PAChain: Private, Authenticated & Auditable Consortium Blockchain *

Tsz Hon Yuen

The University of Hong Kong, Hong Kong
thyuen@cs.hku.hk

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, regulators or law enforcement agencies. Authenticated transaction means that only authorized parties can be involved in the transaction. This functionality is especially important to use with auditability: even if the auditor can extract the public key used in a transaction, he can do nothing if the public key is not associated with a registered user in the system. In this paper, we present a private, authenticated and auditable consortium blockchain, using a number of cryptographic building blocks.

Keywords: Blockchain · privacy.

1 Introduction

Blockchain is a fast-growing field of technology since it is recognized as the core component of Bitcoin [13]. Blockchain can serve as a distributed ledger in a peer-to-peer network to provide publicly verifiable data. Blockchain can be mainly classified into three categories: public, private and consortium blockchain. The *public blockchain* is an open, permissionless system such that every one is allowed to join as a user or miner freely. Bitcoin and most cryptocurrencies falls into this category. Public blockchain is suitable for decentralized network without trusted authority. However, they usually have limited throughput. The *private blockchain* is a closed, permissioned system such that a user must be validated in order to use the system. It is a traditional centralized system with auditability attached. 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, without consolidating power with a single party as the private blockchain. Consortium blockchain is suitable for organizational collaboration.

Consortium blockchain received a great interest from the industry due to the similarity with the existing business model, especially in the finance industry. 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.

* This is the full verion of the paper in CANS 2019 [19].

		Sender Privacy	Recipient Privacy	Transaction Privacy	Auditability	Authentication	Efficiency
Public blockchain	Monero/RingCT-based solutions	◔	●	●			◐
	Zcash/zk-SNARK-based solutions	●	●	●			◕
	DAP [10]	●	●	●	●		◕
Consortium/ Private blockchain	R3 Corda, [12], Hyperledger Fabric	◕	◕	◕		●	●
	Fabric experiment	◐	◕	◕	◕	◐	◕
	PRCash[18]	◐	◐	●	◐		◕
	This paper	◔	●	●	●	●	◔

Table 1. Comparison with Existing Privacy-Preserving Blockchain Schemes

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). Cryptocurrencies such as Monero and Zcash offer privacy in the public blockchain. There are also a number of academic and industrial solutions for privacy-preserving consortium blockchain. Details will be discussed in §3.1.

Our Contributions: Privacy, Authenticated & Auditable in Consortium Blockchain. Auditability is essential for financial blockchain applications and unconditional anonymity may not be desirable. Financial institution has to undertake both internal and external audit for checking if there is any money laundering or terrorist-related activities, under regulations from the government. Auditing is usually performed by sample checking of all transaction records. In case of court order, the institute has to provide the complete information of a particular transaction to the court. For all of 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. Therefore, our construction is suitable to be deployed in real world business use cases. 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.

2 PAChain Overview

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

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 1.0 [1], 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 Txs 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 Txs 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 Txs. 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.
5. Certificate Authority(CA): CA issues certificate for the public key of clients. Only authorized party can be involved in a Tx.

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. Ledger provides a verifiable history of all successful (and also unsuccessful) state changes, occurring during the operation of the system. Ledger is constructed by the orderer as a totally ordered hash-chain of blocks of (valid or invalid) Txs. Each block contains a set of totally ordered Txs. The ledger is distributed in the system and can be accessed by any nodes.

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.[2]

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 (only the first valid Tx is accepted). 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.

[1] Hyperledger Fabric Architecture Explained. http://hyperledger-fabric.readthedocs.io/en/latest/arch-deep-dive.html

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

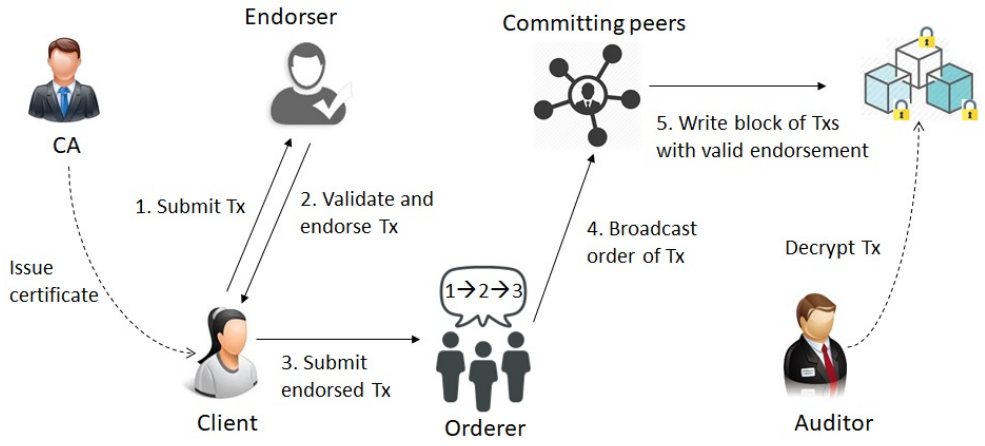

Fig. 1. Nodes and Workflow of PAChain

2.2 High-Level Description of PAChain

We briefly describe how we can achieve privacy and auditability in our 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: It is achieved by the sender Alice using anonymous credential in Step 1 of the workflow. The credential is issued by the endorser of the last corresponding Tx received by Alice, 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. Our method is more efficient than the linkable ring signature approach in Monero and the zk-SNARKs approach in Zcash. We take advantage of having some semi-trusted endorsers 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: It is achieved by generating one-time ephemeral key via Diffie-Hellman protocol between the sender and 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 the zero-knowledge proof that the recipients are authorized into the generation of one-time ephemeral key (see §5.2). It is a new security requirement which does not exist in the public blockchain. This requirement is a major difference between PAChain and [10].

Transaction Privacy with Auditability: Our transaction amount privacy is achieved by using 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 range, and the total Tx input and output amount are balanced. The major challenge is that using Paillier encryption with zero-knowledge range proof is not efficient. To be more specific, encrypting 2048-bit plaintext and performing the binary-decomposition range proof is an overkill for 64-bit of transaction amount (see §4). Therefore, we propose the use of modified ElGamal encryption with signature-based range proof to enhance the efficiency (see §4.2). The size of the zero knowledge proof is independent to the size of the range. It cannot be used in public blockchain since it requires trusted setup.

Auditability: Auditability for sender and recipient identity can be achieved by encrypting their public keys to the auditor, followed the the zero-knowledge proof of the correctness of such encryption in Step 1 of the workflow. We have discussed the auditability of transaction amount above.

Our construction can also be modified to provide fine-grained privacy policy. For example, if the transaction amount M is under \$10,000, then the bank does not need to perform checking for anti-money laundering. As a result, we can leverage our efficient range proof to run the proof of knowledge:

$$PoK\{(M, \text{sender ID, recipient ID}) : 0 \leq M \leq 10,000 \\ \text{or Encrypt } (M, \text{sender ID, recipient ID}) \text{ to the auditor}\}.$$

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. If we use the Paillier encryption instead, then it is difficult to find a suitable and efficient range proof. 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. It cannot be used in public blockchain since it requires trusted setup.

2.3 Threat Model

We assume that the system parameters are generated honestly. 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.[3]

We do not consider network-level privacy issues, such as tracing the sender's IP address or analyzing meta-data in network packets. The correctness of the consensus algorithm is not considered in this paper.

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 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 created in 2014 that provides privacy by linkable ring signature, stealth address and Ring Confidential Transactions [14]. 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 is a cryptocurrency created in 2016 that offers privacy and selective transparency of Txs by using zero-knowledge proofs (zk-SNARK) on special *shielded* transactions [3]. The major disadvantage of Zcash is the large signing key size of 896MB, long key generation time of 8 minutes

[3] One 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).

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 [10]. They tackle the accountability problem in the public blockchain by using the zk-SNARK approach. Hence, it is also not efficient.

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, Txs are handled by different channels and users can only view Txs in their own channel. Therefore, privacy is maintained by the system's access control policy. The level of privacy is lower than that of Monero and Zcash, which are not affected by system administrators. 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 [9] with Fabric [4]. The identity mixer provides a 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. Recently, a private industrial blockchain is proposed in [12], 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[18] is a centrally-issued cryptocurrency with privacy and auditability. They provide anonymity of the sender and recipient identity. A mixing technique is used to obfuscate the relation between multiple inputs and outputs. However, there are still certain linkability between Txs. Anyone can see that the input of a Tx comes from which previous Tx output. For auditability, PRCash provides prefect 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 $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ 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 [14], 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 Txs can be audited by internal auditors or some external regulators. Therefore, it is

[4] https://jira.hyperledger.org/browse/FAB-2005

necessary to replace additive homomorphic commitment with additive homomorphic encryption in confidential transaction, such that the decryption key is hold by the auditor. However, technical difficulties arise when combining the additive homomorphic Paillier encryption [15] 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* method by Boudot [6], *Multi-Base Decomposition* method by Bellare and Goldwasser [2], and *Signature-based* method by Camenisch *et al.* [8].

- **Square Decomposition.** Boudot [6] used a mathematical theorem that any positive integer can be written as a sum of four squares, to carry out a range proof. 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 [2] used the binary decomposition of x to perform a bit-by-bit range proof in $[0, R = 2^k)$. 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 [7] is proposed to reduce the proof size to $O(\log k)$.
- **Signature-based.** Camenisch *et al.* [8] 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 [15] is the most common additive homomorphic encryption to date. However, combining Paillier encryption with existing range proof is not trivial in blockchain applications. Therefore, 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:

- $(\mathsf{param}, \mathsf{ask}_{\mathsf{tp}}) \leftarrow \mathsf{Setup}(1^\lambda, R)$. On input a security parameter 1^λ and the range parameter R, it outputs the auditor secret key $\mathsf{ask}_{\mathsf{tp}}$ 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.
- $(\{C_{\mathsf{out},j}, r_{\mathsf{out},i}\}_{j\in[1,n']}, \pi_{\mathsf{tp}})/\perp \leftarrow \mathsf{TxPrivacySpend}\ (\{M_{\mathsf{out},j}\}_{j\in[1,n']}, \{C_{\mathsf{in},i}, M_{\mathsf{in},i}, r_{\mathsf{in},i}\}_{i\in[1,n]})$. On input n' output transaction amount $M_{\mathsf{out},j}$, n transaction input ciphertext $C_{\mathsf{in},i}$, amount $M_{\mathsf{in},i}$ and randomness $r_{\mathsf{in},i}$, it outputs $\perp$ if $C_{\mathsf{in},i}$ is not a valid ciphertext for $(M_{\mathsf{in},i}, r_{\mathsf{in},i})$ for some $i \in [1, n]$, or $\sum_{i=1}^{n} M_{\mathsf{in},i} \neq \sum_{j=1}^{n'} M_{\mathsf{out},j}$ [5]. For each $M_{\mathsf{out},j}$, it runs the following sub-protocol:
  - $(C_{\mathsf{enc}}, r_{\mathsf{enc}}, \pi_{\mathsf{enc}}) \leftarrow \mathsf{EncProof}(M)$. On input a message $M \in [0, R)$, it outputs a ciphertext C_{enc} encrypted to the auditor, its encryption randomness r_{enc} and a zero-knowledge proof π_{enc} that C encrypts M and $M \in [0, R)$.

  It obtains $(C_{\mathsf{out},j}, r_{\mathsf{out},j}, \pi_{\mathsf{enc},j}) \leftarrow \mathsf{EncProof}(M_{\mathsf{out},j})$. It computes π_{tp} which includes all $\pi_{\mathsf{enc},j}$ and the zero-knowledge proof of $\sum_{i=1}^{n} M_{\mathsf{in},i} = \sum_{j=1}^{n'} M_{\mathsf{out},j}$. It outputs $(\{C_{\mathsf{out},j}, r_{\mathsf{out},i}\}_{j\in[1,n']}, \pi_{\mathsf{tp}})$.

[5] If it is the genesis transaction where money is created, input ciphertext and amount are not needed. The verification of genesis transaction also does not need to check for it. We omit the description for genesis transaction for simplicity.

– $1/0 \leftarrow \mathbf{TxPrivacyVerify}(\{C_{\mathsf{in},i}\}_{i\in[1,n]}, \{C_{\mathsf{out},j}\}_{j\in[1,n']}, \pi_{\mathsf{tp}})$. On input n transaction input ciphertext $C_{\mathsf{in},i}$, n' transaction output ciphertext $C_{\mathsf{out},j}$ and a proof π_{tp}, it outputs 1 if π_{tp} is a valid zero-knowledge proof, and outputs 0 otherwise. It includes running the following sub-protocol $\mathsf{EncVerify}(C_{\mathsf{out},j}, \pi_{\mathsf{enc},j})$ for each $\pi_{\mathsf{enc},j} \in \pi_{\mathsf{tp}}$:
  – $1/0 \leftarrow \mathsf{EncVerify}(C_{\mathsf{enc}}, \pi_{\mathsf{enc}})$. On input a ciphertext C_{enc} and a proof π_{enc}, it outputs 1 if π_{enc} is a valid zero-knowledge proof, and outputs 0 otherwise.
– $M \leftarrow \mathbf{Decrypt}(\mathsf{ask}_{\mathsf{tp}}, C)$. On input the auditor secret key $\mathsf{ask}_{\mathsf{tp}}$ and a ciphertext C, it outputs the decrypted transaction amount M.

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

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* $(\mathsf{param}, \mathsf{ask}_{\mathsf{sp}}) \leftarrow \mathbf{Setup}(1^\lambda, R)$*; $\mathcal{A}$ outputs* $(\{C_{\mathsf{in},i}\}_{i\in[1,n]}, \{C_{\mathsf{out},j}\}_{j\in[1,n']}, \pi_{\mathsf{tp}})$*, then*

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

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

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

where $\mathcal{A}$ can query the Spend *oracle to request generating an honest run of* **TxPrivacySpend**. *We additionally require that $\{C_{\mathsf{in},i}\}$ 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* $\mathbf{Setup}(1^\lambda, R)$*, $\mathcal{A}_1$ outputs* $(\{M^{(0)}_{\mathsf{out},j}, M^{(1)}_{\mathsf{out},j}\}_{j\in[1,n']}, \{C^*_{\mathsf{in},i}, M^*_{\mathsf{in},i}, r^*_{\mathsf{in},i}\}_{i\in[1,n]})$*, and when $\mathcal{A}_2$ is given* $(\{C^*_{\mathsf{out},j}\}_{j\in[1,n']}, \pi^*_{\mathsf{tp}})$*, where* $(\{C^*_{\mathsf{out},j}, r^*_{\mathsf{out},i}\}_{j\in[1,n']}, \pi^*_{\mathsf{tp}}) \leftarrow \mathbf{TxPrivacySpend}\ (\{M^{(b)}_{\mathsf{out},j}\}_{j\in[1,n']}, \{C^*_{\mathsf{in},i}, M^*_{\mathsf{in},i}, r^*_{\mathsf{in},i}\}_{i\in[1,n]})$ *for some random bit b, $\mathcal{A}_1$ outputs its guess b', then:*

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

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

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 $\mu_0, \ldots, \mu_{\ell-1}$ which are

smaller than the message space $\mathfrak{u}$ by $M = \sum_{j=0}^{\ell-1} \mu_j \mathfrak{u}^j$. As a result, we encrypt each μ_j by additive homomorphic ElGamal encryption. The small message space of $\mathfrak{u}$ guarantees efficient decryption. In our implementation, we consider 64-bit transaction amount and the auditor uses a pre-computation table of $(g, g^2, \ldots, g^{\mathfrak{u}-1})$ for efficient decryption. We can choose $\ell = 4, \mathfrak{u} = 65536$ 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 [8]. 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 $R = \mathfrak{u}^\ell$, 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 $i \in [0, \mathfrak{u}-1]$. Suppose $H : \{0,1\}^* \to \mathbb{Z}_p$ is a collision resistant hash function. In addition, suppose the auditor picks a random secret key $\mathsf{ask}_{\mathsf{tp}}$ in $\mathbb{Z}_p$ and outputs its public key $h_{\mathsf{tp}} = g^{\mathsf{ask}_{\mathsf{tp}}}$. It outputs $\mathsf{param} = (g, g_0, h_{\mathsf{tp}}, \hat{g}, \hat{e}(g,\hat{g}), \hat{\mathfrak{Y}}, \mathfrak{u}, \ell, H, \mathfrak{A}_0, \ldots, \mathfrak{A}_{\mathfrak{u}-1})$.
- $\mathsf{TxPrivacySpend}$. On input n' output transaction amount $M_{\mathsf{out},j}$, n transaction input ciphertext $C_{\mathsf{in},i}$, amount $M_{\mathsf{in},i}$ and randomness $r_{\mathsf{in},i}$, it outputs $\perp$ if $C_{\mathsf{in},i}$ is not a valid ciphertext for $(M_{\mathsf{in},i}, r_{\mathsf{in},i})$ for some $i \in [1,n]$, or $\sum_{i=1}^{n} M_{\mathsf{in},i} \neq \sum_{j=1}^{n'} M_{\mathsf{out},j}$.

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

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

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

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

   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:

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

   It computes $\tilde{c} = H(\mathsf{param}, \{V_k, a_k, B_k, C_k, D_k,\ E_k\}_{k\in[0,\ell-1]})$ 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 $\pi_{\mathsf{enc}} = (\{V_k, z_{\mu_k},\ z_{v_k}, z_{r_k}\}_{k\in[0,\ell-1]}, \tilde{c})$.
4. It outputs $(C_{\mathsf{enc}}, r_{\mathsf{enc}}, \pi_{\mathsf{enc}})$.

Observe that $C_{\mathsf{out},j} = \{C_{j,k}, B_{j,k}\}_{k\in[0,\ell-1]}$. For simplicity, denote $C'_{\mathsf{out},j} = \prod_{k=0}^{\ell-1} C_{j,k}^{\mathfrak{u}^k}$ and $B'_{\mathsf{out},j} = \prod_{k=0}^{\ell-1} B_{j,k}^{\mathfrak{u}^k}$. 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 $x_{\mathsf{tp}} = \sum_{j=1}^{n'} r_{\mathsf{out},j} - \sum_{i=1}^{n} r_{\mathsf{in},i}$, such that

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

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

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

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

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

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

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

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

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

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* $\mathfrak{u}$*-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 Txs 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 [16]. 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 Txs mixed. In Monero, the recipient uses *stealth address* [14], 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 [3].

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. $(\mathsf{param}, \mathsf{ask}_{\mathsf{rp}}) \leftarrow$ **Setup**(1^λ). On input a security parameter 1^λ, it outputs the auditor secret key $\mathsf{ask}_{\mathsf{rp}}$ 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. $(\mathsf{usk}, \mathsf{upk}) \leftarrow$ **UserKeyGen**(). The user outputs his long-term secret key usk and long-term public key upk.

3. $(\mathsf{otpk}, R_{tx}) \leftarrow \mathbf{OneTimePkGen}(\mathsf{upk})$. On input a long-term public key upk, it outputs a one-time public key otpk and a transaction randomness R_{tx}.
4. $\mathsf{otsk}/\perp \leftarrow \mathbf{OneTimeSkGen}(\mathsf{otpk}, R_{tx}, \mathsf{usk})$. On input a long-term public key upk, a transaction randomness R_{tx} and long-term secret key usk, it outputs a one-time secret key otsk or $\perp$ if otpk is not related to usk.
5. $(\mathsf{cask}, \mathsf{capk}) \leftarrow \mathbf{CAKeyGen}()$. The CA outputs his secret key cask and public key capk.
6. $\mathsf{cert} \leftarrow \mathbf{CertIssue}(\mathsf{capk}, \mathsf{upk})$. This is an interactive algorithm runs between the CA and a user, with common input CA's public key capk and user's long-term public key 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. $(\pi_{\mathsf{otsk}}, \mathsf{otpk}_r, R_{tx,r}, C_{\mathsf{rp}}, \pi_{\mathsf{rp}}) \leftarrow \mathbf{RecPrivacySpend}$ $(\mathsf{capk}, \mathsf{otpk}_s, R_{tx,s}, \mathsf{usk}, \mathsf{upk}_r, \mathsf{cert}_r)$. On input the CA's public key capk, the sender's input UTXO (including his one-time public key otpk_s and transaction randomness $R_{tx,s}$), the sender's long-term secret key usk[6], the recipient's long term public key upk_r and his certificate cert_r, the sender runs as follows:
   (a) It first runs $\mathsf{otsk} \leftarrow \mathbf{OneTimeSkGen}(\mathsf{otpk}_s, R_{tx,s}, \mathsf{usk})$. It outputs a zero-knowledge proof π_{otsk} of knowing otsk.
   (b) It also outputs $(\mathsf{otpk}_r, R_{tx,r}) \leftarrow \mathbf{OneTimePkGen}(\mathsf{upk}_r)$.
   (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 $\mathsf{cert}_r \leftarrow \mathbf{CertIssue}(\mathsf{capk}, \mathsf{upk}_r)$.

   The sender outputs $(\pi_{\mathsf{otsk}}, \mathsf{otpk}_r, R_{tx,r}, C_{\mathsf{rp}}, \pi_{\mathsf{rp}})$.[7]
8. $1/0 \leftarrow \mathbf{Verify}(\mathsf{capk}, \pi_{\mathsf{otsk}}, \mathsf{otpk}_s, \mathsf{otpk}_r, R_{tx,r}, C_{\mathsf{rp}}, \pi_{\mathsf{rp}})$. On input the CA's public key capk, the sender's one-time public key otpk_s with the proof of corresponding one-time secret key π_{otsk}, the recipient's one-time public key 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 zero-knowledge proofs, and outputs 0 otherwise.
9. $\mathsf{upk}_r \leftarrow \mathbf{Decrypt}(\mathsf{ask}_{\mathsf{rp}}, C_{\mathsf{rp}})$. On input the auditor secret key $\mathsf{ask}_{\mathsf{rp}}$ 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.

The security properties of a recipient privacy protocol are formalized as follows.

Definition 4 (Soundness). *A recipient privacy protocol is called sound if for all PPT adversary $\mathcal{A}$, it holds that given* $(\mathsf{param}, \mathsf{ask}_{\mathsf{rp}}, \mathsf{capk}) \leftarrow \mathbf{Setup}(1^\lambda); (\mathsf{cask}, \mathsf{capk}) \leftarrow \mathbf{CAKeyGen}()$, *$\mathcal{A}$ outputs* $(\pi_{\mathsf{otsk}}, \mathsf{otpk}_s, \mathsf{otpk}^*, R^*_{tx}, C^*_{\mathsf{rp}}, \pi^*_{\mathsf{rp}}), (\pi^*_{\mathsf{otsk}}, \mathsf{otpk}^*, \mathsf{otpk}_r, R_{tx,r}, C_{\mathsf{rp}}, \pi_{\mathsf{rp}})$, *then:*

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

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

– $\mathsf{KeyGen}()$*: it runs* $(\mathsf{usk}, \mathsf{upk}) \leftarrow \mathbf{UserKeyGen}()$, *stores* $(\mathsf{usk}, \mathsf{upk})$ *in a list $\mathcal{L}$ (which is initially empty) and outputs* upk.

[6] If it is the genesis transaction, input UTXO and the sender's key are not needed. The verification of genesis transaction also does not need to check the input UTXO. We omit the description for genesis transaction for simplicity.

[7] In 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.

– $\mathsf{Corrupt}(\mathsf{upk})$*: on input* upk*, it searches* $(\mathsf{usk}, \mathsf{upk}) \in \mathcal{L}$ *and returns* usk*. It returns* $\perp$ *if no such key is found in* $\mathcal{L}$*.*
– $\mathsf{Issue}(\mathsf{upk})$*: on input* upk*, it runs as the CA of the* **CertIssue***(*capk*,* upk*) protocol with private input* cask*. It outputs* cert*.*
– $\mathsf{Spend}(\mathsf{upk}_s, \mathsf{otpk}_s, R_{tx,s}, \mathsf{upk}_r, \mathsf{cert}_r)$*: on input a sender public key* upk_s*, the* $(\mathsf{otpk}_s, R_{tx,s})$ *from the previous transaction*[8]*, an authorized recipient* upk_r *with certificate* cert_r*, it firstly retrieves* $(\mathsf{usk}_s, \mathsf{upk}_s) \in \mathcal{L}$*. Then it outputs* $(\pi_{\mathsf{otsk}}, \mathsf{otpk}_r, R_{tx,r}, C_{\mathsf{rp}}, \pi_{\mathsf{rp}}) \leftarrow$ **RecPrivacySpend** $(\mathsf{capk}, \mathsf{otpk}_s, R_{tx,s}, \mathsf{usk}_s, \mathsf{upk}_r, \mathsf{cert}_r)$*.*

We additionally require that $\mathsf{upk}^* \leftarrow \mathbf{Decrypt}(\mathsf{ask}_{\mathsf{rp}}, C^*_{\mathsf{rp}})$*,* upk^* *is never queried to the* Issue *oracle, and* $(\mathsf{upk}^*, \mathsf{otpk}^*, R^*_{tx}, \cdot, \cdot)$ *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* $\mathsf{param}, \mathsf{cask}, \mathsf{capk}$*, where* $(\mathsf{param}, \mathsf{ask}_{\mathsf{rp}}) \leftarrow \mathbf{Setup}(1^\lambda)$*,* $(\mathsf{cask}, \mathsf{capk}) \leftarrow \mathbf{CAKeyGen}()$ *and outputs* $(\mathsf{otpk}_s, R_{tx,s}, \mathsf{usk}, \mathsf{upk}_{r,0}, \mathsf{cert}_{r,0}, \mathsf{upk}_{r,1}, \mathsf{cert}_{r,1})$*; and* $\mathcal{A}_2$ *is given* $(C^*_{\mathsf{rp}}, \pi^*_{\mathsf{rp}})$*, where* $(\pi^*_{\mathsf{otsk}}, \mathsf{otpk}^*_r, R^*_{tx,r}, C^*_{\mathsf{rp}}, \pi^*_{\mathsf{rp}}) \leftarrow \mathbf{RecPrivacySpend}(\mathsf{capk}, \mathsf{otpk}_s, R_{tx,s}, \mathsf{usk}, \mathsf{upk}_{r,b}, \mathsf{cert}_{r,b})$ *for some random bit* b*,* $\mathcal{A}_2$ *outputs its guess* b'

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

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

5.2 Recipient Privacy for PAChain

Stealth address [14] 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 [1], 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 $\mathsf{ask}_{\mathsf{rp}} \in \mathbb{Z}_p$ and outputs its public key $h_{\mathsf{rp}} = g^{\mathsf{ask}_{\mathsf{rp}}}$. It outputs the public parameters $\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, g, h_{\mathsf{rp}}, g_2, g_3, h_2, \hat{g}_2, H, H')$.

- $\mathsf{UserKeyGen}$. 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 $(\mathsf{usk} = (x_1, x_2), \mathsf{upk} = (Y_1, Y_2))$.

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

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

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

- $\mathsf{CertIssue}$. On input CA's public key capk and the user long-term public key $\mathsf{upk} = (Y_1, \cdot)$, 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 $\mathsf{cask} = \beta$ 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.

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

[8] It can be empty if it is the genesis transaction.

1. The sender with usk decides one or more UTXOs that he wants to spend. For simplicity, assume he picks one UTXO with $(\mathsf{otpk}_s, R_{tx,s})$. He runs $\mathsf{otsk}_s \leftarrow$ **OneTimeSkGen**(param, $\mathsf{otpk}_s, R_{tx,s}, \mathsf{usk}$). It runs a zero-knowledge proof of discrete logarithm: $\mathsf{otsk}_s = \log_{g_2} \mathsf{otpk}_s$. 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 $\mathsf{upk}_r = (Y_1, Y_2)$. The sender generates the recipient's one-time public key by running **OneTimePkGen**. The sender obtains $\mathsf{otpk}_r, R_{tx,r}$ 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 $r_{\mathsf{cert}} \in \mathbb{Z}_p$ and computing $C_{\mathsf{rp}} = (C_{\mathsf{cert}} = Y_1 \cdot h_{\mathsf{rp}}^{r_{\mathsf{cert}}}, B_{\mathsf{cert}} = g^{r_{\mathsf{cert}}})$.
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:

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

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

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

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

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

   (c) **ZKResponse:** It computes:

$$z_\omega = r_\omega + c \cdot w, \quad z_\tau = r_\tau + c \cdot h_{tx}, \quad z_\rho = r_\rho + c \cdot \rho,$$
$$z_c = r_c + c \cdot r_{\mathsf{cert}}, \quad z_\sigma = r_\sigma + c \cdot r_{\mathsf{cert}} \cdot \rho, \quad z_s = r_s + c \cdot \rho \cdot s.$$

   It outputs $\pi_{\mathsf{rp}} = (c, \Theta, z_\omega, z_\tau, z_\rho, z_c, z_\sigma, z_s)$.
5. Output $\pi_{\mathsf{otsk}}, \mathsf{otpk}_s, \mathsf{otpk}_r, R_{tx,r}, C_{\mathsf{rp}}$ and π_{rp}.

- RecPrivacyVerify. On input param, $\mathsf{capk}, \pi_{\mathsf{otsk}}, \mathsf{otpk}_s, \mathsf{otpk}_r$, $R_{tx,r}$, C_{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 $\pi_{\mathsf{rp}} = (c, \Theta, z_\omega, z_\tau, z_\rho, z_c, z_\sigma, z_s)$ is as follows.

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

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

2. **ZKCheck:** It computes $c' = H(\mathsf{CertAuth.mpk}, C_{\mathsf{rp}}, \Theta, R_{\mathsf{cert},1}, R_{\mathsf{cert},2}, R_{\mathsf{cert},3}, R_{\mathsf{cert},4})$. 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 full version of the paper.

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 (LRS) for hiding the real UTXOs used with other UTXOs (by the anonymity property of LRS), preventing double spending (by the linkability property of LRS) and ensuring transaction correctness (by the unforgeability property of LRS) at the same time [14]. The level of anonymity is related to number of UTXOs (denote as L) included in LRS. 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 $O(1)$ [17], 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:

- $(\mathsf{param}, \mathsf{ask}_{\mathsf{rp}}) \leftarrow \mathbf{Setup}(1^\lambda)$. On input a security parameter 1^λ, it outputs the auditor secret key $\mathsf{ask}_{\mathsf{sp}}$ 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.
- $(\mathsf{usk}, \mathsf{upk}) \leftarrow \mathbf{UserKeyGen}()$. The user outputs his long-term secret key usk and long-term public key upk.
- $(\mathsf{esk}, \mathsf{epk}) \leftarrow \mathbf{EndorserKeyGen}()$. The endorser outputs his secret key esk and public key epk.
- $\mathsf{cred} \leftarrow \mathbf{CredIssue}(\mathsf{epk}, \mathsf{upk}, C_{\mathsf{tp}})$. This is an interactive algorithm runs between the endorser and a user, with common input endorser public key epk, user's public key 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 cred to the user.
- $(C_{\mathsf{sp}}, T, \pi_{\mathsf{sp}}) \leftarrow \mathbf{CredSign}(\mathsf{usk}, \mathsf{upk}, \mathsf{cred}, m, r)$. On input a senders' secret/public key $\mathsf{usk}, \mathsf{upk}$, the credential cred corresponding to the input transaction amount m encrypted with randomness r, it runs as follows:
  1. It encrypts upk to the auditor and obtains the ciphertext C_{sp}.
  2. It computes a linking tag T from usk.
  3. It generates a zero-knowledge proof π_{sp} for (1) the correctness of C_{sp}, T above; (2) knowing $\mathsf{cred}, m, r, \mathsf{upk}$ such that $(C_{\mathsf{tp}}, r) = \mathbf{Enc\text{-}Proof}.\mathsf{Enc}(m)$ and $\mathsf{cred} = \mathbf{CredIssue}(\mathsf{epk}, \mathsf{upk}, C_{\mathsf{tp}})$.

  It outputs $(C_{\mathsf{sp}}, T, \pi_{\mathsf{sp}})$.
- $1/0 \leftarrow \mathbf{Link}(T_1, T_2)$. On input two linking tags T_1 and T_2, it outputs 1 if they are linked (computed by the same secret key) or 0 otherwise.
- $1/0 \leftarrow \mathbf{Verify}(\mathsf{epk}, C_{\mathsf{sp}}, T, \pi_{\mathsf{sp}})$. On input the endorser public key epk, the sender's ciphertext C_{sp}, linking tag T, and a proof π_{sp}, it outputs 1 if π_{sp} is a valid zero-knowledge proof, and outputs 0 otherwise.
- $\mathsf{upk}_s \leftarrow \mathbf{Decrypt}(\mathsf{ask}_{\mathsf{sp}}, C_{\mathsf{sp}})$. On input the auditor secret key $\mathsf{ask}_{\mathsf{rp}}$ and a ciphertext C_{sp}, it outputs the decrypted public key upk_s.

Security Model. We first 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.

The security properties of a sender privacy protocol are formalized as follows.

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

$$\Pr\left[\begin{array}{l}\mathbf{Verify}(\mathsf{epk}, C^*_{\mathsf{sp}},\\ T^*, \pi^*_{\mathsf{sp}}) = 1\end{array} : \begin{array}{l}(\mathsf{param}, \mathsf{ask}_{\mathsf{sp}}) \leftarrow \mathbf{Setup}(1^\lambda); (\mathsf{esk}, \mathsf{epk}) \leftarrow \mathbf{EndorserKeyGen}();\\ (C^*_{\mathsf{sp}}, T^*, \pi^*_{\mathsf{sp}}) \leftarrow \mathcal{A}^{\mathsf{Issue}_e, \mathsf{Sign}}(\mathsf{param}, \mathsf{ask}_{\mathsf{sp}}, \mathsf{epk})\end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the issue and sign oracle are defined as below:

- $\mathsf{Issue}_e(\mathsf{upk})$*: on input* upk*, it runs as the endorser of the* **CredIssue***(*epk*,* upk*) protocol with private input* esk *and it outputs* cred*.*
- $\mathsf{Sign}(\mathsf{upk}, \mathsf{usk}, m, r)$*: 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* $(C_{\mathsf{sp}}, T, \pi_{\mathsf{sp}}) \leftarrow \mathbf{CredSign}(\mathsf{usk}, \mathsf{upk}, \mathsf{cred}, m, r)$*.*

We additionally require that $\mathsf{upk}^* \leftarrow \mathbf{Decrypt}(\mathsf{ask}_{sp}, C_{\mathsf{sp}})$ *and* upk^* *is never queried to the* Issue_e *oracle, and* $(C^*_{\mathsf{sp}}, T^*, \pi^*_{\mathsf{sp}})$ *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:*

$$\Pr\left[\begin{array}{l}\mathbf{Verify}(\mathsf{epk}, C^*_{\mathsf{sp}},\\ T^*, \pi^*_{\mathsf{sp}}) = 1\end{array} : \begin{array}{l}(\mathsf{param}, \mathsf{ask}_{\mathsf{sp}}) \leftarrow \mathbf{Setup}(1^\lambda); (\mathsf{esk}, \mathsf{epk}) \leftarrow \mathbf{EndorserKeyGen}();\\ (C^*_{\mathsf{sp}}, T^*, \pi^*_{\mathsf{sp}}) \leftarrow \mathcal{A}^{\mathsf{KeyGen}, \mathsf{Corrupt}, \mathsf{Issue}_u, \mathsf{Sign}}(\mathsf{param}, \mathsf{ask}_{\mathsf{sp}}, \mathsf{epk}, \mathsf{esk})\end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the issue and sign oracle are defined as below:

- $\mathsf{Issue}_u(\mathsf{upk})$*: on input* upk*, it runs as the user of the* **CredIssue***(*epk*,* upk*) protocol with private input* usk*, where* $(\mathsf{usk}, \mathsf{upk}) \in \mathcal{L}$*.*
- $\mathsf{Sign}(\mathsf{upk}, \mathsf{cred}, m, r)$*: 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* $(\mathsf{usk}, \mathsf{upk}) \in \mathcal{L}$*. Then it outputs* $(C_{\mathsf{sp}}, T, \pi_{\mathsf{sp}}) \leftarrow \mathbf{CredSign}(\mathsf{usk}, \mathsf{upk}, \mathsf{cred}, m, r)$*.*

We additionally require that $\mathsf{upk}^* \leftarrow \mathbf{Decrypt}(\mathsf{ask}_{sp}, C_{\mathsf{sp}})$ *and* upk^* *is the output of the* KeyGen *oracle but is never queried to the* Corrupt *oracle, and* $(C^*_{\mathsf{sp}}, T^*, \pi^*_{\mathsf{sp}})$ *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:*

$$\Pr\left[b' = b : \begin{array}{l}(\mathsf{param}, \mathsf{ask}_{\mathsf{rp}}) \leftarrow \mathbf{Setup}(1^\lambda); (\mathsf{esk}, \mathsf{epk}) \leftarrow \mathbf{EndorserKeyGen}(); b \leftarrow \{0,1\};\\ (\{\mathsf{upk}^*_i, \mathsf{cred}^*_i, m^*_i, r^*_i\}_{i\in[0,1]}) \leftarrow \mathcal{A}_1^{\mathsf{KeyGen}, \mathsf{Corrupt}}(\mathsf{param}, \mathsf{epk}, \mathsf{esk});\\ (C^*_{\mathsf{sp}}, T^*, \pi^*_{\mathsf{sp}}) \leftarrow \mathbf{CredSign}(\mathsf{usk}^*_b, \mathsf{upk}^*_b, \mathsf{cred}^*_b, m^*_b, r^*_b),\\ b' \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}, \mathsf{Corrupt}}(\mathsf{param}, \mathsf{esk}, \mathsf{epk}, C^*_{\mathsf{sp}}, T^*, \pi^*_{\mathsf{sp}})\end{array}\right] \leq \mathsf{negl}(\lambda).$$

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

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 [5], since the underlying credential is signed by Boneh-Boyen signature [4], which can be modified to sign on multiple committed values [1]. For the second requirement, we use the tag structure used in most linkable ring signature schemes.

There are two possible constructions: the Tx amount is in plaintext or in ciphertext. In the UTXO model, transaction privacy is required in order to protect sender privacy (otherwise, the attacker can use the Tx amount to link past transactions). In the account-based model, transaction privacy may or may not be needed in the blockchain. 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 $\mathsf{ask}_{\mathsf{sp}}$ in $\mathbb{Z}_p$ and outputs its public key $h_{\mathsf{sp}} = g^{\mathsf{ask}_{\mathsf{sp}}}$. It outputs the public parameters $\mathsf{param} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, g, g_1, g_2, u_1, h_s, f, h_{\mathsf{tp}}, h_{\mathsf{sp}}, \hat{g}_2, H)$.

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

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

- $\mathsf{CredIssue}$. 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 $C = g^m h_{\mathsf{tp}}^{r_{\mathsf{tp}}}$).

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 $\mathsf{esk} = \alpha$ to compute: $A = (h_s \cdot g_1^v \cdot C \cdot Y_1)^{\frac{1}{\alpha+z}}$. The endorser returns the credential $\mathsf{cred} = (A, v, z)$ to the user.

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

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

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

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

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

[9] This 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.

		Sender Privacy	Recipient Privacy	Tx Privacy	Audit-ability	Authenti-cation	Tx Overhead (bytes)	Sender Running Time	Verifier Running Time
Public blockchain	Monero	◔	●	●			12704	300ms	300ms
	Zcash	●	●	●			576	120s	10ms
Consortium blockchain	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.

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

(c) **ZKResponse:** It computes:

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

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

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

1. **ZKReconstruct:** It computes:

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

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

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 $(C_{\mathsf{cred}}, B_{\mathsf{cred}})$ and $\mathsf{ask}_{\mathsf{sp}}$, it computes $Y' = C_{\mathsf{cred}} / B_{\mathsf{cred}}^{\mathsf{ask}_{\mathsf{sp}}}$.

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

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

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 PAChain's transaction privacy, 64-bit of transaction amount implies that the range $R = 2^{64}$. We can take $\mathfrak{u} = 2^{16} = 65536$, $\ell = 4$. 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 bytes [10]. 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 $R_{tx,r}$ 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 (> 3GB). 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 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.

[10] A 64-bit range proof by the recent Bulletproof [7] is about 800 bytes.

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 in 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 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.

References

1. Au, M.H., Susilo, W., Mu, Y.: Constant-size dynamic k-taa. In: Prisco, R.D., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 111–125. Springer (2006)
2. Bellare, M., Goldwasser, S.: Verifiable partial key escrow. In: Graveman, R., Janson, P.A., Neuman, C., Gong, L. (eds.) CCS '97,. pp. 78–91. ACM (1997)
3. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: IEEE SP 2014. pp. 459–474. IEEE Computer Society (2014)
4. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer (2004)
5. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M.K. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer (2004)
6. Boudot, F.: Efficient proofs that a committed number lies in an interval. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 431–444. Springer (2000)
7. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: IEEE SP 2018. pp. 315–334. IEEE (2018)
8. Camenisch, J., Chaabouni, R., Shelat, A.: Efficient protocols for set membership and range proofs. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 234–252. Springer (2008)
9. Camenisch, J., Mödersheim, S., Sommer, D.: A formal model of identity mixer. In: Kowalewski, S., Roveri, M. (eds.) FMICS 2010. LNCS, vol. 6371, pp. 198–214. Springer (2010)

10. Garman, C., Green, M., Miers, I.: Accountable privacy for decentralized anonymous payments. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 81–98. Springer (2016)
11. Hyperledger Fabric: Architecture explained (2017), http://hyperledger-fabric.readthedocs.io/en/latest/arch-deep-dive.html
12. Li, W., Sforzin, A., Fedorov, S., Karame, G.O.: Towards scalable and private industrial blockchains. In: BCC '17. pp. 9–14. ACM (2017)
13. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2009), https://bitcoin.org/bitcoin.pdf
14. Noether, S.: Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098 (2015), http://eprint.iacr.org/
15. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT '99. LNCS, vol. 1592, pp. 223–238. Springer (1999)
16. Ruffing, T., Moreno-Sanchez, P., Kate, A.: Coinshuffle: Practical decentralized coin mixing for bitcoin. In: Kutylowski, M., Vaidya, J. (eds.) ESORICS 2014. LNCS, vol. 8713, pp. 345–364. Springer (2014)
17. Sun, S., Au, M.H., Liu, J.K., Yuen, T.H.: Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) ESORICS 2017. LNCS, vol. 10493, pp. 456–474. Springer (2017)
18. Wüst, K., Kostiainen, K., Capkun, V., Capkun, S.: Prcash: Centrally-issued digital currency with privacy and regulation. Cryptology ePrint Archive, Report 2018/412 (2018), https://eprint.iacr.org/2018/412. In FC 2019.
19. Yuen, T.H.: Pachain: Private, authenticated and auditable consortium blockchain. In: Mu, Y., Deng, R.H., Huang, X. (eds.) CANS 2019. LNCS, vol. 11829, pp. 214–234. Springer (2019)

A Building Our PAChain

We show how to integrate our cryptographic building blocks with the Hyperledger fabric framework. We first describe some modifications for combining our cryptographic building blocks. After that, we demonstrate the system setup followed by the transaction flow in our privacy-preserving auditable consortium blockchain, PAChain.

A.1 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.

Combining Transaction Privacy with Sender Privacy A small change has to be made for $\mathsf{T.TxPrivacyVerify}$ and $\mathsf{S.CredSign}$. It is because $C_{\mathsf{in},i} = g^{M_{\mathsf{in},i}} h_{\mathsf{tp}}^{r_{\mathsf{in},i}}$ and $B_{\mathsf{in},i} = g^{r_{\mathsf{in},i}}$ used for verification in $\mathsf{T.TxPrivacyVerify}$, 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 $\mathsf{S.CredSign}$:

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

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

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

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

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 $\mathsf{S.Decrypt}$ and gets a one-time public key. From this one-time public key, the auditor retrieves the UTXO, and runs $\mathsf{R.Decrypt}$ to obtain the long-term public key of the sender.

A.2 System Setup and User Registration

This section outlines the system setup and user registration in PAChain. In order to simplify our description, we use the Hyperledger Fabric framework [11] and add our privacy-preserving and auditable modulus to it. 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* $\mathsf{param}_T \leftarrow \mathsf{T.Setup}(1^\lambda)$*,* $\mathsf{param}_R \leftarrow \mathsf{R.Setup}(1^\lambda)$ *and* $\mathsf{param}_S \leftarrow \mathsf{S.Setup}(1^\lambda)$*. Suppose the system parameter* $\mathsf{param} = (\mathsf{param}_T, \mathsf{param}_S, \mathsf{param}_R)$ *is known to all users in the system.*

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

CA Setup. The CA[11] obtains a key pair by running $(\mathsf{cask}, \mathsf{capk}) \leftarrow \mathsf{R.CAKeyGen}()$.

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

A.3 Transaction Flow

We outline the modified transactional mechanics that take place during a standard transaction. As in [11], the system hosts smart contracts (called *chaincode*) that comprise the application logic of the system. Chaincode holds state and ledger data, and transactions are operations invoked on the chaincode. Our protocols are mainly added to the client side (TxPropose) and endorser side (Endorse).

Tx Propose. Consider the case that Alice want to sends $\$M_{\mathsf{out},j}$ to user $\mathsf{upk}_{\mathsf{out},j}$ for $j \in [1, n']$. Suppose that user $\mathsf{upk}_{\mathsf{out},j}$ is registered to the CA with certificate $\mathsf{cert}_{\mathsf{out},j}$ 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 $M_{\mathsf{in},\mathsf{i}}$ for $i \in [1, n]$, where $\sum_{i\in[1,n]} M_{\mathsf{in},\mathsf{i}} = \sum_{j\in[1,n']} M_{\mathsf{out},j}$. *Then for each input amount* $M_{\mathsf{in},i}$*, Alice should have the corresponding one time public key* $\mathsf{otpk}_{A,i}$*, commitment* $C_{\mathsf{in},i}$*, commitment randomness* $r_{\mathsf{in},i}$*, credential* $\mathsf{cred}_{\mathsf{in},i}$ *(issued by endorser* E_i*) and block randomness* $R_{\mathsf{in},i}$*.* Alice prepares a transaction message `tx` which includes the following fields:

1. `clientID`. *It includes the endorsers' IDs* E_i *for* $i \in [1, n]$. [12]
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 ⟨`source`, `metadata`, `policies`⟩. `source` denotes the source code of the chaincode and `policies` includes the endorsement policy. `metadata` *includes the following:*
   (a) RecipientPK. For each recipient $\mathsf{upk}_{\mathsf{out},j}$*, Alice runs the zero-knowledge protocol for recipient privacy, without using the part for* otsk*:* $(\cdot, \mathsf{otpk}_{\mathsf{out},j}, R_{tx,\mathsf{out}}, C_{\mathsf{rp},j}, \pi_{\mathsf{rp},j}) \leftarrow \mathsf{R.RecPrivacySpend}(\mathsf{capk}, \cdot, \cdot, \cdot, \mathsf{upk}_{\mathsf{out},j}, \mathsf{cert}_{\mathsf{out},j})$.
   It is because the possession of otsk *will later be shown by* $\mathsf{S.CredSign}$*. Note that for all* $j \in [1, n']$*, Alice generates the same* $R_{tx,\mathsf{out}}$ *output during the computation. The Recipient PK field includes* $(\mathsf{otpk}_{\mathsf{out},j}, C_{\mathsf{rp},j}, \pi_{\mathsf{rp},j})$ *for all* $j \in [1, n']$.
   (b) Output. Alice generates the output field $(\{C_{\mathsf{out},j}\}_{j\in[1,n']}, \pi_{\mathsf{tp}})$*, where:* $(\{C_{\mathsf{out},j}, r_{\mathsf{out},j}\}_{j\in[1,n']}, \pi_{\mathsf{tp}}) \leftarrow \mathsf{T.TxPrivacySpend}\ (\{M_{\mathsf{out},j}\}_{j\in[1,n']}, \{C_{\mathsf{in},i}, M_{\mathsf{in},i}, r_{\mathsf{in},i}\}_{i\in[1,n]})$.
   (c) Info. Alice puts the block randomness $R_{tx,\mathsf{out}}$ *in the information field* [13]*.*

[11] CA is called membership service provider (MSP) in Fabric.

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

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

5. clientSig. *Alice obtains the one-time secret keys of her n inputs by* $x'_{\mathsf{in},i} \leftarrow \mathsf{R.OneTimeSkGen}(\mathsf{otpk}_{A,i}, R_{\mathsf{in},i}, \mathsf{usk}_A)$ *for* $i \in [1,n]$. *Alice runs* $(C_{\mathsf{sp},i}, T_i, \pi_{\mathsf{sp},i}) \leftarrow \mathsf{S.CredSign}'(x'_{\mathsf{in},i}, \mathsf{otpk}_{A,i}, \mathsf{cred}_{\mathsf{in},i}, M_{\mathsf{in},i}, r_{\mathsf{in},i})$ *for all* $i \in [1,n]$: *Finally, clientSig includes* $(C_{\mathsf{sp},i}, T_i, \pi_{\mathsf{sp},i})$ *for all* $i \in [1,n]$.

Alice submits a proposal message $\langle$`PROPOSE`, `tx`, [`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 Hash(`tx`).

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

1. the transaction proposal `tx` is well-formed,
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* $i \in [1,n]$*:*

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

   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* $\mathsf{S.Link}(T^*, T')$ *for all* $(\cdot, T^*, \cdot) \in$ `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:* $1/0 \leftarrow \mathsf{B.Verify}(\mathsf{capk}, \cdot, \cdot, \mathsf{otpk}_{\mathsf{out},j}, R_{tx,\mathsf{out}}, C_{\mathsf{rp},j}, \pi_{\mathsf{rp},j})$. *We ignore the checking of* otsk *in B.Verify since it is checked in S.Verify.*
6. *the transaction amount is correct. It runs:*

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

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 $\langle$`TRANSACTION-ENDORSED`, `tid`, `tran-proposal`, `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* $\mathsf{cred}_{\mathsf{out},j} \leftarrow \mathsf{S.CredIssue}(\mathsf{epk}, \mathsf{otpk}_{\mathsf{out},j}, C_{\mathsf{out},j})$ *for all* $j \in [1,n']$, *with private input* esk. [14] *The endorser puts* $\{\mathsf{cred}_{\mathsf{out},j}\}_{j\in[1,n']}$ *into* `txContentBlob`.
- `epSig` is the endorser's signature on `tran-proposal` by using its signing key ssk.

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

Tx Submit. Alice waits until it receives *enough* messages and signatures on $\langle$`TRANSACTION-ENDORSED`, `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, $\langle$`DELIVER`, `seqno`, `prevhash`, `endorsement`$\rangle$ is broadcast to all peers.

Tx Commit. Upon receiving $\langle$`DELIVER`, `seqno`, `prevhash`, `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.

[14] 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}.

B Security Proofs

B.1 Transaction Privacy

We first prove the security of the proof π_{enc}.

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

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

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

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:

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

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 $[0, \mathfrak{u}-1]$. 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. □

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

Then, we can give the security of our transaction privacy protocol. 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.

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

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

$$h_{\mathsf{tp}}^{z_{\mathsf{tp}}} (\prod_{i=1}^{n} C'_{\mathsf{in},i} / \prod_{j=1}^{n'} C'_{\mathsf{out},j})^{\tilde{c}'} = h_{\mathsf{tp}}^{z_{\mathsf{tp},2}} (\prod_{i=1}^{n} C'_{\mathsf{in},i} / \prod_{j=1}^{n'} C'_{\mathsf{out},j})^{\tilde{c}'_2},$$
$$g^{z_{\mathsf{tp}}} (\prod_{i=1}^{n} B'_{\mathsf{in},i} / \prod_{j=1}^{n'} B'_{\mathsf{out},j})^{\tilde{c}'} = g^{z_{\mathsf{tp},2}} (\prod_{i=1}^{n} B'_{\mathsf{in},i} / \prod_{j=1}^{n'} B'_{\mathsf{out},j})^{\tilde{c}'_2},$$

Denote $x^*_{\mathsf{tp}} = \frac{z_{\mathsf{tp}} - z_{\mathsf{tp},2}}{\tilde{c}'_2 - \tilde{c}'}$. Then

$$\prod_{i=1}^{n} C'_{\mathsf{in},i} / \prod_{j=1}^{n'} C'_{\mathsf{out},j} = h_{\mathsf{tp}}^{x^*_{\mathsf{tp}}}, \quad \prod_{i=1}^{n} B'_{\mathsf{in},i} / \prod_{j=1}^{n'} B'_{\mathsf{out},j} = g^{x^*_{\mathsf{tp}}}.$$

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

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

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

B.2 Recipient Privacy

Proof of Soundness.

Proof. The simulator is given the q-SDH problem instance. It randomly picks $w_1, \ldots, w_{q-1}, \iota_1, \iota_2, f', w' \in \mathbb{Z}_p$ and sets $h_2 = g_1^{\prod_{i=1}^{q-1}(\beta + w_i)}$, $g_3 = h_2^{\frac{(w'+\beta)f'-1}{\iota_2}}$, $g_2 = g_3^{\iota_1}$, $\hat{W}_2 = \hat{g}_2^{\beta}$. The rest of param are generated as in **Setup**.

The oracles are simulated as follows:

- $\mathsf{KeyGen}()$: The simulator runs $(\mathsf{usk}, \mathsf{upk}) \leftarrow$ **UserKeyGen**(), stores $(\mathsf{usk}, \mathsf{upk})$ in a list $\mathcal{L}$ (which is initially empty) and outputs upk.
- $\mathsf{Corrupt}(\mathsf{upk})$: On input upk, it searches $(\mathsf{usk}, \mathsf{upk}) \in \mathcal{L}$ and returns usk. It returns $\perp$ if no such key is found.
- $\mathsf{Issue}(\mathsf{upk})$: On i-th distinct input $\mathsf{upk}_i = (Y_{i,1}, Y_{i,2})$, 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 $\mathsf{cert}_i = (F_i, w_i, s_i)$.

  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 $\mathsf{cert}_q = (F', w', s')$.
- $\mathsf{Spend}(\mathsf{upk}_s, \mathsf{otpk}_s, R_{tx,s}, \mathsf{upk}_r, \mathsf{cert}_r)$: For each Spend oracle query to any public key upk_r with certificate cert_r from some public key upk_s corresponding to $(\mathsf{otpk}_s, R_{tx,s})$, it runs as follows: (1) The simulator runs the simulator of the zero-knowledge proof π_{otsk}. (2) It generates $\mathsf{otpk}_r, R_{tx,r}$ and r_{tx} as in the **OneTimePkGen** protocol. (3) It generates $C_{\mathsf{rp}} = (C_{\mathsf{cert}}, B_{\mathsf{cert}})$ as in the protocol. (4) It picks some $\pi_{\mathsf{rp}} = (c, \Theta, z_\omega, z_\tau, z_\rho, z_c, z_\sigma, z_s)$ in the corresponding domains and calculates $R_{\mathsf{cert},1}$, $R_{\mathsf{cert},2}, R_{\mathsf{cert},3}, R_{\mathsf{cert},4}$ as in **ZKReconstruct**. It sets the random oracle $c' = H(\mathsf{CertAuth.mpk}, C_{\mathsf{rp}}, \Theta, R_{\mathsf{cert},1}, R_{\mathsf{cert},2}, R_{\mathsf{cert},3}, R_{\mathsf{cert},4})$. It outputs $\pi_{\mathsf{otsk}}, \mathsf{otpk}_s, \mathsf{otpk}_r, R_{tx,r}, B_{\mathsf{cert}}, C_{\mathsf{cert}}$ 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, $\mathsf{upk}^* = (Y^*_1, \cdot)$ should not be input to both the Issue oracle, and $(\mathsf{upk}^*, \mathsf{otpk}^*, R^*_{tx}, \cdot, \cdot)$ should not be queried to the Spend oracle.

Denote the first Tx output is $(\pi_{\mathsf{otsk}}, \mathsf{otpk}_s, \mathsf{otpk}^*, R^*_{tx}, C^*_{\mathsf{rp}}, \pi^*_{\mathsf{rp}})$ and denote $C^*_{\mathsf{rp}} = (C_{\mathsf{cert}}, B_{\mathsf{cert}})$, $\pi^*_{\mathsf{rp}} = (c, \Theta, z_\omega, z_\tau, z_\rho, z_c, z_\sigma, z_s)$. 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:

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

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

$$\hat{e}((h_2 \cdot C_{\mathsf{cert}})^{\rho^*} g_3^{s^*} \Theta^{-w^*} h_{\mathsf{rp}}^{-\sigma^*}, \hat{g}_2) = \hat{e}(\Theta, \hat{W}_2), \tag{1}$$
$$B_{\mathsf{cert}} = g^{r^*_{\mathsf{cert}}}, \tag{2}$$
$$B_{\mathsf{cert}}^{\rho^*} = g^{\sigma^*}, \tag{3}$$
$$C_{\mathsf{cert}} = \mathsf{otpk}^* \cdot h_{\mathsf{rp}}^{r^*_{\mathsf{cert}}} g_2^{-h^*_{\mathsf{tx}}}. \tag{4}$$

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

$$\hat{e}((h_2 \cdot C_{\mathsf{cert}})^{\rho^*} g_3^{s^*} \Theta^{-w^*} h_{\mathsf{rp}}^{-\rho^* r^*_{\mathsf{cert}}}, \hat{g}_2) = \hat{e}(\Theta, \hat{W}_2),$$
$$\hat{e}(h_2 \cdot \mathsf{otpk}^* \cdot g_2^{-h^*_{\mathsf{tx}}} \cdot g_3^{s^*}, \hat{g}_2) = \hat{e}(\Theta^{\frac{1}{\rho^*}}, \hat{g}_2^{w^*} \hat{W}_2).$$

If the adversary wins the game, it means that he can spend the money of otpk^* by issusing a zero-knowledge proof π^*_{otsk} of $\mathsf{otsk}^* = \log_{g_2} \mathsf{otpk}^*$. Since $(\mathsf{upk}^*, \mathsf{otpk}^*, R^*_{tx}, \cdot, \cdot)$ 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:

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

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

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

If $Z_{-1} \neq 0$, the simulator can compute $A^* = (\Theta^{\frac{1}{\rho^*}} g_1^{-\sum_{i=0}^{q-2} Z_i \beta^i})^{\frac{1}{Z_{-1}}}$. 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 $\mathsf{upk_i} = (Y_{i,1}, \cdot)$ 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 $\hat{e}(h_2 \cdot g_2^{\mathsf{otsk}^* - h^*_{tx}} \cdot g_3^{s^*}, \hat{g}_2) = \hat{e}(\Theta^{\frac{1}{\rho^*}}, \hat{g}_2^{w^*} \hat{W}_2)$. If $\Theta^{\frac{1}{\rho^*}} = F_i$, then we have $s^* = s_i$ and $\mathsf{otsk}^* - h^*_{tx} = x_{i,1}$ if the DL assumption holds between g_2 and g_3. Recall that the decryption of $(C_{\mathsf{cert}}, B_{\mathsf{cert}})$ gives Y_1^* if the adversary wins the game. By Eq. 2 and 4, we have $Y_1^* = \mathsf{otpk}^* g_2^{-h^*_{\mathsf{tx}}}$. By the soundness of zero-knowledge proof π^*_{otsk}, we have $\mathsf{otsk}^* = \log_{g_2} \mathsf{otpk}^*$. 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 $\Theta^{\frac{1}{\rho^*}} \neq F_i$ for some i, we show that the simulator can also solve the q-SDH. With probability $1/q$, $w^* = w'$. Then denote $F^* = \Theta^{\frac{1}{\rho^*}}$ and $y^* = \iota_1(\mathsf{otsk}^* - h^*_{tx}) + s^*$:

$$F^{*\beta+w^*} = h_2 g_2^{\mathsf{otsk}^* - h^*_{tx}} \cdot g_3^{s^*} = h_2 g_3^{\iota_1(\mathsf{otsk}^* - h^*_{tx}) + s^*} = h_2 g_3^{y^*} = h_2^{1 + y^* \cdot \frac{(w'+\beta)f'-1}{\iota_2}},$$
$$F^* = h_2^{\frac{\iota_2 - y^* + y^* f'(\beta+w')}{\iota_2(\beta+w')}} = h_2^{\frac{\iota_2 - y^*}{\iota_2(\beta+w')} + \frac{y^* f'}{\iota_2}},$$
$$(F^* h_2^{\frac{-y^* f'}{\iota_2}})^{\frac{\iota_2}{\iota_2 - y^*}} = h_2^{\frac{1}{\beta+w'}} = g_1^{\frac{\prod_{i=1}^{q-1}(\beta+w_i)}{\beta+w'}}$$

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 $Z'_{-1}, Z'_0, \ldots, Z'_{q-2} \in \mathbb{Z}_p$ 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. □

Proof of Anonymity.

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

- π^*_{otsk} is not related to the recipient.
- $C^*_{\mathsf{rp}} = (C_{\mathsf{cert}}, B_{\mathsf{cert}})$ 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 $\mathsf{otpk}^*_r = Y_1 g_2^{H(Y_2^{r_{tx,r}})}$ 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), $\mathsf{otpk}^*_r, R^*_{tx,r}$ 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 $R_{\mathsf{cert},1}, R_{\mathsf{cert},2}, R_{\mathsf{cert},3}, R_{\mathsf{cert},4}$ as in **ZKReconstruct**. It sets the random oracle $c' = H(\mathsf{CertAuth.mpk}, C^*_{\mathsf{rp}}, \Theta, R_{\mathsf{cert},1}, R_{\mathsf{cert},2}, R_{\mathsf{cert},3}, R_{\mathsf{cert},4})$. 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. □

B.3 Sender Privacy

Proof of Soundness.

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

The oracles are simulated as follows:

- $\mathsf{Issue}_e(\mathsf{upk})$: On k-th distinct input $\mathsf{upk}_k = (Y_{k,1}, \cdot)$ 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 $C_k = g^{m_k} h_{\mathsf{tp}}^{r_{\mathsf{tp},k}}$. 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:

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

  The oracle returns the certificate $\mathsf{cert}_k = (A_k, z_k, v_k)$.
  For the q-th query, it sets $v' = \iota_2 - \iota_3 m_q - \iota_4 r_{\mathsf{tp},q} - \iota_1 x_{q,1}$ and $A' = h_2^{a'}$. The oracle returns the certificate $\mathsf{cert}_q = (A', z', v')$.
- $\mathsf{Sign}(\mathsf{upk}, \mathsf{usk}, m, r)$: On input a sender's public key upk, secret key usk, the input transaction amount m encrypted with randomness r, it firstly denote $\mathsf{upk} = (Y'_{\mathsf{in}}, \cdot)$. 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 $R_{\mathsf{cred},1}, R_{\mathsf{cred},2}, R_{\mathsf{cred},3}, R_{\mathsf{cred},4}, R_{\mathsf{cred},5}, R_{\mathsf{cred},6}$ as in ZKReconstruct. It sets $c = H(\mathsf{CredAuth.mpk}, C_{\mathsf{sp}}, T, S, \Xi, R_{\mathsf{cred},1}, R_{\mathsf{cred},2}, R_{\mathsf{cred},3}, R_{\mathsf{cred},4}, R_{\mathsf{cred},5}, R_{\mathsf{cred},6})$ in the random oracle model. Denote $\pi_{\mathsf{sp}} = (c, S, \Xi, z_k, z_a, z_z, z_b, z_v, z_m, z_r, z_\psi)$. Then it outputs $(C_{\mathsf{sp}}, T, \pi_{\mathsf{sp}})$.

In the challenge phase, the adversary outputs $(C^*_{\mathsf{sp}} = (C^*_{\mathsf{cred}}, B^*_{\mathsf{cred}}), T^*, \pi^*_{\mathsf{sp}})$. 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 $(C^*_{\mathsf{sp}}, T^*, \pi^*_{\mathsf{sp}})$ should not be the output to the Sign oracle.

For the same $B^*_{\mathsf{cred}}, C^*_{\mathsf{cred}}, T^*$ and commitment $(S^*, \Xi^*, R^*_{\mathsf{cred},1}, R^*_{\mathsf{cred},2}, R^*_{\mathsf{cred},3}, R^*_{\mathsf{cred},4}, R^*_{\mathsf{cred},5}, R^*_{\mathsf{cred},6})$, 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 $(z'_k, z'_a, z'_z, z'_b, z'_v, z'_m, z'_r, z'_\psi)$. We have:

$$\begin{aligned}
\hat{e}(u_1^{z_b} S^{*-z_z} g_1^{z_v} g^{z_m} h_{\mathsf{tp}}^{z_r} g_2^{z_k}, \hat{g}_2) \cdot \hat{e}(u_1^{z_a}, W_{\mathsf{sp}}) \cdot [\hat{e}(h_s, \hat{g}_2) \cdot \hat{e}(S^{*-1}, W_{\mathsf{sp}})]^c \\
= \hat{e}(u_1^{z_b} S^{*-z'_z} g_1^{z'_v} g^{z'_m} h_{\mathsf{tp}}^{z'_r} g_2^{z'_k}, \hat{g}_2) \cdot \hat{e}(u_1^{z'_a}, W_{\mathsf{sp}}) \cdot [\hat{e}(h_s, \hat{g}_2) \cdot \hat{e}(S^{*-1}, W_{\mathsf{sp}})]^{c'}, \\
g_1^{z_a} \cdot \Xi^{*-c} = g_1^{z'_a} \cdot \Xi^{*-c'}, \\
\Xi^{*z_x} g_1^{-z_b} = \Xi^{*z'_x} g_1^{-z'_b}, \\
g^{z_\psi} \cdot {B^*_{\mathsf{cred}}}^{-c} = g^{z'_\psi} \cdot {B^*_{\mathsf{cred}}}^{-c'}, \\
g_2^{z_k} h_{\mathsf{sp}}^{z_\psi} {C^*_{\mathsf{cred}}}^{-c} = g_2^{z'_k} h_{\mathsf{sp}}^{z'_\psi} {C^*_{\mathsf{cred}}}^{-c'}, \\
f^{z_k} \cdot T^{*-c} = f^{z'_k} \cdot T^{*-c'}.
\end{aligned}$$

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

Then we have:

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

From the second and the third equation, we have $b^* = a^* z^*$. From the fourth and the fifth equation, we can see that $(C^*_{\mathsf{cred}}, B^*_{\mathsf{cred}})$ 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:

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

Denote the polynomial $\frac{(1 + \frac{(z'+\alpha)a'-1}{\iota_2}(v^* + \iota_3 m^* + \iota_4 r^*_{\mathsf{in}} + \iota_1 x^*)) \prod_{j=1}^{q-1} (\alpha + z_j)}{\alpha + z^*} = \sum_{j=0}^{q-2} Z_j \alpha^j + \frac{Z_{-1}}{\alpha + z^*}$ for some coefficients $Z_{-1}, Z_0, \ldots, Z_{q-2} \in \mathbb{Z}_p$ 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 $\hat{e}(h_s g_1^{v_k} g^{m_k} h_{\mathsf{tp}}^{r_k} g_2^{x_k}, \hat{g}_2) = \hat{e}(A_k, \hat{g}_2^{z^*} W_{\mathsf{sp}})$. Recall that $\hat{e}(h_s g_1^{v^*} g^{m^*} h_{\mathsf{tp}}^{r^*_{\mathsf{in}}} g_2^{x^*}, \hat{g}_2) = \hat{e}(S^* u_1^{-a^*}, \hat{g}_2^{z^*} W_{\mathsf{sp}})$. 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 $(C^*_{\mathsf{cred}}, B^*_{\mathsf{cred}})$ 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 $y^* = v^* + \iota_3 m^* + \iota_4 r^*_{\mathsf{in}} + \iota_1 x^*$:

$$A^{*\alpha+z^*} = h_s g_1^{v^*} g^{m^*} h_{\mathsf{tp}}^{r^*_{\mathsf{in}}} g_2^{x^*} = h_s g_1^{v^*+\iota_3 m^*+\iota_4 r^*_{\mathsf{in}}+\iota_1 x^*} = h_s g_1^{y^*} = h_s^{1+y^* \cdot \frac{(z'+\alpha)a'-1}{\iota_2}},$$

$$A^* = h_s^{\frac{\iota_2 - y^* + y^* a'(\alpha+z')}{\iota_2(\alpha+z')}} = h_2^{\frac{\iota_2 - y^*}{\iota_2(\alpha+z')} + \frac{y^* a'}{\iota_2}},$$

$$(A^* h_2^{\frac{-y^* a'}{\iota_2}})^{\frac{\iota_2}{\iota_2 - y^*}} = h_2^{\frac{1}{\alpha+z'}} = g_0^{\frac{\prod_{i=1}^{q-1}(\alpha+z_i)}{\alpha+z'}}$$

Denote the polynomial $\frac{\prod_{i=1}^{q-1}(\alpha+z_i)}{\alpha+z'} = \sum_{i=0}^{q-2} Z'_i \alpha^i + \frac{Z'_{-1}}{\alpha+z'}$ for some coefficients $Z'_{-1}, Z'_0, \ldots, Z'_{q-2} \in \mathbb{Z}_p$ 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. □

Proof of Unforgeability.

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 param, $\mathsf{ask}_{\mathsf{sp}}$, epk, esk are honestly generated and are given to the adversary.

The oracles are simulated as follows:

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

In the challenge phase, the adversary outputs $(C^*_{\mathsf{sp}} = (C^*_{\mathsf{cred}}, B^*_{\mathsf{cred}}), T^*, \pi^*_{\mathsf{sp}})$. 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 $\mathsf{Corrupt}$ oracle and $(C^*_{\mathsf{sp}}, T^*, \pi^*_{\mathsf{sp}})$ 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

$$C^*_{\mathsf{cred}} = g_2^{x^*} h_{\mathsf{sp}}^{r^*_{\mathsf{cred}}}, \quad B^*_{\mathsf{cred}} = g^{r^*_{\mathsf{cred}}}.$$

Suppose there are q_c queries to the KeyGen oracle. With probability $1/q_c$, $Y^*_{\mathsf{in}} = y_0$. In this case, the simulator can return x^* as the solution to the DL problem. □

Proof of Anonymity.

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 $g = g_0, g_2 = g_0^{\delta_2}, h_{\mathsf{sp}} = g_0^a, f = g_0^{a\delta_1}$. The rest of param, epk, esk are honestly generated and are given to the adversary.

- $\mathsf{KeyGen}()$: The simulator runs $(\mathsf{usk}, \mathsf{upk}) \leftarrow \mathbf{UserKeyGen}()$, stores $(\mathsf{usk}, \mathsf{upk})$ 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 $\mathsf{upk}^* = (g_0^b, Y_2)$ and puts $(\perp, \mathsf{upk}^*)$ in $\mathcal{L}$.
- $\mathsf{Corrupt}(\mathsf{upk})$: On input upk, it searches $(\mathsf{usk}, \mathsf{upk}) \in \mathcal{L}$ and returns usk. It declares failure and exits if $\mathsf{upk} = \mathsf{upk}^*$.

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

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

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 $R_{\mathsf{cred},1}^*, R_{\mathsf{cred},2}^*, R_{\mathsf{cred},3}^*, R_{\mathsf{cred},4}^*, R_{\mathsf{cred},5}^*, R_{\mathsf{cred},6}^*$ as in ZKReconstruct. It sets $c^* = H(\mathsf{CredAuth.mpk}, C_{\mathsf{sp}}^*, T^*, S^*, \Xi^*, R_{\mathsf{cred},1}^*, R_{\mathsf{cred},2}^*, R_{\mathsf{cred},3}^*, R_{\mathsf{cred},4}^*, R_{\mathsf{cred},5}^*, R_{\mathsf{cred},6}^*)$ in the random oracle model. Denote $\pi_{\mathsf{sp}}^* = (c^*, S^*, \Xi^*, z_k^*, z_a^*, z_z^*, z_b^*, z_v^*, z_m^*, z_r^*, z_\psi^*)$. Then it returns $(C_{\mathsf{sp}}^*, T^*, \pi_{\mathsf{sp}}^*)$ 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. □