suppose that user $\text{upk}_{\text{out},j}$ is registered to the CA with certificate $\text{cert}_{\text{out},j}$ and it is known to Alice. Denote Alice's long-term secret key as usk_A .

Assume that Alice's money is obtained from n different transactions of amount $M_{\text{in},i}$ for $i \in [1, n]$, where $\sum_{i \in [1, n]} M_{\text{in},i} = \sum_{j \in [1, n']} M_{\text{out},j}$. *Then for each input amount $M_{\text{in},i}$, Alice should have the corresponding one time public key $\text{otpk}_{A,i}$, commitment $C_{\text{in},i}$, commitment randomness $r_{\text{in},i}$, credential $\text{cred}_{\text{in},i}$ (issued by endorser E_i) and block randomness $R_{\text{in},i}$.* Alice prepares a transaction message tx which includes the following fields:

1. **clientID.** *It includes the endorsers' IDs E_i for $i \in [1, n]$.* ¹¹

¹¹In the Hyperledger Fabric, this field includes the sender ID instead. Hence, it does not provide sender anonymity.

2. **chaincodeID**. It refers to the chaincode to which the transaction pertains.
3. **timestamp**. It is a monotonically increasing integer maintained by the client.
4. **txPayload**. It is the payload containing the submitted transaction, composed of $\langle \text{source}, \text{metadata}, \text{policies} \rangle$. **source** denotes the source code of the chaincode and **policies** includes the endorsement policy. **metadata** includes the following:
 - (a) **RecipientPK**. For each recipient $\text{upk}_{\text{out},j}$, Alice runs the zero-knowledge protocol for recipient privacy, without using the part for **otsk**: $(\cdot, \text{otpk}_{\text{out},j}, R_{\text{tx},\text{out}}, C_{\text{rp},j}, \pi_{\text{rp},j}) \leftarrow \text{R.RecPrivacySpend}(\text{capk}, \cdot, \cdot, \cdot, \text{upk}_{\text{out},j}, \text{cert}_{\text{out},j})$. It is because the possession of **otsk** will later be shown by **S.CredSign**. Note that for all $j \in [1, n']$, Alice generates the same $R_{\text{tx},\text{out}}$ output during the computation. The **Recipient PK** field includes $(\text{otpk}_{\text{out},j}, C_{\text{rp},j}, \pi_{\text{rp},j})$ for all $j \in [1, n']$.
 - (b) **Output**. Alice generates the output field $(\{C_{\text{out},j}\}_{j \in [1, n']}, \pi_{\text{tp}})$, where: $(\{C_{\text{out},j}, r_{\text{out},j}\}_{j \in [1, n']}, \pi_{\text{tp}}) \leftarrow \text{T.TxPrivacySpend}(\{M_{\text{out},j}\}_{j \in [1, n']}, \{C_{\text{in},i}, M_{\text{in},i}, r_{\text{in},i}\}_{i \in [1, n]})$.
 - (c) **Info**. Alice puts the block randomness $R_{\text{tx},\text{out}}$ in the information field¹².
5. **clientSig**. Alice obtains the one-time secret keys of her n inputs by $x'_{\text{in},i} \leftarrow \text{R.OneTimeSkGen}(\text{otpk}_{A,i}, R_{\text{in},i}, \text{usk}_A)$ for $i \in [1, n]$. Alice runs $(C_{\text{sp},i}, T_i, \pi_{\text{sp},i}) \leftarrow \text{S.CredSign}'(x'_{\text{in},i}, \text{otpk}_{A,i}, \text{cred}_{\text{in},i}, M_{\text{in},i}, r_{\text{in},i})$ for all $i \in [1, n]$: Finally, **clientSig** includes $(C_{\text{sp},i}, T_i, \pi_{\text{sp},i})$ for all $i \in [1, n]$.

Alice submits a proposal message $\langle \text{PROPOSE}, \text{tx}, [\text{anchor}] \rangle$ to the endorsing peers, where **anchor** is an optional field containing the version numbers for some keys (of the key-value pairs in chaincode). The transaction ID **tid** can be computed as $\text{Hash}(\text{tx})$.

Endorse. On reception of a message $\langle \text{PROPOSE}, \text{tx}, [\text{anchor}] \rangle$ from a client, the endorser verifies that:

1. the transaction proposal **tx** is well-formed,

¹²Optionally, Alice can encrypt her long-term public key with $(M_{\text{out},j}, r_{\text{out},j})$ to user $\text{upk}_{\text{out},j}$. If this information is not sent by blockchain, it can be sent by private channel between Alice and the recipient.

2. the version number in `anchor` (if any) matches the current read version number of corresponding keys.
3. the signature `clientSig` in `tx` is valid. *Instead of checking against sender's public key `upk` in Hyperledger Fabric, our proposal runs for all $i \in [1, n]$:*

$$1/0 \leftarrow \text{S.Verify}(\text{epk}_i, C_{\text{sp}_i}, T_i, \pi_{\text{sp}_i}),$$

where epk_i is the public key of the endorser specified in the field `clientID` in `tx`.

4. *the proposal does not contain double spending. It runs $\text{S.Link}(T^*, T')$ for all $(\cdot, T^*, \cdot) \in \text{clientSig}$ and for all T' of the past transactions. If any of them returns 1, it implies double spending.*
5. *the recipient is authorized. For all $j \in [1, n']$, it runs: $1/0 \leftarrow \text{R.Verify}(\text{capk}, \cdot, \cdot, \text{otpk}_{\text{out},j}, R_{\text{tx},\text{out}}, C_{\text{rp},j}, \pi_{\text{rp},j})$. We ignore the checking of `otsk` in R.Verify since it is checked in S.Verify .*
6. *the transaction amount is correct. It runs:*

$$1/0 \leftarrow \text{T.TxPrivacyVerify}'(\{\pi_{\text{sp},i}\}_{i \in [1,n]}, \{C_{\text{out},j}\}_{j \in [1,n']}, \pi_{\text{tp}}).$$

After the checking, the endorser tentatively executes a transaction (`txPayload`), by invoking the chaincode with ID `chaincodeID` and the copy of the state that the endorser locally holds. As a result of the execution, the endorser computes read version dependencies (`readset`) and state updates (`writeset`). It returns the message $\langle \text{TRANSACTION-ENDORSED}, \text{tid}, \text{tran-proposal}, \text{epSig} \rangle$ to Alice, where:

- `tran-proposal` := (`epID`, `tid`, `chaincodeID`, `txContentBlob`, `readset`, `writeset`), with `epID` as the endorser ID, and `txContentBlob` as transaction specific information. *In particular, the endorser runs $\text{cred}_{\text{out},j} \leftarrow \text{S.CredIssue}(\text{epk}, \text{otpk}_{\text{out},j}, C_{\text{out},j})$ for all $j \in [1, n']$, with private input `esk`.¹³ The endorser puts $\{\text{cred}_{\text{out},j}\}_{j \in [1,n']}$ into `txContentBlob`.*
- `epSig` is the endorser's signature on `tran-proposal` by using its signing key `ssk`.

¹³ π_{ci} is not needed here, since the transaction includes π_{rp} , which implies the existence of π_{ca} in R.CertIssue . The proof π_{ca} is the same as π_{ci} .

If in any case the endorser refuses to endorse the transaction, it may send a message $\langle \text{TRANSACTION-INVALID}, \text{tid}, \text{REJECTED} \rangle$ to Alice.

Tx Submit. Alice waits until it receives *enough* messages and signatures on $\langle \text{TRANSACTION-ENDORSED}, \text{tid}, *, * \rangle$ statements to conclude that the transaction proposal is endorsed. The collection of signed `TRANSACTION-ENDORSED` messages are denoted as `endorsement`. Alice invokes the ordering service with `endorsement`.

Ordering. The ordering peers run a consensus algorithm with the `endorsement` they received. Once an agreement is reached, $\langle \text{DELIVER}, \text{seqno}, \text{prevhash}, \text{endorsement} \rangle$ is broadcast to all peers.

Tx Commit. Upon receiving $\langle \text{DELIVER}, \text{seqno}, \text{prevhash}, \text{endorsement} \rangle$, a peer checks: (1) `endorsement` is valid according to the policy of the chaincode having `ID = endorsement.tran-proposal.chaincodeID`; (2) the dependencies in `endorsement.tran-proposal.readset` have not been violated meanwhile. If all these checks pass, the transaction is deemed valid or committed. The peer applies `endorsement.tran-proposal.writeset` to blockchain state.

9. Conclusion

In this paper, we propose efficient solution for privacy, auditability and authentication in consortium blockchain. We give module solutions for them, so that they can be added to blockchain according to actual business need. We implemented our schemes and they are more efficient than the existing solutions in public blockchain. We showed that it can be integrated with Hyperledger fabric framework.

- [1] T. H. Yuen, Pachain: Private, authenticated and auditable consortium blockchain, in: Y. Mu, R. H. Deng, X. Huang (Eds.), CANS 2019, Vol. 11829 of LNCS, Springer, 2019, pp. 214–234.
- [2] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, <https://bitcoin.org/bitcoin.pdf> (2009).
- [3] C. Garman, M. Green, I. Miers, Accountable privacy for decentralized anonymous payments, in: J. Grossklags, B. Preneel (Eds.), FC 2016, Vol. 9603 of LNCS, Springer, 2016, pp. 81–98.
- [4] W. Li, A. Sforzin, S. Fedorov, G. O. Karame, Towards scalable and private industrial blockchains, in: BCC '17, ACM, 2017, pp. 9–14.

- [5] K. Wüst, K. Kostiaainen, V. Capkun, S. Capkun, Prcash: Centrally-issued digital currency with privacy and regulation, Cryptology ePrint Archive, Report 2018/412, <https://eprint.iacr.org/2018/412>. In FC 2019. (2018).
- [6] S. Noether, Ring signature confidential transactions for monero, Cryptology ePrint Archive, Report 2015/1098, <http://eprint.iacr.org/> (2015).
- [7] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, M. Virza, Zerocash: Decentralized anonymous payments from bitcoin, in: IEEE SP 2014, IEEE Computer Society, 2014, pp. 459–474.
- [8] M. U. Hassan, M. H. Rehmani, J. Chen, Privacy preservation in blockchain based iot systems: Integration issues, prospects, challenges, and future research directions, Future Generation Comp. Syst. 97 (2019) 512–529. doi:10.1016/j.future.2019.02.060.
URL <https://doi.org/10.1016/j.future.2019.02.060>
- [9] A. A. Omar, M. Z. A. Bhuiyan, A. Basu, S. Kiyomoto, M. S. Rahman, Privacy-friendly platform for healthcare data in cloud based on blockchain environment, Future Generation Comp. Syst. 95 (2019) 511–521. doi:10.1016/j.future.2018.12.044.
URL <https://doi.org/10.1016/j.future.2018.12.044>
- [10] W. Jiang, H. Li, G. Xu, M. Wen, G. Dong, X. Lin, PTAS: privacy-preserving thin-client authentication scheme in blockchain-based PKI, Future Generation Comp. Syst. 96 (2019) 185–195. doi:10.1016/j.future.2019.01.026.
URL <https://doi.org/10.1016/j.future.2019.01.026>
- [11] J. Camenisch, S. Mödersheim, D. Sommer, A formal model of identity mixer, in: S. Kowalewski, M. Roveri (Eds.), FMICS 2010, Vol. 6371 of LNCS, Springer, 2010, pp. 198–214.
- [12] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in: J. Stern (Ed.), EUROCRYPT '99, Vol. 1592 of LNCS, Springer, 1999, pp. 223–238.

- [13] F. Boudot, Efficient proofs that a committed number lies in an interval, in: B. Preneel (Ed.), EUROCRYPT 2000, Vol. 1807 of LNCS, Springer, 2000, pp. 431–444.
- [14] M. Bellare, S. Goldwasser, Verifiable partial key escrow, in: R. Gravenman, P. A. Janson, C. Neuman, L. Gong (Eds.), CCS '97., ACM, 1997, pp. 78–91.
- [15] J. Camenisch, R. Chaabouni, A. Shelat, Efficient protocols for set membership and range proofs, in: J. Pieprzyk (Ed.), ASIACRYPT 2008, Vol. 5350 of LNCS, Springer, 2008, pp. 234–252.
- [16] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, G. Maxwell, Bulletproofs: Short proofs for confidential transactions and more, in: IEEE SP 2018, IEEE, 2018, pp. 315–334.
- [17] T. Ruffing, P. Moreno-Sanchez, A. Kate, Coinshuffle: Practical decentralized coin mixing for bitcoin, in: M. Kutylowski, J. Vaidya (Eds.), ESORICS 2014, Vol. 8713 of LNCS, Springer, 2014, pp. 345–364.
- [18] M. H. Au, W. Susilo, Y. Mu, Constant-size dynamic k -taa, in: R. D. Prisco, M. Yung (Eds.), SCN 2006, Vol. 4116 of LNCS, Springer, 2006, pp. 111–125.
- [19] S. Sun, M. H. Au, J. K. Liu, T. H. Yuen, Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero, in: S. N. Foley, D. Gollmann, E. Sneekenes (Eds.), ESORICS 2017, Vol. 10493 of LNCS, Springer, 2017, pp. 456–474.
- [20] D. Boneh, X. Boyen, H. Shacham, Short group signatures, in: M. K. Franklin (Ed.), CRYPTO 2004, Vol. 3152 of LNCS, Springer, 2004, pp. 41–55.
- [21] D. Boneh, X. Boyen, Short signatures without random oracles, in: C. Cachin, J. Camenisch (Eds.), EUROCRYPT 2004, Vol. 3027 of LNCS, Springer, 2004, pp. 56–73.
- [22] Hyperledger Fabric, Architecture explained, <http://hyperledger-fabric.readthedocs.io/en/latest/arch-deep-dive.html> (2017).

Appendix A. Security Proofs

Appendix A.1. Security Proofs of Transaction Privacy

Lemma 1. *The zero-knowledge proof π_{enc} is sound if the \mathbf{u} -SDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$ in the random oracle model.*

Proof. Suppose the simulator is given \mathbf{u} Boneh-Boyen signatures on messages $0, \dots, \mathbf{u} - 1$ for a public key $\hat{\mathfrak{Q}}$ and he wants to break the unforgeability of Boneh-Boyen signature. The simulator uses these signatures as \mathfrak{A}_i for $i \in [0, \mathbf{u} - 1]$. The adversary outputs the ciphertext $\{B_j, C_j\}_{j \in [0, \ell - 1]}$ and proof $\pi_{\text{enc}} = (\{V_j, z_{\mu_j}, z_{v_j}, z_{r_j}\}_{j \in [0, \ell - 1]}, \tilde{c})$. The simulator rewinds \tilde{c} and receives another proof $\pi'_{\text{enc}} = (\{V_j, z'_{\mu_j}, z'_{v_j}, z'_{r_j}\}_{j \in [0, \ell - 1]}, \tilde{c}')$. Then we have:

$$\hat{e}(V_j, \hat{\mathfrak{Q}}^{\tilde{c} - \tilde{c}'} \hat{g}^{z'_{\mu_j} - z_{\mu_j}}) = \hat{e}(g, \hat{g})^{z'_{v_j} - z_{v_j}}, C_j^{\tilde{c} - \tilde{c}'} = g_0^{z'_{\mu_j} - z_{\mu_j}} h_{\text{tp}}^{z'_{r_j} - z_{r_j}}, B_j^{\tilde{c} - \tilde{c}'} = g^{z'_{r_j} - z_{r_j}}.$$

By setting $\mu_j^* = \frac{z'_{\mu_j} - z_{\mu_j}}{\tilde{c} - \tilde{c}'}$, $v_j^* = \frac{z'_{v_j} - z_{v_j}}{\tilde{c} - \tilde{c}'}$, $r_j^* = \frac{z'_{r_j} - z_{r_j}}{\tilde{c} - \tilde{c}'}$, we have:

$$\hat{e}(V_j, \hat{\mathfrak{Q}} \cdot \hat{g}^{\mu_j^*})^{\frac{1}{v_j^*}} = \hat{e}(g, \hat{g}), \quad C_j = g_0^{\mu_j^*} h_{\text{tp}}^{r_j^*}, \quad B_j = g^{r_j^*}.$$

The last two equations showed that (C_j, B_j) is a valid ElGamal ciphertext of μ_j^* . With non-negligible probability, μ_j^* is not in $[0, \mathbf{u} - 1]$. Then the simulator can return V_j^{1/v_j^*} as the forged Boneh-Boyen signature on message μ_j^* . If the SDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$, the proof π_{enc} is sound. \square

Lemma 2. *The zero-knowledge proof π_{enc} is zero-knowledge in the random oracle model.*

Proof. The simulator picks some random $\{V_j, B_j, C_j, z_{\mu_j}, z_{v_j}, z_{r_j}\}_{j \in [0, \ell - 1]}, \tilde{c}$ for the corresponding domain and computes $a_j = \hat{e}(V_j, \hat{\mathfrak{Q}}^{\tilde{c}} \hat{g}^{-z_{\mu_j}}) \cdot \hat{e}(g, \hat{g})^{z_{v_j}}$, $D_j = B_j^{\tilde{c}} g^{z_{r_j}}$, $E_j = C_j^{\tilde{c}} g_0^{z_{\mu_j}} h_{\text{tp}}^{z_{r_j}}$. The simulator sets \tilde{c} as $H(\text{param}, \{V_j, a_j, B_j, C_j, D_j, E_j\}_{j \in [0, \ell - 1]})$ in the random oracle model. Hence, the proof π_{enc} is zero knowledge about the Tx amount. \square

Proof of Soundness and Privacy for TP. The property of soundness follows from Lemma 1. The privacy of the transaction amount is straightforward following the IND-CPA security of the underlying ElGamal encryption (for C_{enc}) and Lemma 2 for the zero-knowledge property of π_{enc} .

Proof of Balance for TP.

Proof. Suppose the simulator is given the DL problem instance (g', y') and sets $g_0 = y', h_{\text{tp}} = g'$. The simulator picks a random $\text{ask}_{\text{tp}} \in \mathbb{Z}_p$ and computes $g = h_{\text{tp}}^{1/\text{ask}_{\text{tp}}}$. The simulator honestly runs the rest of **Setup** and generates **param**. It gives **param** and ask_{tp} to the adversary, and answers all **Spend** oracle queries.

In the challenge phase, the adversary outputs $C_{\text{in},i}$ for $i \in [1, n]$, $C_{\text{out},j}$ for $j \in [1, n']$ and a proof $\pi_{\text{tp}} = (z_{\text{tp}}, \tilde{c}', \{\pi_{\text{enc},j}\}_{j \in [1, n']})$. The simulator rewinds \tilde{c}' to \tilde{c}'_2 and obtains $z_{\text{tp},2}$. Therefore we have:

$$\begin{aligned} h_{\text{tp}}^{z_{\text{tp}}} \left(\prod_{i=1}^n C'_{\text{in},i} / \prod_{j=1}^{n'} C'_{\text{out},j} \right)^{\tilde{c}'} &= h_{\text{tp}}^{z_{\text{tp},2}} \left(\prod_{i=1}^n C'_{\text{in},i} / \prod_{j=1}^{n'} C'_{\text{out},j} \right)^{\tilde{c}'_2}, \\ g^{z_{\text{tp}}} \left(\prod_{i=1}^n B'_{\text{in},i} / \prod_{j=1}^{n'} B'_{\text{out},j} \right)^{\tilde{c}'} &= g^{z_{\text{tp},2}} \left(\prod_{i=1}^n B'_{\text{in},i} / \prod_{j=1}^{n'} B'_{\text{out},j} \right)^{\tilde{c}'_2}, \end{aligned}$$

Denote $x_{\text{tp}}^* = \frac{z_{\text{tp}} - z_{\text{tp},2}}{\tilde{c}'_2 - \tilde{c}'}$. Then we have $\prod_{i=1}^n C'_{\text{in},i} / \prod_{j=1}^{n'} C'_{\text{out},j} = h_{\text{tp}}^{x_{\text{tp}}^*}$ and $\prod_{i=1}^n B'_{\text{in},i} / \prod_{j=1}^{n'} B'_{\text{out},j} = g^{x_{\text{tp}}^*}$.

By the soundness of $\{\pi_{\text{enc},j}\}_{j \in [1, n']}$, $B'_{\text{out},j} = g^{r_{\text{out},j}}$, $C'_{\text{out},j} = g_0^{M_{\text{out},j}} h_{\text{tp}}^{r_{\text{out},j}}$ for all $j \in [1, n']$, and the value of $M_{\text{out},j}, r_{\text{out},j}$ can be extracted. Also from the **Spend** oracle queries, we have $C'_{\text{in},i} = g_0^{M_{\text{in},i}} h_{\text{tp}}^{r_{\text{in},i}}$, $B'_{\text{in},i} = g^{r_{\text{in},i}}$ for all $i \in [1, n]$. If $\sum_{i=1}^n M_{\text{in},i} \neq \sum_{j=1}^{n'} M_{\text{out},j}$, we have

$$g_0^{\sum_{i=1}^n M_{\text{in},i} - \sum_{j=1}^{n'} M_{\text{out},j}} h_{\text{tp}}^{\sum_{i=1}^n r_{\text{in},i} - \sum_{j=1}^{n'} r_{\text{out},j}} = h_{\text{tp}}^{x_{\text{tp}}^*}.$$

Then the simulator extract $\log_{g'} y' = \log_{h_{\text{tp}}} g_0 = \frac{x_{\text{tp}}^* - \sum_{i=1}^n r_{\text{in},i} - \sum_{j=1}^{n'} r_{\text{out},j}}{\sum_{i=1}^n M_{\text{in},i} - \sum_{j=1}^{n'} M_{\text{out},j}}$ as the answer to the DL problem. \square

Appendix A.2. Security Proofs for Recipient Privacy

Proof of Soundness for RP.

Proof. The simulator is given the q -SDH problem instance. It randomly picks $w_1, \dots, w_{q-1}, t_1, t_2, f', w' \in \mathbb{Z}_p$ and sets $h_2 = g_1^{\prod_{i=1}^{q-1} (\beta + w_i)}$, $g_3 = h_2^{\frac{(w' + \beta)f' - 1}{t_2}}$, $g_2 = g_3^{t_1}$, $\hat{W}_2 = \hat{g}_2^\beta$. The rest of **param** are generated as in **Setup**.

The oracles are simulated as follows:

- **KeyGen()**: The simulator runs $(\text{usk}, \text{upk}) \leftarrow \text{UserKeyGen}()$, stores (usk, upk) in a list \mathcal{L} (which is initially empty) and outputs upk .
- **Corrupt(upk)**: On input upk , it searches $(\text{usk}, \text{upk}) \in \mathcal{L}$ and returns usk . It returns \perp if no such key is found.
- **Issue(upk)**: On i -th distinct input $\text{upk}_i = (Y_{i,1}, Y_{i,2})$, the simulator uses the extractor of the ZK proof π_{ca} to obtain $x_{i,1} = \log_{g_2} Y_{i,1}$. Without loss of generality, assume the simulator assigns w_i to the i -th query for $i \in [1, q-1]$ and sets $w_q = w'$. For the query with $i \in [1, q-1]$, it picks a random $s_i \in \mathbb{Z}_p$ and uses w_i to compute:

$$F_i = (h_2 \cdot Y_{i,1} \cdot g_3^{s_i})^{\frac{1}{\beta+w_i}} = g_1^{(1+(\iota_1+s_i x_{i,1}) \cdot \frac{(w'+\beta)f'-1}{\iota_2}) \prod_{j=1, j \neq i}^{q-1} (\beta+w_j)}.$$

The oracle returns the certificate $\text{cert}_i = (F_i, w_i, s_i)$.

For the query with $i = q$, it sets $s' = \frac{\iota_2 - x_{q,1}}{\iota_1}$ and $F' = h_2^{f'}$. The oracle returns the certificate $\text{cert}_q = (F', w', s')$.

- **Spend($\text{upk}_s, \text{otpk}_s, R_{tx,s}, \text{upk}_r, \text{cert}_r$)**: For each **Spend** oracle query to any public key upk_r with certificate cert_r from some public key upk_s corresponding to $(\text{otpk}_s, R_{tx,s})$, it runs as follows: (1) The simulator runs the simulator of the zero-knowledge proof π_{otsk} . (2) It generates $\text{otpk}_r, R_{tx,r}$ and r_{tx} as in the **OneTimePkgGen** protocol. (3) It generates $C_{\text{rp}} = (C_{\text{cert}}, B_{\text{cert}})$ as in the protocol. (4) It picks some $\pi_{\text{rp}} = (c, \Theta, z_\omega, z_\tau, z_\rho, z_c, z_\sigma, z_s)$ in the corresponding domains and calculates $R_{\text{cert},1}, R_{\text{cert},2}, R_{\text{cert},3}, R_{\text{cert},4}$ as in **ZKReconstruct**. It sets the random oracle $c' = H(\text{CertAuth.mpk}, C_{\text{rp}}, \Theta, R_{\text{cert},1}, R_{\text{cert},2}, R_{\text{cert},3}, R_{\text{cert},4})$. It outputs $\pi_{\text{otsk}}, \text{otpk}_s, \text{otpk}_r, R_{tx,r}, B_{\text{cert}}, C_{\text{cert}}$ and π_{rp} .

In the challenge phase, the adversary outputs two valid transactions where otpk^* is the recipient of the first Tx and otpk^* is the sender of the second Tx. In the second Tx, the adversary spends the money of otpk^* by using π_{otsk}^* . The simulator can decrypt C_{rp}^* of the first Tx and obtains Y_1^* . If the adversary wins the game, $\text{upk}^* = (Y_1^*, \cdot)$ should not be input to both the **Issue** oracle, and $(\text{upk}^*, \text{otpk}^*, R_{tx}^*, \cdot, \cdot)$ should not be queried to the **Spend** oracle.

Denote the first Tx output is $(\pi_{\text{otsk}}, \text{otpk}_s, \text{otpk}^*, R_{tx}^*, C_{\text{rp}}^*, \pi_{\text{rp}}^*)$ and denote $C_{\text{rp}}^* = (C_{\text{cert}}, B_{\text{cert}})$, $\pi_{\text{rp}}^* = (c, \Theta, z_\omega, z_\tau, z_\rho, z_c, z_\sigma, z_s)$. The simulator rewinds c

and receives another proof $\pi'_{rp} = (c', \Theta, z'_\omega, z'_\tau, z'_\rho, z'_c, z'_\sigma, z'_s)$. Then we have:

$$\begin{aligned} \hat{e}((h_2 \cdot C_{\text{cert}})^{z_\rho - z'_\rho} g_3^{z'_s - z_s} \Theta^{z'_\omega - z_\omega} h_{\text{rp}}^{z'_\sigma - z_\sigma}, \hat{g}_2) &= \hat{e}(\Theta, \hat{W}_2)^{c' - c}, \\ B_{\text{cert}}^{c - c'} &= g^{z_c - z'_c}, \quad B_{\text{cert}}^{z_\rho - z'_\rho} = g^{z_\sigma - z'_\sigma}, \quad h_{\text{rp}}^{z_c - z'_c} g_2^{z'_\tau - z_\tau} = (C_{\text{cert}} / \text{otpk}^*)^{c' - c}. \end{aligned}$$

By setting $\rho^* = \frac{z'_\rho - z_\rho}{c - c'}$, $w^* = \frac{z'_w - z_w}{c - c'}$, $\sigma^* = \frac{z'_\sigma - z_\sigma}{c - c'}$, $r_{\text{cert}}^* = \frac{z'_c - z_c}{c - c'}$, $h_{\text{tx}}^* = \frac{z'_\tau - z_\tau}{c - c'}$, $s^* = \frac{z'_s - z_s}{c - c'}$, we have:

$$\hat{e}((h_2 \cdot C_{\text{cert}})^{\rho^*} g_3^{s^*} \Theta^{-w^*} h_{\text{rp}}^{-\sigma^*}, \hat{g}_2) = \hat{e}(\Theta, \hat{W}_2), \quad (\text{A.1})$$

$$B_{\text{cert}} = g^{r_{\text{cert}}^*}, \quad (\text{A.2})$$

$$B_{\text{cert}}^{\rho^*} = g^{\sigma^*}, \quad (\text{A.3})$$

$$C_{\text{cert}} = \text{otpk}^* \cdot h_{\text{rp}}^{r_{\text{cert}}^*} g_2^{-h_{\text{tx}}^*}. \quad (\text{A.4})$$

From Eq. A.2 and A.3, we have $\sigma^* = \rho^* r_{\text{cert}}^*$. Putting it and Eq. A.4 back to Eq. A.1, we have:

$$\begin{aligned} \hat{e}((h_2 \cdot C_{\text{cert}})^{\rho^*} g_3^{s^*} \Theta^{-w^*} h_{\text{rp}}^{-\rho^* r_{\text{cert}}^*}, \hat{g}_2) &= \hat{e}(\Theta, \hat{W}_2), \\ \hat{e}(h_2 \cdot \text{otpk}^* \cdot g_2^{-h_{\text{tx}}^*} \cdot g_3^{s^*}, \hat{g}_2) &= \hat{e}(\Theta^{\frac{1}{\rho^*}}, \hat{g}_2^{w^*} \hat{W}_2). \end{aligned}$$

If the adversary wins the game, it means that he can spend the money of otpk^* by issuing a zero-knowledge proof π_{otsk}^* of $\text{otsk}^* = \log_{g_2} \text{otpk}^*$. Since $(\text{upk}^*, \text{otpk}^*, R_{\text{tx}}^*, \cdot, \cdot)$ was not be queried to the **Spend** oracle, π_{otsk}^* was not from the output of the simulated oracle query. By using the extractor of π_{otsk}^* , the simulator can extract otsk^* . Then:

$$\begin{aligned} \hat{e}(h_2 \cdot g_2^{\text{otsk}^* - h_{\text{tx}}^*} \cdot g_3^{s^*}, \hat{g}_2) &= \hat{e}(\Theta^{\frac{1}{\rho^*}}, \hat{g}_2^{w^*} \hat{W}_2), \\ \hat{e}(g_1^{(1 + (\iota_1(\text{otsk}^* - h_{\text{tx}}^*) + s^*) \frac{(\beta + w') f' - 1}{\iota_2}) \prod_{i=1}^{q-1} (\beta + w_i)}, \hat{g}_2) &= \hat{e}(\Theta^{\frac{1}{\rho^*}}, \hat{g}_2^{\beta + w^*}). \end{aligned}$$

Denote the polynomial $\frac{(1 + (\iota_1(\text{otsk}^* - h_{\text{tx}}^*) + s^*) \frac{(\beta + w') f' - 1}{\iota_2}) \prod_{i=1}^{q-1} (\beta + w_i)}{\beta + w^*}$ as $\sum_{i=0}^{q-2} Z_i \beta^i + \frac{Z_{-1}}{\beta + w^*}$ for some coefficients $Z_{-1}, Z_0, \dots, Z_{q-2} \in \mathbb{Z}_p$ which can be computed by the simulator. Then we have:

$$\begin{aligned} \hat{e}(g_1^{\sum_{i=0}^{q-2} Z_i \beta^i + \frac{Z_{-1}}{\beta + w^*}}, \hat{g}_2^{\beta + w^*}) &= \hat{e}(\Theta^{\frac{1}{\rho^*}}, \hat{g}_2^{\beta + w^*}), \\ \hat{e}(\Theta^{\frac{1}{\rho^*}} g_1^{-\sum_{i=0}^{q-2} Z_i \beta^i}, \hat{g}_2^{\beta + w^*}) &= \hat{e}(g_1^{Z_{-1}}, \hat{g}_2). \end{aligned}$$

If $Z_{-1} \neq 0$, the simulator can compute $A^* = (\Theta_{\rho^*}^{\frac{1}{\rho^*}} g_1^{-\sum_{i=0}^{q-2} Z_i \beta^i})^{\frac{1}{Z_{-1}}}$. It returns (A^*, w^*) as the solution to the q -SDH problem.

If $Z_{-1} = 0$, it implies that $w^* = w_i$ for some i -th Issue oracle output in the past. With probability $1 - 1/q$, $w^* \neq w'$. Denote such oracle input is $\text{upk}_i = (Y_{i,1}, \cdot)$ and output is (F_i, w^*, s_i) . Then we have $\hat{e}(h_2 \cdot g_2^{x_{i,1}} \cdot g_3^{s_i}, \hat{g}_2) = \hat{e}(F_i, \hat{g}_2^{w^*} \hat{W}_2)$. Recall that $\hat{e}(h_2 \cdot g_2^{\text{otsk}^* - h_{tx}^*} \cdot g_3^{s^*}, \hat{g}_2) = \hat{e}(\Theta_{\rho^*}^{\frac{1}{\rho^*}}, \hat{g}_2^{w^*} \hat{W}_2)$. If $\Theta_{\rho^*}^{\frac{1}{\rho^*}} = F_i$, then we have $s^* = s_i$ and $\text{otsk}^* - h_{tx}^* = x_{i,1}$ if the DL assumption holds between g_2 and g_3 . Recall that the decryption of $(C_{\text{cert}}, B_{\text{cert}})$ gives Y_1^* if the adversary wins the game. By Eq. A.2 and A.4, we have $Y_1^* = \text{otpk}^* g_2^{-h_{tx}^*}$. By the soundness of zero-knowledge proof $\pi_{\text{otsk}^*}^*$, we have $\text{otsk}^* = \log_{g_2} \text{otpk}^*$. It implies that $Y_1^* = g_2^{x_{i,1}}$, which contradicts that Y_1^* is never queried to the Issue oracle.

If $Z_{-1} = 0$, $w^* = w_i$ and $\Theta_{\rho^*}^{\frac{1}{\rho^*}} \neq F_i$ for some i , we show that the simulator can also solve the q -SDH. With probability $1/q$, $w^* = w'$. Then denote $F^* = \Theta_{\rho^*}^{\frac{1}{\rho^*}}$ and $y^* = \iota_1(\text{otsk}^* - h_{tx}^*) + s^*$:

$$\begin{aligned} F^{*\beta+w^*} &= h_2 g_2^{\text{otsk}^* - h_{tx}^*} \cdot g_3^{s^*} = h_2 g_3^{\iota_1(\text{otsk}^* - h_{tx}^*) + s^*} = h_2 g_3^{y^*} = h_2^{1+y^* \cdot \frac{(w'+\beta)f'-1}{\iota_2}}, \\ F^* &= h_2^{\frac{\iota_2 - y^* + y^* f'(\beta+w')}{\iota_2(\beta+w')}} = h_2^{\frac{\iota_2 - y^*}{\iota_2(\beta+w')} + \frac{y^* f'}{\iota_2}}, \\ (F^* h_2^{\frac{-y^* f'}{\iota_2}})^{\frac{\iota_2}{\iota_2 - y^*}} &= h_2^{\frac{1}{\beta+w'}} = g_1^{\frac{\prod_{i=1}^{q-1} (\beta+w_i)}{\beta+w'}} \end{aligned}$$

Denote the polynomial $\frac{\prod_{i=1}^{q-1} (\beta+w_i)}{\beta+w'} = \sum_{i=0}^{q-2} Z'_i \beta^i + \frac{Z'_{-1}}{\beta+w'}$ for some coefficients $Z'_{-1}, Z'_0, \dots, Z'_{q-2} \in \mathbb{Z}_p$ which can be computed by the simulator. Note that $Z'_{-1} \neq 0$ in this case since the simulator sets $w' \neq w_i$ for all i . Then the simulator can compute $A' = [(F^* h_2^{\frac{-y^* f'}{\iota_2}})^{\frac{\iota_2}{\iota_2 - y^*}} g_1^{-\sum_{i=0}^{q-2} Z'_i \beta^i}]^{\frac{1}{Z'_{-1}}}$. It returns (A', w') as the solution to the q -SDH problem. \square

Proof of Anonymity for RP.

Proof. In the output of the challenge phase, the simulator runs `RecPrivateSpend` for some challenge recipient $\text{upk}_r = (Y_1, Y_2)$, where:

- $\pi_{\text{otsk}^*}^*$ is not related to the recipient.
- $C_{\text{rp}}^* = (C_{\text{cert}}, B_{\text{cert}})$ is the ElGamal encryption of Y_1 to the auditor. It does not provide information of Y_r if the DDH assumption holds in \mathbb{G}_1 .

- Observe that $\text{otpk}_r^* = Y_1 g_2^{H(Y_2^{r_{tx,r}})}$ and $R_{tx,r}^* = g_2^{r_{tx,r}}$. Assume H is a secure hash function (the pseudorandomness of output) and the CDH assumption holds in \mathbb{G}_1 (for the computation from the Diffie-Hellman tuple), $\text{otpk}_r^*, R_{tx,r}^*$ does not provide information of upk_r .
- For π_{rp}^* , the simulator picks some $(c, \Theta, z_\omega, z_\tau, z_\rho, z_c, z_\sigma, z_s)$ in the corresponding domains and calculates $R_{\text{cert},1}, R_{\text{cert},2}, R_{\text{cert},3}, R_{\text{cert},4}$ as in **ZKReconstruct**. It sets the random oracle $c' = H(\text{CertAuth.mpk}, C_{rp}^*, \Theta, R_{\text{cert},1}, R_{\text{cert},2}, R_{\text{cert},3}, R_{\text{cert},4})$. Hence, π_{rp}^* does not provide information of upk_r .

Hence, the adversary has no advantage of winning the anonymity game if the DDH assumption holds in \mathbb{G}_1 in the random oracle model. \square

Appendix A.3. Sender Privacy

Proof of Soundness for SP.

Proof. The simulator is given the q -SDH problem instance $(g_0, g_0^\alpha, g_0^{\alpha^2}, \dots, g_0^{\alpha^q}, \hat{g}_2, \hat{g}_2^\alpha)$. It randomly picks $z_1, \dots, z_{q-1}, \iota_1, \iota_2, \iota_3, \iota_4, a', z' \in \mathbb{Z}_p$ and sets $h_s = g_0^{\prod_{i=1}^{q-1} (\alpha + z_i)}$, $g_1 = h_s^{\frac{(z'+\alpha)a'-1}{\iota_2}}$, $g_2 = g_1^{\iota_1}$, $W_{\text{sp}} = \hat{g}_2^\alpha$, $g = g_1^{\iota_3}$, $h_{\text{tp}} = g_1^{\iota_4}$. The rest of param and ask_{sp} are generated as in **Setup**. It sets $\text{epk} = W_{\text{sp}}$. The adversary is given param , ask_{sp} and epk .

The oracles are simulated as follows:

- $\text{Issue}_e(\text{upk})$: On k -th distinct input $\text{upk}_k = (Y_{k,1}, \cdot)$ and a ciphertext C_k , the simulator uses the extractor of the ZK proof π_{ca} to obtain $x_{k,1} = \log_g Y_{k,1}$ and the extractor of the ZK proof π_{tp} to obtain $C_k = g^{m_k} h_{\text{tp}}^{r_{\text{tp},k}}$. Without loss of generality, assume the simulator assigns z_k to the k -th query for $k \in [1, q-1]$ and sets $z_q = z'$. For the query with $k = [1, q-1]$, it picks a random $v_k \in \mathbb{Z}_p$ and uses z_k to compute:

$$\begin{aligned} A_k &= (h_s \cdot g_1^{v_k} \cdot g^{m_k} h_{\text{tp}}^{r_{\text{tp},k}} \cdot Y_{k,1})^{\frac{1}{\alpha + z_k}} \\ &= g_0^{(1 + (v_k + \iota_3 m_k + \iota_4 r_{\text{tp},k} + \iota_1 x_{k,1}) \cdot \frac{(z'+\alpha)a'-1}{\iota_2}) \prod_{j=1, j \neq i}^{q-1} (\alpha + z_j)}. \end{aligned}$$

The oracle returns the certificate $\text{cert}_k = (A_k, z_k, v_k)$.

For the q -th query, it sets $v' = \iota_2 - \iota_3 m_q - \iota_4 r_{\text{tp},q} - \iota_1 x_{q,1}$ and $A' = h_2^{a'}$. The oracle returns the certificate $\text{cert}_q = (A', z', v')$.

- **Sign**(upk, usk, m, r): On input a sender's public key upk, secret key usk, the input transaction amount m encrypted with randomness r, it firstly denote $\text{upk} = (Y'_{\text{in}}, \cdot)$. The simulator honestly computes C_{sp}, T, Ξ and picks some random $S \in \mathbb{G}$. The simulator picks some random $c, z_k, z_a, z_z, z_b, z_v, z_m, z_r, z_\psi \in \mathbb{Z}_p$. It then computes $R_{\text{cred},1}, R_{\text{cred},2}, R_{\text{cred},3}, R_{\text{cred},4}, R_{\text{cred},5}, R_{\text{cred},6}$ as in ZKReconstruct. It sets $c = H(\text{CredAuth.mpk}, C_{\text{sp}}, T, S, \Xi, R_{\text{cred},1}, R_{\text{cred},2}, R_{\text{cred},3}, R_{\text{cred},4}, R_{\text{cred},5}, R_{\text{cred},6})$ in the random oracle model. Denote $\pi_{\text{sp}} = (c, S, \Xi, z_k, z_a, z_z, z_b, z_v, z_m, z_r, z_\psi)$. Then it outputs $(C_{\text{sp}}, T, \pi_{\text{sp}})$.

In the challenge phase, the adversary outputs $(C_{\text{sp}}^* = (C_{\text{cred}}^*, B_{\text{cred}}^*), T^*, \pi_{\text{sp}}^*)$. The simulator can decrypt C_{sp}^* and obtains Y_{in}^* . If the adversary wins the game, then Y_{in}^* should not be input to the **Issue_e** oracle and $(C_{\text{sp}}^*, T^*, \pi_{\text{sp}}^*)$ should not be the output to the **Sign** oracle.

For the same $B_{\text{cred}}^*, C_{\text{cred}}^*, T^*$ and commitment $(S^*, \Xi^*, R_{\text{cred},1}^*, R_{\text{cred},2}^*, R_{\text{cred},3}^*, R_{\text{cred},4}^*, R_{\text{cred},5}^*, R_{\text{cred},6}^*)$, suppose the simulator rewinds and has two challenges c and c' and two corresponding responses $(z_k, z_a, z_z, z_b, z_v, z_m, z_r, z_\psi)$ and $(z'_k, z'_a, z'_z, z'_b, z'_v, z'_m, z'_r, z'_\psi)$.

$$\begin{aligned} \text{Denote } x^* &= \frac{z_k - z'_k}{c - c'}, a^* = \frac{z_a - z'_a}{c - c'}, z^* = \frac{z_z - z'_z}{c - c'}, b^* = \frac{z_b - z'_b}{c - c'}, v^* = \frac{z_v - z'_v}{c - c'}, \\ m^* &= \frac{z_m - z'_m}{c - c'}, r_{\text{in}}^* = \frac{z_r - z'_r}{c - c'}, r_{\text{cred}}^* = \frac{z_\psi - z'_\psi}{c - c'}. \end{aligned}$$

Then we have:

$$\begin{aligned} \hat{e}(u_1^{b^*} S^{*-z^*} g_1^{v^*} g^{m^*} h_{\text{tp}}^{r_{\text{in}}^*} g_2^{x^*}, \hat{g}_2) \cdot \hat{e}(u_1^{a^*}, W_{\text{sp}}) &= \hat{e}(h_s, \hat{g}_2)^{-1} \cdot \hat{e}(S^*, W_{\text{sp}}), \\ \Xi^* &= g_1^{a^*}, \\ \Xi^{*z^*} &= g_1^{b^*}, \\ B_{\text{cred}}^* &= g^{r_{\text{cred}}^*}, \\ C_{\text{cred}}^* &= g_2^{x^*} h_{\text{sp}}^{r_{\text{cred}}^*}, \\ T^* &= f^{x^*}. \end{aligned}$$

From the second and the third equation, we have $b^* = a^* z^*$. From the fourth and the fifth equation, we can see that $(C_{\text{cred}}^*, B_{\text{cred}}^*)$ is the ciphertext of $g_2^{x^*}$ encrypted to the auditor. From the last equation, we can see that $T^* = f^{x^*}$. From the first equation, we have:

$$\begin{aligned} \hat{e}(h_s g_1^{v^*} g^{m^*} h_{\text{tp}}^{r_{\text{in}}^*} g_2^{x^*}, \hat{g}_2) &= \hat{e}(S^{*z^*} u_1^{-b^*}, \hat{g}_2) \cdot \hat{e}(S^* u_1^{-a^*}, W_{\text{sp}}) \\ \hat{e}(h_s^{1 + \frac{(z' + \alpha)\alpha' - 1}{\iota_2} (v^* + \iota_3 m^* + \iota_4 r_{\text{in}}^* + \iota_1 x^*)}, \hat{g}_2) &= \hat{e}(S^* u_1^{-a^*}, \hat{g}_2^{z^*} \cdot W_{\text{sp}}). \end{aligned}$$

Denote the polynomial $\frac{(1+\frac{(z'+\alpha)a'-1}{\iota_2}(v^*+\iota_3m^*+\iota_4r_{\text{in}}^*+\iota_1x^*))\prod_{j=1}^{q-1}(\alpha+z_j)}{\alpha+z^*} = \sum_{j=0}^{q-2} Z_j\alpha^j + \frac{Z_{-1}}{\alpha+z^*}$ for some coefficients $Z_{-1}, Z_0, \dots, Z_{q-2} \in \mathbb{Z}_p$ which can be computed by the simulator. Then we have:

$$\begin{aligned}\hat{e}(g_0^{\sum_{j=0}^{q-2} Z_j\alpha^j + \frac{Z_{-1}}{\alpha+z^*}}, \hat{g}_2^{\alpha+z^*}) &= \hat{e}(S^*u_1^{-a^*}, \hat{g}_2^{\alpha+z^*}), \\ \hat{e}(S^*u_1^{-a^*} g_0^{-\sum_{j=0}^{q-2} Z_j\alpha^j}, \hat{g}_2^{\alpha+z^*}) &= \hat{e}(g_0^{Z_{-1}}, \hat{g}_2).\end{aligned}$$

If $Z_{-1} \neq 0$, the simulator can compute $A^* = (S^*u_1^{-a^*} g_0^{-\sum_{j=0}^{q-2} Z_j\alpha^j})^{\frac{1}{Z_{-1}}}$. It returns (A^*, z^*) as the solution to the q -SDH problem.

If $Z_{-1} = 0$, it implies that $z^* = z_k$ for some k -th Issue_e oracle output in the past. With probability $1 - 1/q$, $z^* \neq z'$. Denote such oracle input extracted the value of (m_k, r_k, x_k) and the output is (A_k, z^*, v_k) . Then we have $\hat{e}(h_s g_1^{v_k} g^{m_k} h_{\text{tp}}^{r_k} g_2^{x_k}, \hat{g}_2) = \hat{e}(A_k, \hat{g}_2^* W_{\text{sp}})$. Recall that $\hat{e}(h_s g_1^{v^*} g^{m^*} h_{\text{tp}}^{r^*} g_2^{x^*}, \hat{g}_2) = \hat{e}(S^*u_1^{-a^*}, \hat{g}_2^* W_{\text{sp}})$. If $A_k = S^*u_1^{-a^*}$, then we have $v^* = v_k, m^* = m_k, r^* = r_k$ and $x^* = x_k$ if the discrete logarithm assumption holds between g, g_1, g_2 and h_{tp} . Recall that the decryption of $(C_{\text{cred}}^*, B_{\text{cred}}^*)$ gives $g_2^{x^*}$ if the adversary wins the game. However, $x^* = x_k$ contradicts that $g_2^{x^*}$ is never queried to the Issue_e oracle.

If $Z_{-1} = 0$, $z^* = z_k$ and $A_k \neq S^*u_1^{-a^*}$ for some k -th Issue_e oracle output in the past, we show that the simulator can also solve the q -SDH. With probability $1/q$, $z^* = z'$. Then denote $A^* = S^*u_1^{-a^*}$ and $y^* = v^* + \iota_3m^* + \iota_4r_{\text{in}}^* + \iota_1x^*$:

$$\begin{aligned}A^{*\alpha+z^*} &= h_s g_1^{v^*} g^{m^*} h_{\text{tp}}^{r_{\text{in}}^*} g_2^{x^*} = h_s g_1^{v^*+\iota_3m^*+\iota_4r_{\text{in}}^*+\iota_1x^*} = h_s g_1^{y^*} = h_s^{1+y^* \cdot \frac{(z'+\alpha)a'-1}{\iota_2}}, \\ A^* &= h_s^{\frac{\iota_2 - y^* + y^* a'(\alpha+z')}{\iota_2(\alpha+z')}} = h_2^{\frac{\iota_2 - y^*}{\iota_2(\alpha+z')} + \frac{y^* a'}{\iota_2}}, \\ (A^* h_2^{\frac{-y^* a'}{\iota_2}})^{\frac{\iota_2}{\iota_2 - y^*}} &= h_2^{\frac{1}{\alpha+z'}} = g_0^{\frac{\prod_{i=1}^{q-1}(\alpha+z_i)}{\alpha+z'}}\end{aligned}$$

Denote the polynomial $\frac{\prod_{i=1}^{q-1}(\alpha+z_i)}{\alpha+z'}$ for some coefficients $Z'_{-1}, Z'_0, \dots, Z'_{q-2} \in \mathbb{Z}_p$ which can be computed by the simulator. Note that $Z'_{-1} \neq 0$ in this case since the simulator sets $z' \neq z_i$ for all i . Then the simulator can compute $A' = [(A^* h_2^{\frac{-y^* a'}{\iota_2}})^{\frac{\iota_2}{\iota_2 - y^*}} g_0^{-\sum_{i=0}^{q-2} Z'_i \alpha^i}]^{\frac{-1}{Z'_{-1}}}$. It returns (A', w') as the solution to the q -SDH problem. \square

Proof of Unforgeability for SP.

Proof. The simulator is given the DL problem instance (g_0, y_0) . It picks a random $\delta \in \mathbb{Z}_p$ and sets $g_2 = g_0, f = g_0^\delta$. The rest of $\text{param}, \text{ask}_{\text{sp}}, \text{epk}, \text{esk}$ are honestly generated and are given to the adversary.

The oracles are simulated as follows:

- **KeyGen()**: The simulator runs $(\text{usk}, \text{upk}) \leftarrow \text{UserKeyGen}()$, stores (usk, upk) in a list \mathcal{L} (which is initially empty) and outputs upk . Except for one time, the simulator picks a random $Y_2 \in \mathbb{G}$ and returns $\text{upk} = (y_0, Y_2)$ and puts (\perp, upk) in \mathcal{L} .
- **Corrupt(upk)**: On input upk , it searches $(\text{usk}, \text{upk}) \in \mathcal{L}$ and returns usk . It declares failure and exits if $\text{upk} = (y_0, Y_2)$.
- **Issue_u(upk)**: On input upk , it searches $(\text{usk}, \text{upk}) \in \mathcal{L}$ and runs **Issue** honestly as the user. If $\text{upk} = (y_0, Y_2)$, it runs the simulator of the zero-knowledge proof π_{ci} to obtain a proof without the knowledge of the corresponding secret key.
- **Sign(upk, cred, m, r)**: on input a sender's public key upk , credential cred for the input transaction amount m encrypted with randomness r , it searches $(\text{usk}, \text{upk}) \in \mathcal{L}$. If $\text{upk} \neq (y_0, Y_2)$, it runs the **CredSign** honestly. If $\text{upk} = (y_0, Y_2)$, the simulator honestly computes C_{sp}, S, Ξ and sets $T = y_0^\delta$. The simulator picks some random $c, z_k, z_a, z_z, z_b, z_v, z_m, z_r, z_\psi \in \mathbb{Z}_p$. It then computes $R_{\text{cred},1}, R_{\text{cred},2}, R_{\text{cred},3}, R_{\text{cred},4}, R_{\text{cred},5}, R_{\text{cred},6}$ as in **ZKReconstruct**. It sets $c = H(\text{CredAuth.mpk}, C_{\text{sp}}, T, S, \Xi, R_{\text{cred},1}, R_{\text{cred},2}, R_{\text{cred},3}, R_{\text{cred},4}, R_{\text{cred},5}, R_{\text{cred},6})$ in the random oracle model. Denote $\pi_{\text{sp}} = (c, S, \Xi, z_k, z_a, z_z, z_b, z_v, z_m, z_r, z_\psi)$. Then it outputs $(C_{\text{sp}}, T, \pi_{\text{sp}})$.

In the challenge phase, the adversary outputs $(C_{\text{sp}}^* = (C_{\text{cred}}^*, B_{\text{cred}}^*), T^*, \pi_{\text{sp}}^*)$. The simulator can decrypt C_{sp}^* and obtains Y_{in}^* . If the adversary wins the game, then Y_{in}^* should not be input to the **Corrupt** oracle and $(C_{\text{sp}}^*, T^*, \pi_{\text{sp}}^*)$ should not be the output to the **Sign** oracle.

Similar to the soundness proof, the simulator rewinds and extract some x^*, r_{cred}^* such that $C_{\text{cred}}^* = g_2^{x^*} h_{\text{sp}}^{r_{\text{cred}}^*}$ and $B_{\text{cred}}^* = g^{r_{\text{cred}}^*}$. Suppose there are q_c queries to the **KeyGen** oracle. With probability $1/q_c$, $Y_{\text{in}}^* = y_0$. In this case, the simulator can return x^* as the solution to the DL problem. \square

Proof of Anonymity for SP.

Proof. The simulator is given the DDH problem instance (g_0, g_0^a, g_0^b, Z) and to decide if $Z = g_0^{ab}$. It picks a random $\delta_1, \delta_2 \in \mathbb{Z}_p$ and sets $g = g_0, g_2 = g_0^{\delta_2}, h_{\text{sp}} = g_0^a, f = g_0^{a\delta_1}$. The rest of $\text{param}, \text{epk}, \text{esk}$ are honestly generated and are given to the adversary.

- **KeyGen()**: The simulator runs $(\text{usk}, \text{upk}) \leftarrow \text{UserKeyGen}()$, stores (usk, upk) in a list \mathcal{L} (which is initially empty) and outputs upk . Except for one time, the simulator picks a random $Y_2 \in \mathbb{G}$ and returns $\text{upk}^* = (g_0^b, Y_2)$ and puts (\perp, upk^*) in \mathcal{L} .
- **Corrupt(upk)**: On input upk , it searches $(\text{usk}, \text{upk}) \in \mathcal{L}$ and returns usk . It declares failure and exits if $\text{upk} = \text{upk}^*$.

In the challenge phase, the adversary \mathcal{A}_1 outputs $\{\text{upk}_i^*, \text{cred}_i^*, m_i^*, r_i^*\}_{i \in [0,1]}$. If both $\text{upk}_0^* \neq \text{upk}^*$ and $\text{upk}_1^* \neq \text{upk}^*$, the simulator declares failure and exits. Without loss of generality, assume $\text{upk}_0^* = \text{upk}^*$. The simulator picks some random $r_{\text{enc}} \in \mathbb{Z}_p$ and computes $C_{\text{sp}}^* = (C_{\text{cred}}^*, B_{\text{cred}}^*), T^*$, where:

$$B_{\text{cred}}^* = (g_0^b)^{r_{\text{enc}}}, \quad C_{\text{cred}}^* = (g_0^b)^{\delta_2} Z^{r_{\text{enc}}}, \quad T^* = Z^{\delta_1}.$$

The simulator picks some random $S^*, \Xi^* \in \mathbb{G}$ and $c^*, z_k^*, z_a^*, z_z^*, z_b^*, z_v^*, z_m^*, z_r^*, z_\psi^* \in \mathbb{Z}_p$. It then computes $R_{\text{cred},1}^*, R_{\text{cred},2}^*, R_{\text{cred},3}^*, R_{\text{cred},4}^*, R_{\text{cred},5}^*, R_{\text{cred},6}^*$ as in ZKReconstruct . It sets $c^* = H(\text{CredAuth.mpk}, C_{\text{sp}}^*, T^*, S^*, \Xi^*, R_{\text{cred},1}^*, R_{\text{cred},2}^*, R_{\text{cred},3}^*, R_{\text{cred},4}^*, R_{\text{cred},5}^*, R_{\text{cred},6}^*)$ in the random oracle model. Denote $\pi_{\text{sp}}^* = (c^*, S^*, \Xi^*, z_k^*, z_a^*, z_z^*, z_b^*, z_v^*, z_m^*, z_r^*, z_\psi^*)$. Then it returns $(C_{\text{sp}}^*, T^*, \pi_{\text{sp}}^*)$ to the adversary \mathcal{A}_2 .

If the adversary outputs the correct guess of upk^* , then the simulator outputs $Z = g_0^{ab}$ to the DDH problem. Otherwise, the simulator outputs $Z \neq g_0^{ab}$ to the DDH problem. \square