

Privacy Protection for the Blockchains with Account and Multi-Asset Model

Donghui Ding^{1,2}, Kang Li³, Zhongcheng Li¹, Jun Li¹, and Yi Sun¹

¹ Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
`{dingdonghui,zcli,liju,sunyi}@ict.ac.cn`

² University of Chinese Academy of Sciences, Beijing, China

³ Ant Financial Services Group, Hangzhou, China
`lightningli.lk@antfin.com`

Abstract. The blockchain technology has been applied to wide areas. However, the open and transparent properties of the blockchain pose serious challenges to users' privacy. Among all the schemes for the privacy protection, the zero-knowledge proof algorithm conceals most of the private information in a transaction, while participants of the blockchain can validate this transaction without the private information. However, current schemes are only aimed at blockchains with the UTXO model, and only one type of assets circulate on these blockchains.

Based on the zero-knowledge proof algorithm, this paper proposes a privacy protection scheme for blockchains that use the account and multi-asset model. We design the transaction structure, anonymous addresses and anonymous asset metadata, and also propose the methods of the asset transfer and double-spending detection. The zk-SNARKs algorithm is used to generate and to verify the zero-knowledge proof. And finally, we use experiments to evaluate our scheme.

Keywords: Blockchain · Privacy protection · Zero-knowledge proof algorithm · Account and multi-asset model.

1 Introduction

In 2008, Satoshi Nakamoto proposed Bitcoin, which for the first time used the blockchain technology to implement an electronic currency system without the involvement of third parties. Blockchains enable participants to maintain a transparent ledger, and to safely transfer assets by cryptography. Nowadays, blockchains have been used in areas such as banks and supply chains.

However, blockchains pose challenges to users' privacy. All transactions are recorded into the blockchain, and each participant maintains a complete replica of the blockchain. The private information in a transaction is thus exposed to all participants. Research has analyzed users' behaviour by tracking the transactions, and even mapped the addresses in transactions into real entities [5].

Aimed at this problem, different schemes are proposed. Among all schemes, the zero-knowledge proof algorithm conceals most private information in a transaction. Each participant can validate the transaction without such information.

ZCash [7] is the most mature blockchain that uses the zero-knowledge proof algorithm. This blockchain is built on the UTXO model, and only a single type of assets circulates on ZCash blockchain. When assets are spent on ZCash, a transaction with multiple inputs and outputs is constructed. These inputs consume assets in previous transactions' outputs, while assets in the newly-generated outputs can be used by future transactions. Such transactions can realize the conversion between transparent assets and anonymous assets at a 1:1 rate.

However, blockchains with the account and multi-asset model have appeared in recent years. The state of a blockchain with this model is organized by a series of *accounts*. Assets spent by a transaction are from the sender's account instead of other transactions' outputs. Therefore, the transaction structure of this model is different from that of the UTXO model. Since the scheme based on the zero-knowledge proof algorithm is tightly coupled with the transaction structure, the scheme of ZCash is incompatible with the account and multi-asset model.

In addition, this model contains different assets with various types. So it is necessary to ensure the conversion between the same type of anonymous and transparent assets, and the transfer of the same type of anonymous assets.

Based on the zero-knowledge proof algorithm, we design a privacy protection scheme for blockchains that use the account and multi-asset model. The sender, receiver, and amount in a transaction is concealed, and all participants can validate the transactions without this information.

2 Related Work

Various privacy protection schemes have been proposed. Dashcoin[3] cuts off the relation between the senders and receivers, but causes potential centralization. Monero [8] conceals senders' addresses by the ring signature, but the transaction amounts are still exposed to the public. The homomorphic encryption is also used to protect users' privacy, but it suffers from the low efficiency.

Recently, the zero-knowledge proof algorithm has attracted increasing attentions, by which one party can prove to the other party that a statement is correct without revealing other extra information. In all congeneric algorithms, the zk-SNARKs algorithm is the most practical one because it uses a succinct character string rather than complex interactions to finish the proof procedure. This character string is defined as "zero-knowledge proof". Zerocoin[6] firstly applied the zero-knowledge proof algorithm to the blockchain. ZCash improved Zerocoin, and conceals most private information including the senders, receivers and amounts. However, ZCash is only fit for the UTXO and single-asset model.

3 Verification Rules for Shielded Transactions

A shielded transaction realizes the conversion between transparent assets and anonymous assets, or transfers anonymous assets between different accounts. Shielded transactions conceal information including the senders, receivers and

amounts, and thus protect users' privacy. All blockchain nodes run the verification rules for shielded transactions, and reach consensus based on these rules.

Verification rules are required to verify the existence and ownership of the assets spent by the transaction senders. Furthermore, the rules should ensure the safe transfer of assets, and detect double-spending behaviours. Aimed at these issues, the rules include the structure of the shielded transaction, the anonymous address of the account, the metadata of the anonymous asset, the transfer of anonymous assets, and the anti double-spending mechanism.

3.1 Structure of the Shielded Transaction

The structure of a shielded transaction in our system contains following fields.

nonce The number of transactions sent by the account of the sender. It is used to prevent replay attacks.

sender The account address of the sender. When the transaction is used to transfer anonymous assets, this field is set to zero.

receiver The account address of the receiver. When the transaction is used to transfer anonymous assets, this field is set to zero.

assetID The type of the assets to be transferred.

in_value The amount of transparent assets spent by the sender.

out_value The amount of transparent assets received by the receiver.

vAggreSeg The description of an **AggreSeg** (aggregated segment), which is the basic unit that describes how anonymous assets are generated or transferred in a shielded transaction.

nAggreSeg The number of vAggreSegs in a shielded transaction.

aggreSegSig In order to ensure each vAggreSeg is correlated with a shielded transaction and can not be replayed by attackers, the sender generates a key pair for each shielded transaction, and signs this transaction except **txSig**.

aggreSegPubKey The temporary public key used to verify **aggreSegSig**.

txSig The signature of the whole transaction.⁴

In our system, the vAggreSeg is represented by a C++ class **AggreSegDesc**. The structure of the vAggreSeg is shown as following.

vpub_old and **vpub_new** Transparent assets this vAggreSeg spends and generates. Both of these fields can be set to zero.

anchor A Merkle tree root. It is used to verify the existence of the anonymous asset spent by this vAggreSeg. More details are discussed in the section 3.3.

assetId The type of the anonymous asset spent by this vAggreSeg.

sequence A serial number assigned to the asset to be spent. It can prevent double-spending behaviours. More details are discussed in the section 3.5.

commitments[2] The commitments for newly-generated anonymous assets. The design of the commitments is discussed in the section 3.3.

ephemeralKey A temporary public key for Curve25519, which is used to negotiate a symmetric key between the sender and receiver.

⁴ It is used to judge the validity of the account address of the sender, and is concealed when the shielded transaction is used to transfer anonymous assets.

randomSeed A 256-bit random value, which is used to calculate the sequences of newly-generated assets. More details are discussed in the section 3.5.

zkproof The zero-knowledge proof of this vAggreSeg.

encCiphertexts [2] The encrypted metadata of the newly-generated assets. More details are discussed in the section 3.3 and section 3.4.

Since the zk-SNARKs algorithm needs parameters with fixed sizes, it is necessary to determine the number of anonymous assets to be spent and the number of newly-generated assets in a vAggreSeg. We find that the usage characteristics of anonymous assets are similar to those of UTXOs in Bitcoin. Therefore, we analyzed all Bitcoin transactions from January 2009 to April 2018, and counted the number of inputs and outputs of each transaction. The proportion distribution chart is shown in Fig 1. (a) is the chart for transaction inputs, and shows that transactions with 1 input account for the majority of all transactions. (b) is the chart for transaction outputs, and shows that after 2011 transactions with 2 outputs account for the majority of all transactions.

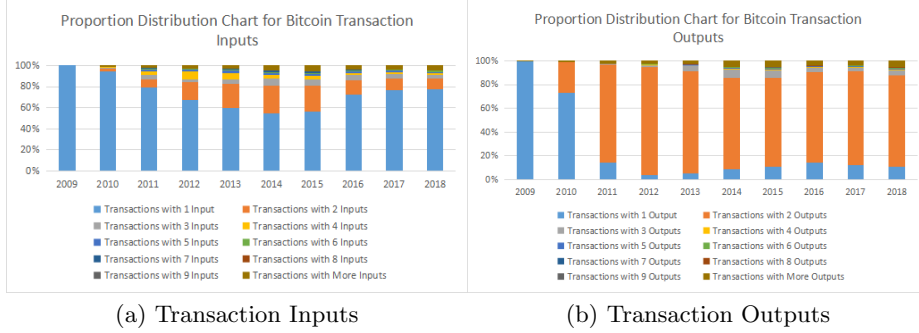


Fig. 1: Proportion Distribution Chart for Bitcoin Transactions

Based on the above analysis, each vAggreSeg spends one anonymous asset, and generates two new anonymous assets. If the number of assets to be spent is greater than 1, multiple vAggreSegs can be contained in a shielded transaction. The spending and generation of anonymous assets are discussed in the section 3.4 and section 3.5.

3.2 Anonymous Addresses of the Account

An anonymous address consists of the payment private key (a_{sk}), payment public key (a_{pk}), communication private key (sk_{enc}) and communication public key (sk_{pub}). The payment keys a_{sk} and a_{pk} are used to verify the sender's ownership of an anonymous asset, and the communication keys sk_{enc} and sk_{pub} are used to encrypt the metadata of anonymous assets.

Current asymmetric cryptography is complex and unfit for the zero-knowledge proof algorithm, so Equation 1 is used to deduce a_{pk} from a_{sk} .

$$a_{pk} = SHA256Compress(1100 + a_{sk}, [0]^{256}) \quad (1)$$

In Equation 1, **SHA256Compress** is a hash function. The inputs of this function are two 256-bit values, and the output is one 256-bit value. The payment private key a_{sk} is a 252-bit random value selected by the user⁵.

The relation between a_{sk} and a_{pk} is encoded into the zero-knowledge proof. The blockchain nodes validate the zero-knowledge proof to judge whether a_{pk} is deduced from a_{sk} . The ownership of an anonymous asset is thus verified.

By Curve25519 algorithm, sk_{enc} and sk_{pub} are used to negotiate a symmetric key that encrypts the metadata of anonymous assets. Given Equation 1, 2 and 3, a_{pk} , sk_{enc} and sk_{pub} can be deduced from a_{sk} , so the user only needs to store a_{sk} . The details of Equation 2 and 3 are introduced in the paper [2].

$$sk_{enc} = clamp_{Curve25519}(SHA256Compress(a_{sk}, 00000001 + [0]^{248})) \quad (2)$$

$$sk_{pub} = KA.DerivePublic(sk_{enc}) \quad (3)$$

3.3 Metadata of the Anonymous Asset

This paper defines the metadata of the anonymous asset, which coexists with the original assets of the blockchain system.

Payment Public Key a_{pk} . It is used to identify the ownership of this asset.

Value. Although the number of assets that a vAggreSeg spends and generates is fixed, an asset can be assigned to an arbitrary value as long as it does not cause double-spending. This ensures the flexibility of the transaction amounts.

Sequence Factor ρ . It is used to generate a sequence for this asset.

Commitment Factor r . It is a random value, and adds randomness to the commitment to improve the security.

id. It is the type of this anonymous asset.

When an anonymous asset is spent, the first problem is how to represent the asset without disclosing the private information. We use the **commitment**(cm) to represent an anonymous asset. The generation of cm is shown in Equation 4.

$$cm = SHA256(10110000 + a_{pk} + Value + \rho + r + id) \quad (4)$$

The sender encodes the commitment, the metadata and the relation between them into the zero-knowledge proof. A blockchain node can thus verify whether a given commitment is generated by this anonymous asset.

The second problem is how to verify the existence of an anonymous asset. Aimed at this problem, a blockchain node organizes all existed commitments into a Merkle tree. Since anonymous assets are ceaselessly generated, each anonymous asset is corresponding to a new Merkle tree. When an asset is spent, the corresponding Merkle tree root rt is put into the vAggreSeg (*i.e.*, **anchor**). The

⁵ **SHA256Compress** is used for the generation of a_{pk} , sk_{enc} , sequences and other parameters, so a prefix “1100” is added to a_{sk} to distinguish different functions of **SHA256Compress**.

sender encodes rt , the commitment for the asset to be spent, Merkle branch of this commitment, and the verification method of the Merkle tree into the zero-knowledge proof. A blockchain node checks the existence of rt and verifies the zero-knowledge proof so the existence of an anonymous asset is proved.

3.4 Asset Transfer Mechanism

Each vAggreSeg disables the old anonymous asset, and generates new anonymous assets⁶. The destruction of the old asset is used to prevent double-spending behaviours, and is discussed in the section 3.5. The commitments for the new assets are put into the vAggreSeg (*i.e.*, `commitments[2]`), by which blockchain nodes can generate the new Merkle tree. The relation between the metadata of new assets and its commitments is encoded into the zero-knowledge proof.

The sender encrypts the metadata of the new assets using the key generated by Curve25519[2], and puts the ciphertext into the vAggreSeg (*i.e.*, `encCiphertexts[2]`). The receiver tries to decrypt the cyphertext of each shielded transaction in the network, and puts the new anonymous assets into the wallet if the corresponding cyphertext can be decrypted.

3.5 Anti Double-Spending Mechanism

Each anonymous asset is assigned to a unique sequence. The generation of the new sequence sn_i^{new} is shown in Equation 5, 6 and 7.

$$h_{sig} = BLAKE2b - 256("AssetComputeSig", randomSeed + sn^{old} + aggreSegPubKey) i \in 0, 1 \quad (5)$$

$$\rho_i^{new} = SHA256Compress(0 + i + 10 + \varphi, h_{sig}) \quad (6)$$

$$sn_i^{new} = SHA256Compress(a_{sk}, \rho_i^{new}) \quad (7)$$

sn^{old} is the sequence of the asset spent by this vAggreSeg, φ is a random value selected by the sender, and $BLAKE2b - 256$ is a hash function proposed in the paper [1]. Since sn^{old} is a unique value, sn_i^{new} deduced from sn^{old} is unique.

No anonymous assets exist at the beginning. In this case, new anonymous assets are generated from transparent assets. The vAggreSeg will spend an "initial" anonymous asset with $value = 0$. A random value is assigned to the sequence of the "initial" asset, so the sequences of new assets are generated.

A blockchain node maintains a set $snSet$, which contains all anonymous assets' sequences. A shielded transaction contains the sequences of assets to be spent. For each shielded transaction, a blockchain node verifies whether any of the sequences has been present in $snSet$. If so, it proves a double-spending behaviour so the transaction is rejected. By contrast, if the shielded transaction is valid, these sequences will be added into $snSet$.

⁶ Each vAggreSeg generates two new anonymous assets. In most cases, one asset is sent to the receiver, and the other acts as the sender's change.

4 Generation and Verification of Zero-Knowledge Proof

Based on the verification rules, we use the `libsnark` library[4] to generate and to verify the zero-knowledge proof. The technology behind the `libsnark` library is the zk-SNARKs algorithm.

4.1 Public Parameter Initialization

The zk-SNARKs algorithm needs two public parameters before usage: the proving key and the verifying key. The sender uses the proving key to generate the zero-knowledge proof, and the blockchain nodes use the verifying key to validate the zero-knowledge proof. The `libsnark` library utilizes the elliptic curve cryptography based on the `alt_bn128` to generate the public parameters.

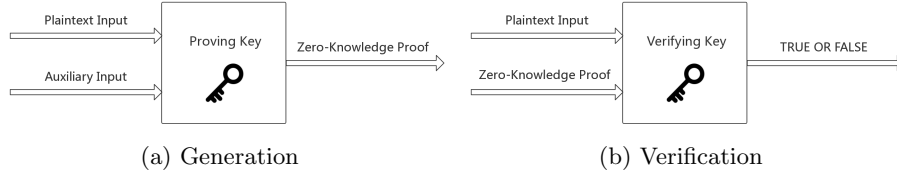


Fig. 2: Zero-Knowledge Proof Processing

4.2 Generation of Zero-Knowledge Proof

The generation of zero-knowledge proof is shown in Fig 2(a). The plaintext input is known by the sender and validators, and is represented by the `vAggreSeg`. The auxiliary input is only known by the sender. Table 1 shows the auxiliary input.

Table 1: Structure of Auxiliary Input

Field	Description
$merkle_{Branch}$	Merkle branch of the commitment for the asset to be spent
$asset^{old}$	Metadata of the asset to be spent
$asset_i^{new}, i \in \{0, 1\}$	Metadata of each newly-generated asset
a_{sk}^{old}	Payment private key of the asset to be spent
φ	Factor to generate the sequence, shown in Equation 6

4.3 Verification of Zero-Knowledge Proof

As is shown in Fig 2(b), a blockchain node verifies the zero-knowledge proof using the verifying key and the plaintext input.

If the zero-knowledge proof is valid, it proves that:

1. The asset to be spent exists in the blockchain system.
2. The sender has the ownership of the asset to be spent.
3. The sequence of the asset to be spent is valid.
4. ρ_i^{new} in each newly-generated asset $asset_i^{new}$ is valid.
5. The commitment for each newly-generated asset $asset_i^{new}$ is valid.
6. The type of the asset to be spent, the type of the newly-generated assets, and the `assetId` in the `vAggreSeg` are the same.
7. The balance of payment is ensured. In other words, Equation 8 is satisfied.

$$v_{pub}^{old} + value^{old} = v_{pub}^{new} + \sum_{i=0}^1 value_i^{new} \quad (8)$$

v_{pub}^{old} and v_{pub}^{new} are the values of `vpub_old` and `vpub_new` in the `vAggreSeg`, $value^{old}$ is the value of $asset^{old}$, and $value_i^{new}$ is the value of $asset_i^{new}$.

5 Implementation and Experiments

In this section, we will first introduce our development experiments, and then introduce the initialization and verification of a shielded transaction. Finally, we will describe our experiments and give the experiment results.

5.1 Environments for Implementation and Experiments

Our system is developed by C++, and the environment is shown in Table 2. It relies on the `rocksdb` to store the data such as anonymous assets.

Table 2: System development environment

Environment Type	Specifications
Operating System	Ubuntu 16.04
Memory	16G
CPU	Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz
Network Bandwidth	10Mbps

5.2 Initialization and Verification of a Shielded Transaction

When a shielded transaction is initialized, a request is sent to the wallet through RPC (Remote Procedure Call), and anonymous assets are searched in the wallet. For example, Alice transfers anonymous assets to Bob and Carl, and the values to be transferred are 10 and 8. Assume an asset with value = 20 is selected, and two `vAggreSegs` are constructed shown in Fig 3. The asset back to Alice in the first `vAggreSeg` is used by the second `vAggreSeg`. After the anonymous assets

	Input	Output	
	Alice	Bob	Alice
AggreSegDesc_1	20	10	10
AggreSegDesc_2	Alice	Carl	Alice
	10	8	2

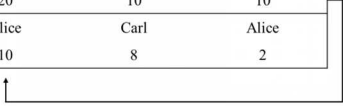


Fig. 3: vAggreSegs in the Shielded Transaction

are selected, the zero-knowledge proof is generated and packed into the shielded transaction. Finally, the shielded transaction is broadcast to the network.

When a blockchain node receives a shielded transaction from the network, it first detects the double-spending behaviour, and then verifies the existence of the Merkle tree root contained in each vAggreSeg. At last, the node verifies the zero-knowledge proof. If the shielded transaction is valid, the blockchain state will be updated, and persisted to the `rocksdb`.

5.3 Experiments

We evaluated our system by experiments under the environment in Table 2.

Step1. 200 accounts with anonymous addresses were generated, and then we allocated transparent assets of the type `0x01` and `0x02` to each account.

Step2. Each of the first 100 accounts transferred anonymous assets of the type `= 0x01` to the last 100 accounts. And then, each of the last 100 accounts transferred anonymous assets of the type `= 0x02` to the first 100 accounts. Since no anonymous assets existed before Step2, the anonymous assets to be transferred were generated from the transparent assets.

Step3. Each of the first 100 accounts transferred anonymous assets of the type `= 0x02` to the last 100 accounts. And then, each of the last 100 accounts transferred anonymous assets of the type `= 0x01` to the first 100 accounts.

Step2 converted transparent assets into anonymous assets, and Step3 transferred anonymous assets between accounts. The operations in our experiments were thus divided into two categories: the conversion and transfer of anonymous assets. The number of transactions in each category was 200. For each category, we measured the time of zero-knowledge proof generation and verification.

The time of generation is shown in Fig 4(a). This time for both categories is maintained at about 60 seconds. The generation takes a long time, because the algorithm involves hundreds of thousands of times of polynomial multiplication.

The time of verification is shown in Fig 4(b). This time for both categories is maintained at about 45 milliseconds. The verification takes a shorter time, because it only performs a small amount of computation on the elliptic curve.

6 Conclusion

Blockchains pose challenges to users' privacy. Among all the schemes for the privacy protection, the zero-knowledge proof algorithm conceals most private

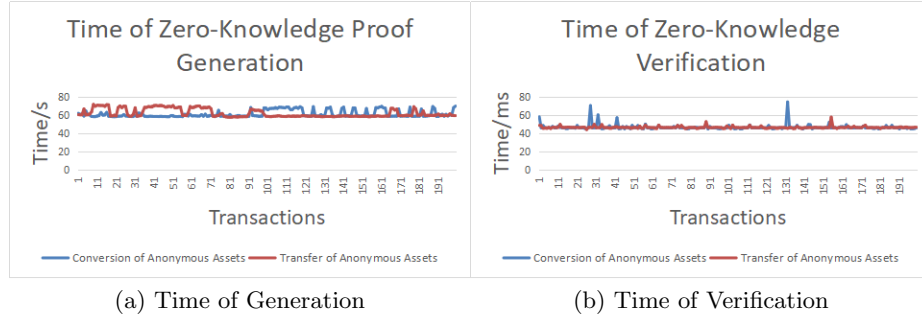


Fig. 4: Time of Zero-Knowledge Proof Generation and Verification

information in a transaction, while each participant can validate the transaction without such information. However, current schemes based on the zero-knowledge proof algorithm are only aimed at the UTXO and single-asset model.

Based on the zero-knowledge proof algorithm, we propose a privacy protection scheme aimed at the account and multi-asset model. We design the verification rules for the shielded transactions, and uses zk-SNARKs algorithm to generate and to verify the zero-knowledge proof. We implemented our scheme, and evaluated the system by experiments. Our future work includes finding more efficient algorithms for the zero-knowledge proof generation.

7 Acknowledgments

This paper is supported by National Natural Science Foundation of China under Grant 61772502 and 61672499.

References

1. Aumasson, J.P., Neves, S., Wilcox-O’Hearn, Z., Winnerlein, C.: Blake2: simpler, smaller, fast as md5. In: International Conference on Applied Cryptography and Network Security. pp. 119–135 (2013)
2. Bernstein, D.J.: Curve25519: New diffie-hellman speed records. Lecture Notes in Computer Science **3958**(1), 207–228 (2006)
3. Evan, D., Daniel, D.: Dash: A payments-focused cryptocurrency. <https://github.com/dashpay/dash/wiki/Whitepaper>
4. Madars, V.: libsnark. <https://github.com/scipr-lab/libsnark>
5. Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G.M., Savage, S.: A fistful of bitcoins: characterizing payments among men with no names. In: Conference on Internet Measurement Conference. pp. 127–140 (2013)
6. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from bitcoin. IEEE Symposium on Security & Privacy pp. 397–411 (2013)
7. Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: Security and Privacy. pp. 459–474 (2014)
8. Shen, N., Sarang, N.: Monero is not that mysterious. <https://lab.getmonero.org/pubs/MRL-0003.pdf>