A Decentralized and Anonymous Data Transaction Scheme Based on Blockchain and Zero-Knowledge Proof in Vehicle Networking (Workshop Paper)

Wei Ou(✉), Mingwei Deng, and Entao Luo

Hunan University of Science and Engineering, Yongzhou, China
ouwei1978430@163.com

Abstract. Data transaction in internet of vehicles is a transaction occurs between vehicle owner and data buyer. Blockchain is a new technology that brings decentralized ledger system for user, which means users could make payment without the third party. There are several projects combined internet of vehicles and Blockchain, however, none of them realize a trustworthy anonymous data transaction. In this paper, we first propose the concept of Super Nodes to guarantee data authenticity, then we construct the anonymity for the transaction base on *zero-knowledge Succinct Non-interactive Argument of knowledge* (zk-SNARKs) and DAP from Zerocash. Moreover, a smart contract is deployed for mutual benefits. Simulation experiment shows this scheme is practical.

Keywords: Internet of vehicles · Blockchain · Zero knowledge · Smart contract

1 Introduction

In recent years, with continuous growth of car ownership, road carrying capacity has reached saturation in many cities, and traffic safety, travel efficiency, and environmental protection have become increasingly prominent. As an important field of in-depth integration of informatization and industrialization, internet of vehicles (IoV) is of great significance to promote integration and upgrade of the automobile, transportation, information and communication industries, and reshape of relevant industrial ecology and value chain systems.

Blockchain is a new technology that first proposed by bitcoin [1]. It describes a decentralized ledger without participation of the third party. In such ledger, it cannot be tampered once data is confirmed. That is achieved by a novel consensus mechanism, called Proof of Work (PoW), and timestamp server. PoW describes a safe accounting

Supported by the construct program of applied characteristic discipline in Hunan University of Science and Engineering.

X. Wang et al. (Eds.): CollaborateCom 2019, LNICST 292, pp. 712–726, 2019.
https://doi.org/10.1007/978-3-030-30146-0_48

system that solves the Byzantine Problem by introducing a computing power competition of distributed nodes to ensure data consistency and ledger consensus. The node who wins the competition will broadcast a "block" in whole network, which contains all transactions he collects and a timestamp to avoid double spending. After more than 10 years of development, consensus mechanisms in Blockchain has from the beginning of a single POW into Proof of gaining (PoS), Delegated Proof of Stack (DPoS), Practical Byzantine Fault Tolerance (PBFT), and other common mechanism occurring together. Moreover, Ethereum introduced smart contracts for Blockchain in 2014, which are Blockchain-based programs that directly control digital assets [2]. Concretely, a smart contract is a computer program that automatically enforces contract terms, is deployed on a shared, replicated ledger. It could maintain its status, control its assets and respond to incoming external information or assets. The digital form means that the contract has to be written into computer-readable code. As long as the participations agree, the rights and obligations established by the smart contract are performed by a computer or computer network. And all these agreements and its realization would be recorded in Blockchain.

There are several projects that combined IoV and Blockchain, such as Smartcarchain and Carblock. They expect that Blockchain and its decentralized characteristic could return the data ownership to data producer, general speaking, the vehicle owners, which means they can freely dispose those data that collected by their vehicles. It gives users an incentive to collect more data. And the more data users collect, the more valuable data are. Automotive and transportation companies are all interested in these data, because they can be analyzed to build better, more targeted products and services, thus earning more profits. This kind of demand for data makes data transaction important. Carblock proposed a data transaction method through smart contract, eliminating the participation of the third party [3]. However, such a method requires buyer to deploy two smart contracts for verifying data and transaction. That may not convenient to a buyer.

Moreover, the anonymity is a feature that users care about. In 2014, based on Bitcoin system, Zcash [4] proposes a new scheme to make an anonymous payment in blockchain system by using zero knowledge proof. This scheme realizes strong anonymity for a payment in Blockchain. It brings us to do research in anonymity of data transaction in IoV, however, there has been no relevant research so far.

In this paper we propose a decentralized anonymous data transaction scheme for vehicle networking based on Blockchain. This scheme allows a data buyer to buy the data that collected by vehicle sensors from a seller. Concretely, it achieves two goals: *(i)* decentralization: there is no third party in our transaction; *(ii)* anonymity: transaction participants and relevant information (including transfer amount) are invisible to others; To do this, we provide a construction of the scheme and design a simulation experiment to test its performance.

The rest of this paper is organized as follow. Section 2 is an overview of our scheme, we describe the concept of *Super Nodes* and main steps of our scheme; Sect. 3 provides technical background including *zk-SNARKs* and DAP scheme; Sect. 4 gives the formal construction of our scheme; Sect. 5 shows the result of simulation experiment; last, we summarize our contribution and discuss the future work.

2 Overview

Our scheme is an extension based on *Zerocash*, which is a blockchain that supports anonymous payment. It describes such a system: every node in the system could deposit their base currency (e.g., bitcoin) in an escrow to mint a coin in *Zerocash* ledger, this new coin is bound to the address public key of the node. The minting process will generate some secret values, only someone who knows the corresponding address private key and some secret values can use it; to use the coin, the node who owns the address private key and the secret values will broadcast a transaction: I "destroy" my old coin to mint two new coins that one of them or both of them are bound to the target address public key, which is bound to the receiver, and the total value of two new coins is equal to the value of the old coin; besides, in order to avoid additional infrastructure and assumption, the transaction contains a ciphertext of the secret values of new coin, which is encrypted by receiver's encryption public key; lastly, the coin receiver scans the transaction in the ledger and use his encryption private key to decrypt the ciphertext until he finds the transaction for him. Then, he can use the coin with his address private key and the secret values he gets from the ledger. This system achieve anonymity by introducing *zk-SNARKs*, which is a cryptology that allows anyone check a proof of a statement in a short time (we show details in Sect. 3.1).

We realize such a system could use for the data transaction in vehicle networking. Every buyer and seller can join such a system by deposit any digital currency to an escrow, thus achieving the goal of anonymous transaction. However, this transaction is just a currency transfer, it only meets the requirement for sellers rather than mutual benefit. Namely, this system cannot guarantee that a buyer obtains the data he wants. To solve the problems, we construct the concept of *Super Nodes*.

2.1 Super Nodes

Super Nodes are some nodes that run by some trusted third party, they have all function of normal nodes such as transaction broadcasting, transaction check and consensus reaching, meanwhile they also have some special functions.

- **Data checking and storage.** A vehicle owner could upload data to the *Super Nodes* through networking protocol interface from corresponding vehicle sensor, note that these data are encrypted by data encryption public key that the hardware generated, which guarantee that the *Super nodes* can't "see" these data. When a *Super Node* receive the data, it will verify its authenticity by checking if they come from vehicle sensors, then storages and compresses them if verify success. This function enables *Super Node* to become a de-centralized server, and data in server are authentic.
- **dataID generating.** After the process of data check and storage, every *Super Node* will generate a *dataID* for the data and a cyphertext of data uploader's address public key (encrypted by an encryption public key). *dataID* is a hash of corresponding data and encrypted by the uploader's encryption public key (note this is a key pair different from data encryption key), due to that every *Super Node* processes the same procedures, different *Super Node* will generate same *dataID* if they

received same row data. Consequently, they can make a consensus through Practical Byzantine fault tolerance [5] among all *Super Nodes*. After that, the *dataID* is broadcasted to whole Blockchain system, to make the consensus that append it in the public ledger. In this way, every buyer could verify the *dataID* for confirming the authenticity of data by simply checking the public ledger.

- **Data retrieval.** Anyone in the system can get the encryption of data by the corresponding *dataID* from *Super Nodes*. However, only the one who owns data encryption private key could get the origin data. Note that there is no constructed of private communication channel such as that a seller sends the data to a buyer individually, therefore we don't have to add additional infrastructures or worry about being eavesdropped.

2.2 Steps

With *Super Nodes,* which we show function component in Fig. 1, a seller could upload his data, and a buyer could confirm the authenticity and get the data. However, this is still not enough to make an anonymous data transaction, because the data is encrypted by the seller's data encryption public key, the buyer must to know the data encryption private key for getting the data.

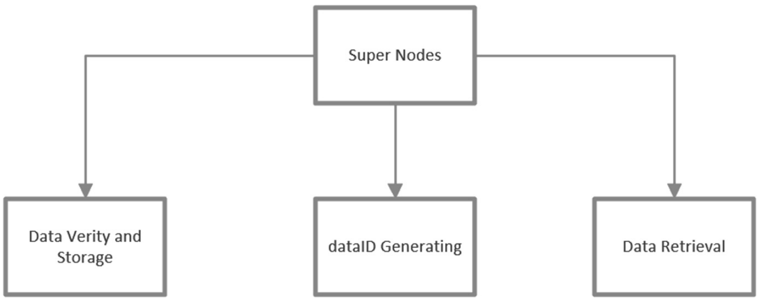

Fig. 1. Function of *Super Nodes*

So far, we have converted the issue that money to data to the issue that money to private key. So how can we make the money-to-key transaction in Blockchain? We find a solution: *Hashed-Timelock Contract* (*HLTC*) [6]. *HLTC* is a technology used for cross-chain payment. It requires the payer to set a smart contract about a cryptographic puzzle (usually is a hash value of sha256 function), anyone trigger the contract and solve the puzzle (e.g., support the preimage of a hash value) could get the money that the payer deposit to an escrow. The key point of *HLTC* is make sure only the recipient knows the answer of the puzzle.

With *Super Nodes* and *HLTC*, we can finally extend *Zerocash* system to make an anonymous data transaction. Here, we show main steps of data transaction; algorithms construction is in Sect. 4.

Step1: Info Sending. If a buyer wants to buy a seller's data, the seller should firstly send some information to the buyer. Which including *(i) dataID*; *(ii)* hash of data decryption private key. The buyer checks *dataID* in the ledger. If it does exist in the ledger, the buyer confirms the authenticity of data. But he cannot get the data immediately, because the data is encrypted and he don't know the decryption key. The hash is a required item for smart contract deployment after coin transfer.

Step2: Anonymous Payment. The buyer now needs deposit corresponding amount of digital currency for data in escrow to mint a coin A. After that, he mints another coin B that is bound to the receiver's address public key and transfer the value of coin A to coin B. Next, we modify the origin scheme of *Zerocash*: we broadcast the transfer of coin value but we don't broadcast the ciphertext of the secret values of coin B.

Step3: Smart Contract. So far, the seller still doesn't get the coin because he doesn't know the secret value and the buyer doesn't get the data because he doesn't know the data encryption private key. So, the buyer first encrypts the secret value of coin B by the receiver's encryption public key. Then, the buyer creates such a hash lock smart contract using hash that he obtains in Step1: if someone can support the pre-image of the hash, which is the data encryption private key, he will get the ciphertext of the secret values.

This is an overview of our construction, we show sketch in Fig. 2. If the smart contract is trigged and completed, the buyer gets the encryption private key to decrypt the data and the seller get the secret values to use the coin. So that, the data transaction completed.

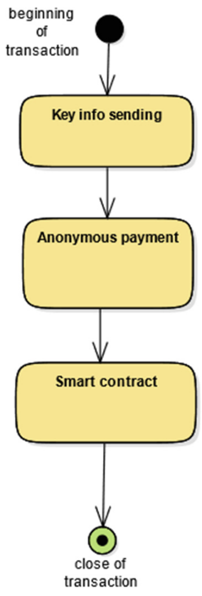

Fig. 2. Steps of our data transaction

3 Background

3.1 zk-SNARKs

We used the zero-knowledge Succinct Non-interactive Argument of Knowledge (zk-SNARK) as our main cryptographic technology to make an anonymous data transaction in this paper. In this section, we will give three components of zk-SNARKs and we refer the reader to [7, 8] for concrete protocol and implementation.

Arithmetic Circuit
An arithmetic circuit is consisted of wires with specific values and *bilinear gate* with only addition and multiplication. Given a finite field $\mathbb{F}$, an $\mathbb{F}$-*arithmetic circuit* takes input that are element in $\mathbb{F}$, and its gates output elements in $\mathbb{F}$. Considering we have an input $x \in \mathbb{F}^n$ and an auxiliary input $a \in \mathbb{F}^h$, we have the definition of arithmetic circuit satisfiability that analogous to the boolean case as follows:

Definition 3.1. *The circuit satisfaction problem of a circuit* $C\colon \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ *with bilinear gate is defined by the relation* $\mathcal{R}_C = \{(x, a) \in \mathbb{F}^n \times \mathbb{F}^h : C(x, a) = 0^l\}$; *and its language is* $\mathcal{L}_C = \left\{x \in \mathbb{F}^n : \exists a \in \mathbb{F}^h, C(x, a) = 0^l\right\}$.

Note that a is what we want to obtain in zk-SNARKs, which we called *witness*.

Quadratic Arithmetic Program
zk-SNARKs leverages quadratic arithmetic programs (QAPs) [9] to converted any arithmetic circuit to corresponding sets of polynomials. The main idea of QAPs is that converting circuit to three basic sets of polynomials and a target polynomial, these polynomials must meet such a fact: there is a product of three basic polynomials sets and some coefficients could divide the target polynomial. We give the formal definition of a QAPs below:

Definition 3.2. *A quadratic arithmetic program of size m and degree d over* $\mathbb{F}$ *is a tuple* $(\vec{A}, \vec{B}, \vec{C}, Z)$, *where* $\vec{A}, \vec{B}, \vec{C}$ *are three vectors, each of them is m + 1* polynomials in $\mathbb{F}^{\leq d-1}[z]$, *and* $Z \in \mathbb{F}[z]$ *has degree exactly d.*

And as we mentioned above, *a* QAPs induces a satisfaction problem:

Definition 3.3. *The* **satisfaction problem** *of a size-m QAP* $(\vec{A}, \vec{B}, \vec{C}, Z)$ *is the relation* $\mathcal{R}_{(\vec{A},\vec{B},\vec{C},Z)}$ *of pairs* (x, s) *such that (i)* $x \in \mathbb{F}^n$, $s \in \mathbb{F}^m$, *and* $n \leq m$; *(ii)* $x_i = s_i$ *for* $i \in [n]$*(i.e., x extends s); (iii) the polynomial* $Z(z)$ *divided the following one:*
$\left(A_0(z) + \sum_{i=1}^m s_i A_i(z)\right) \cdot \left(B_0(z) + \sum_{i=1}^m s_i B_i(z)\right) - \left(C_0(z) + \sum_{i=1}^m s_i C_i(z)\right)$. *We denote by* $\mathcal{L}_{(\vec{A},\vec{B},\vec{C},Z)}$ *the language of* $\mathcal{R}_{(\vec{A},\vec{B},\vec{C},Z)}$.

So far, we have the definition of arithmetic circuit and QAPs. Due to that the QAPs is a result of encoding arithmetic circuit, we can combine the Definitions 3.1, 3.2 and 3.3 to obtain a complete definition of QAP:

Definition 3.4. *A QAP Q over field* $\mathbb{F}$ *consist of three sets of m + 1 polynomials* $\vec{A} = \{a_k(x)\}$, $\vec{B} = \{b_k(x)\}$, $\vec{C} = \{c_k(x)\}$, *for* $k \in \{0 \ldots m\}$, *and a target polynomials t(x). Suppose F is a function that takes as input n elements of* $\mathbb{F}$ *and output* n' *elements, for a total of* $N = n + n'$ *I/O elements. Then we say that Q computes F if:* $(s_1, \ldots, s_N) \in \mathbb{F}^N$ *is* *valid assignment of F's inputs and outputs,* if and only if *there exist coefficients* $(s_{N+1}, \ldots, s_m)$ *such that t(x) divides p(x), where:*

$$p(x) = \left(a_0(x) + \sum_{k=1}^{m} s_k \cdot a_k(x)\right) \cdot \left(b_0(x) + \sum_{k=1}^{m} s_k \cdot b_k(x)\right) - \left(c_0(x) + \sum_{k=1}^{m} s_k \cdot c_k(x)\right).$$

In other words, there must exist some polynomial h(x) such that h(x) · t(x) = p(x). The size of Q is m, and the degree is the degree of t(x).

Verifiable Computation (VC)

A verifiable computation (VC) [10] for $\mathbb{F}$-*arithmetic circuit C:* $\mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ allows a prover to generate a non-interactive proofs for the language $\mathcal{L}_C$ using the public parameters that generated by VC, and anyone can use another generated public parameter to verify these proofs. Moreover, the verification process only requires a short time. Concretely, a VC contains three set of polynomial-time algorithms: KeyGen(), Compute() and Verify(). Below we defined the three algorithms:

- $(EK_F, VK_F) \leftarrow \text{KeyGen}(F, 1^\lambda)$: *The public key generation algorithm takes the function F, which is exact* $\mathbb{F}$*-arithmetic circuit C, and a security parameter* λ *as inputs; Then it output a public evaluation key* EK_F *and a public verification key* VK_F*.*
- $(y, \pi) \leftarrow \text{Compute}(EK_F, x)$: *The proof computation algorithm takes evaluation key* EK_F *and x to output* $y \leftarrow F(x)$ *and a non-interactive proof* π *for y's correctness.*
- $b \leftarrow \text{Verify}(VK_F, x, \pi, y)$: *The proof verification algorithm takes verification key* VK_F*, x and a proof* π *as input. It outputs* $b = 1$ *if* $y \leftarrow F(x)$*.*

The three algorithms defined above are main part of VC scheme. Note that this system that we defined is actually not public verifiable, the verification key VK_F should be hidden in some *designated verifier* otherwise the scheme is vulnerable to attack. To avoid such issue, we introduce the concept of **Zero-knowledge Verifiable computation**, which required the verifier learns nothing about the prover's input beyond the output of computation. Concretely, we change the proof computation algorithm Compute() and the proof verification algorithm Verify().

- $(\pi) \leftarrow \text{Compute}(EK_F, x, a)$ *The proof computation algorithm takes evaluation key* EK_F *and* $(x, a) \in \mathcal{R}_C$ *(see* Definition 3.1*) as input to output a non-interactive proof* π *for the statement* $x \in \mathcal{L}_C$*.*
- $1 \leftarrow \text{Verify}(VK_F, x, \pi)$ *The proof verification algorithm takes verification key* VK_F*, x and a proof* π *as input to outputs* $b = 1$ *if it is convinced that* $x \in \mathcal{L}_C$*.*

With this change, *evaluation key* EK_F and *verification key* VK_F could be public in system, thus allowing anyone to check the *proof* π. That is a very applicable scheme for blockchain system, because it allows every node in the system to check a transaction and thus making consensus in the chain. Such a scheme also referred to as a non-interactive zero-knowledge proof. From these definitions, we give the properties that zk-VC scheme should satisfy:

- **Correctness.** *For any function F, and any input u to F, a honest prover could always convince the verifier that he knows the witness a. Namely, if we run* $(EK_F, VK_F) \leftarrow \text{KeyGen}(F, 1^\lambda)$ *and* $(\pi) \leftarrow \text{Compute}(EK_F, x, a)$*, we will always get* $1 \leftarrow \text{Verify}(VK_F, x, \pi)$*.*

- **Security.** For any function F and any probabilistic polynomial-time adversary $\mathcal{A}$, $\Pr[(\hat{u}, \hat{\pi}) \leftarrow \mathcal{A}(EF_K, VK_F) : \text{x} \notin \mathcal{L}_C \text{ and } 1 = \text{Verify}(VK_F, \hat{u}, \hat{\pi})] \le \text{negl}(\lambda)$[1].
- **Efficiency.** KeyGen() is assumed to be a one-time operation whose cost is amortized over many calculations, however, we required a cheaper Verify() than evaluating F.

3.2 Dap

The main payment scheme used in this paper is *Decentralized Anonymous Payment* (DAP) scheme, which is proposed by *Zerocash*. As we described in Sect. 2, this is a scheme making anonymity in a payment. Here, we provide basic components of DAP. We refer the interested reader to [4] for complete scheme.

DAP scheme is consisted of a tuple of polynomial-time algorithms: Setup, CreateAddress, Mint, Pour, Receive.

- **Setup.** This algorithm is executed by a trusted third party. It requires to input a security parameter, then it will output a public parameter pp, which includes the *public knowledge* of zk-SNARKs (see Sect. 3.1).
- **CreateAddress.** This algorithm is executed by users in the system. Each user can generate at least one pair $(a_{\text{pk}}, a_{\text{sk}})$, where $a_{\text{pk}} = (\text{addr}_{\text{pk}}, \text{pk}_{\text{enc}})$, and $a_{\text{sk}} = (\text{addr}_{\text{sk}}, \text{sk}_{\text{enc}})$. Concretely, addr_{sk} is a random number, and $\text{addr}_{\text{pk}} := \text{PRF}^{\text{addr}}_{a_{\text{sk}}}(0)$[2], they are an address key pair bound to the user; Encryption key pair $(\text{pk}_{\text{enc}}, \text{sk}_{\text{enc}})$ is generated based on a *key-private encryption scheme* [11], it is used for encryption. Note a user may generate any number of key pairs.
- **Mint.** This algorithm is executed by a payer. It requiresto input public parameter pp, a coin value v, and a destination address public key addr_{pk}, then it will outputs a coin c and a $\text{TX}_{\text{min}t}$. Here, we give concrete steps to mint a coin (*i*) the algorithm generates three random value ρ, r, s; (*ii*) the algorithm compute $\text{sn} := \text{PRF}^{\text{sn}}_{a_{\text{sk}}}(\rho)$, $k := \text{COMM}_r(\text{addr}_{\text{pk}}||\rho)$[3] and $\text{cm} := \text{COMM}_s(v||k)$. (*iii*) the algorithm outputs the minting result: a coin $\text{c} := (\text{addr}_{\text{pk}}, v, \rho, r, s, \text{cm})$ and a mint transaction $\text{TX}_{\text{min}t} = (v, k, s, \text{ cm})$. Note that anyone could verify if cm is a coin commitment of a coin with value v by checking that $\text{cm} := \text{COMM}_s(v||k)$ is equal to cm and no one can discern the coin owner or a serial number sn, because they don't know the address key addr_{pk} and the secret value ρ. As before, $\text{TX}_{\text{min}t} = (v, k, s, \text{cm})$ is added in ledger only by the payer deposits the correct amount of basecoin to escrow.
- **Pour.** This algorithm is executed by a payer to spend a coin. This is an operation to transfer the values of a set of input coins to another set of new output coins, the total value of input coins is equal to the total values of the output coins. Suppose a payer

[1] Negligible function.

[2] PRF() is a pseudorandom function.

[3] COMM() is a statistically-hiding non-interactive commitment scheme, which satisfy the verifiability: given $c := \text{COMM}_r(s)$, one who knows r and s can verify that $\text{COMM}_r(s)$ is equal to c.

with address key pair $\left(\mathrm{addr}_{\mathrm{pk}}^{old}, \mathrm{addr}_{\mathrm{sk}}^{old}\right)$ wants to pay the coin $\mathrm{c}^{old} = \left(\mathrm{addr}_{\mathrm{pk}}^{old}, v^{old}, \rho^{old}, r^{old}, s^{old}, \mathrm{cm}^{old}\right)$ to a recipient. To do this, the payer produces two new coins c_1^{new} and c_2^{new} targeted at two address public key $\mathrm{addr}_{\mathrm{pk},1}^{new}$ and $\mathrm{addr}_{\mathrm{pk},2}^{new}$ with the value meet $v_1^{new} + v_2^{new} = v^{old}$ (note one of them is recipient's address public key and another may give value of 0 for hiding the concrete transfer amount); Take inputs as public parameter pp, the Merkle-tree root rt, an old coin c^{old}, an old addresses secret key $\mathrm{addr}_{\mathrm{sk}}^{old}$, a authentication path path_{old} from cm^{old} to root rt, two new values v_1^{new} and v_2^{new}, two new addresses public key $\mathrm{addr}_{\mathrm{pk},1}^{new}$ and $\mathrm{addr}_{\mathrm{pk},1}^{new}$, and some transaction information. For each $i \in \{1, 2\}$, the algorithm proceeds as follows: (*i*) the algorithm generates three random value $\rho_i^{new}, r_i^{new}, s_i^{new}$; (*ii*) the algorithm computes $k_i^{new} := \mathrm{COMM}_{r_i^{new}}\left(\mathrm{addr}_{\mathrm{pk},i}^{new} \parallel \rho_i^{new}\right)$, $\mathrm{c}_i^{new} := \left(\mathrm{addr}_{\mathrm{pk},i}^{new}, v_i^{new}, \rho_i^{new}, r_i^{new}, s_i^{new}, \mathrm{cm}_i^{new}\right)$. (*iii*) the algorithm computes a zk-SNARKs proof π for the following NP statement:

"Given a Merkle-tree root rt, *serial number* sn^{old}, *and coin commitment* cm_1^{new}, cm_2^{new}, *I know coins* c^{old}, c_1^{new}, c_2^{new}, *and secret key* $\mathrm{addr}_{\mathrm{sk}}^{old}$ *meet:*

a. *The coins satisfy: for* c^{old} *it holds that* $k^{old} = \mathrm{COMM}_{r^{old}}\left(\mathrm{addr}_{\mathrm{pk}}^{old} \parallel \rho^{old}\right)$ and $\mathrm{cm}^{old} = \mathrm{COMM}_{s^{old}}\left(v^{old} \parallel k^{old}\right)$; *Similarly for* c_1^{new} *and* c_2^{new}.
b. *The address secret key matches the public key*: $\mathrm{addr}_{\mathrm{pk}}^{old} = \mathrm{PRF}_{\mathrm{addr}_{\mathrm{sk}}^{old}}^{\mathrm{addr}}(0)$.
c. *The serial number is computed correctly:* $\mathrm{sn}^{old} := \mathrm{PRF}_{\mathrm{addr}_{\mathrm{sk}}^{\mathrm{old}}}^{\mathrm{sn}}\left(\rho^{old}\right)$.
d. *The coin commitment* cm^{old} *appears as a leaf of a Merkle-tree with root* rt.
e. *The values added up:* $v_1^{new} + v_2^{new} = v^{old}$.

With all these processes, the algorithm outputs a transaction $\mathrm{TX}_{pour} = \left(\mathrm{rt}, \mathrm{sn}^{old}, \mathrm{cm}_1^{new}, \mathrm{cm}_2^{new}, \mathrm{info}, \pi, \mathrm{C}_1, \mathrm{C}_2\right)$, where C_1 is a ciphertext that is the encryption of the plaintext $\left(v_1^{new}, \rho_1^{new}, r_1^{new}, s_1^{new}\right)$ under $\mathrm{pk}_{\mathrm{enc}}$ (similar to C_2); two new coins c_1^{new}, c_2^{new}. As before, TX_{pour} is rejected by the ledger if the serial number sn^{old} appears in a previous TX_{pour}, therefore avoiding double spending. Note to use a coin c_i, the payer must know following required items: value v_i, three rand ρ_i, r_i, s_i and the corresponding address secret key $\mathrm{addr}_{\mathrm{sk}}$.

- **Receive.** This algorithm is executed by the recipient. Take a public parameter pp, a pair of recipient key $\left(a_{\mathrm{pk}}, a_{\mathrm{sk}}\right)$, and the current ledger, the algorithm scan the transaction TX_{pour} in the ledger to find and decrypt the ciphertext C_i (using his $\mathrm{sk}_{\mathrm{enc}}$), thus obtaining the required secret values to use the coin.

Anonymity of DAP is mainly reflected in pour transaction, because of zk-SNARKs, the payer doesn't have to reveal the identity of both sides, transaction amount and account balance in public. Moreover, the buyer can not trace the coin flow of the coin he mints in pour transaction, because he doesn't know the serial number of the coin.

4 Construction of Decentralized and Anonymous Data Transaction Scheme

In this section, we show how to construct a decentralized and anonymous data transaction scheme using DAP and smart contract. First, we define some basic notion and notation for what we will use in the construction. Then, we will use them to construct our algorithms.

4.1 Basic Notion and Notation

Payer. A *payer* is a node in Blockchain. We use *payer* to denote a data buyer. Namely, the side that pays in payment. In construction, we sometimes use *he* or *him* for convenience.

Recipient. A *recipient* is a node in Blockchain. We use *recipient* to denote a data seller. Namely, the side that receives in payment. In construction, we sometimes use *she* or *her* for convenience.

dataID. A *dataID* is generated by *Super Nodes* from data. Concretely, a *dataID* is obtained by hashing the data and encrypt the result with uploader's *encryption public key*.

ID. It contains a *dataID* and a cyphertext of uploader's *address public key*.

Address. A user may join the system whenever they generate an *address key pair* $(\mathrm{addr}_{\mathrm{pk}}, \mathrm{addr}_{\mathrm{sk}})$ and publishes the public key $\mathrm{addr}_{\mathrm{pk}}$ in the system. The private key $\mathrm{addr}_{\mathrm{sk}}$ is keep by the user to receive the coins sent to him. Note that a user can generate any number of *address key pairs*.

Coins. A coin c contains a *coin commitment* cm, a *coin value* v, a *serial number* s, and a *coin address* $\mathrm{addr}_{\mathrm{pk}}$. cm is a string generated by some cryptographic function (see Sect. 3.2) and it will be appended to the ledger if coin c is minted; v is the denomination of c, namely, the amount of the basecoins (e.g., bitcoin); s is a unique string binding with the c to avoid double spending; $\mathrm{addr}_{\mathrm{pk}}$ is the *address public key* of coin owner, representing who owns c.

Ledger. Our scheme is based on a digital currency system such as Bitcoin. Here, we refer to basecoin as *Basecoin*. Our ledger L is a sequence of transaction and it's append-only. Moreover, it contains transaction of *Basecoin* and two types of new transaction.

λ. It represents an adjustable security parameter to produce a set of global public parameter pp.

Data Encryption Key. It contains $\left(\mathrm{E}_{\mathrm{pk}}, \mathrm{E}_{\mathrm{sk}}\right)$, where E_{pk} is *data encryption public key* used for encrypting the data, and E_{sk} is *data encryption secret key* used for decrypting the data.

Encryption Key. It is a key pair generated by *key-private encryption scheme*. It consists of *encryption public key* pk_{enc} and *encryption private key* sk_{enc}.

Hash. We use *hash* to denote the hash function Hash256.

New Transaction

- *Mint*. A mint transaction TX_{mint} is a statement: a coin with commitment cm and value v is minted. It contains a *coin commitment* cm, a coin value v and two value k and s. Namely, $\mathrm{TX}_{mint} =: (\mathrm{cm},\ v,\ k,\ s)$.
- *Pour*. A pour transaction TX_{pour} is a statement: an old coin is be "destroyed", two new coins is be minted and the value of old coin is transferred to the two new coins. It contains a Merkle tree root rt, the serial number of old coin sn^{old}, two new *coin commitment* cm_1^{new} and cm_2^{new}, and a proof π that prove the transaction initiator owns the old coin, a public value. Note that at least one of cm_1^{new} and cm_2^{new} is bound to the recipient's address public key and the total value of the two coins should equal to old coin. Namely, $\mathrm{TX}_{pour} =: \left(\mathrm{rt},\ \mathrm{sn}^{old}, \mathrm{cm}_1^{new}, \mathrm{cm}_2^{new}, \pi, \mathrm{info}\right)$.

List. For given time T, there are three lists beyond the ledger as public knowledge:

- $IDList_T$. This is a list for all *ID* generated by *Super Nodes*.
- $cmList_T$. This is a list for all coin commitments appearing in mint and pour transactions in L_T
- $snList_T$. This is a list for all serial numbers appearing in pour transaction in L_T.

Note that *ID* contained in $IDList_T$ are not only required to make consensus in the *Super Nodes*, but also in whole blockchain system.

Merkle Tree Overs Coin Commitment and dataID. For given time T, there is an Tree_T over $cmList_T$ and rt_T is its root. Besides, we use $Path_i$ to denote a valid authentication path for leaf cm_i with respect to rt_T.

4.2 Algorithms Construction

Our scheme is a tuple of polynomial-time algorithms (GenerateID, GetID,VerifyID, Setup, CreateAddress, Mint, Pour, VerifyTransaction, Recieve) and a smart contract. The algorithm details of DAP have showed in Sect. 3.2, here we only briefly summarize.

- $\left(dataID, \mathrm{C}_{\mathrm{pk}}\right) \leftarrow \mathrm{GenerateID}\left(\mathrm{data}, \mathrm{addr}_{\mathrm{pk}}, \mathrm{pk}_{enc}\right)$. On input a data set, a $\mathrm{addr}_{\mathrm{pk}}$ and a pk_{enc}, the algorithm computes the result of *hash*(data) and then outputs *dataID*, which is a ciphertext that is encrypted with pk_{enc} on the result, and C_{pk} is a ciphertext that $\mathrm{addr}_{\mathrm{pk}}$ encrypted using pk_{enc}.
- $(dataID) \leftarrow \mathrm{GetID}\left(IDList_T, \mathrm{addr}_{\mathrm{pk}}, \mathrm{sk}_{enc}\right)$. On input an $IDList_T$, a $\mathrm{addr}_{\mathrm{pk}}$ and a sk_{enc}, this algorithm will decrypt every C_{pk} in $IDList_T$ using sk_{enc} to outputting *dataID* that corresponding to his $\mathrm{addr}_{\mathrm{pk}}$.
- $(b1) \leftarrow \mathrm{VerifyID}(dataID, IDList_T)$. On input a *dataID* and a $IDList_T$, the algorithm will scan $IDList_T$, and outputs $b1 = 1$ if the *dataID* appears in the $IDList_T$.
- $(\mathrm{pp}) \leftarrow \mathrm{Setup}(\lambda)$. On input a security parameter λ, the algorithm outputs a public parameter pp, which contains public parameter for zk-SNARKS and some pseudorandom values.

- $(\mathrm{addr_{pk}}, \mathrm{addr_{sk}}) \leftarrow \mathrm{CreateAddress(pp)}$. On input the public parameter pp, algorithm outputs an *address key pair* $\mathrm{addr_{pk}}$ and $\mathrm{addr_{sk}}$.
- $(\mathrm{c}, \mathrm{TX}_{\mathrm{min}t}) \leftarrow \mathrm{Mint}(\mathrm{pp}, v, \mathrm{addr_{pk}})$. On input the public parameter pp, coin value v and destination *address public key* $\mathrm{addr_{pk}}$, the algorithm outputs a new coin c and a *mint* transaction $\mathrm{TX}_{\mathrm{min}t}$.
- $(\mathrm{c}_1^{new}, \mathrm{c}_2^{new}, \mathrm{TX}_{pour}) \leftarrow \mathrm{Pour}(\mathrm{pp}, \mathrm{rt}, \mathrm{c}^{old}, \mathrm{addr}_{\mathrm{sk}}^{old}, \mathrm{path}_{old}, \mathrm{cm}^{old}, v_1^{new}, v_2^{new}, \mathrm{addr}_{\mathrm{pk},1}^{new}, \mathrm{addr}_{\mathrm{pk},2}^{new}, \mathrm{info})$. On input a public parameter pp, a Merkle tree root rt, a coin c^{old}, an *address public key* $\mathrm{addr}_{\mathrm{pk}}^{old}$, a authentication path path_{old} from commitment cm^{old} to root rt, two coin values v_1^{new} and v_2^{new}, two new *address pubic key* $\mathrm{addr}_{\mathrm{pk},1}^{new}$ and $\mathrm{addr}_{\mathrm{pk},2}^{new}$, and some transaction string info, the algorithm outputs two new coin c_1^{new}, c_2^{new} and a *pour* transaction TX_{pour}.
- $(b2) \leftarrow \mathrm{VerifyTransaction}(\mathrm{pp}, \mathrm{TX}_{\mathrm{min}t}/\mathrm{TX}_{pour}, L)$. On input a public parameter pp, a $\mathrm{TX}_{\mathrm{min}t}$ or a TX_{pour} and a ledger L_T in time T, the algorithm outputs $b2 = 1$ if the transaction is valid.
- $(\mathrm{coinsSet}) \leftarrow \mathrm{Recieve}(\mathrm{pp}, \mathrm{addr_{pk}}, \mathrm{addr_{sk}}, L_T)$. On input a public parameter pp, an *address key pair* $(\mathrm{addr_{pk}}, \mathrm{addr_{sk}})$ and a ledger L_T in time T, the algorithm outputs a set of coins coinsSet.

4.3 Smart Contract Construction

Our smart contract is a *HLTC*. In Pour, four random values $(v_i^{new}, \rho_i^{new}, r_i^{new}, s_i^{new})$ are generated for a new coin c_i^{new}, they are required items to use the coin (another required item is $\mathrm{addr_{sk}}$, see algorithm Pour). However, in our algorithm, we didn't reveal them to recipient or public. Which means even if a TX_{pour} was broadcast by *payer* and verified, the *recipient* still can't get the *coin* that *payer* mints for *her*, because the *payer* didn't get data. Therefore, after a TX_{pour} was broadcast, *payer* deploys a smart contract in blockchain to get data. Here, we give the steps of the smart contract deployment.

1. *Payer* input two initial values: $hash(\mathrm{E_{sk}})$, C_v, $\mathrm{addr_{pk}}$. Where $hash(\mathrm{E_{sk}})$ is a hash value of the data's *data encryption private key* $\mathrm{E_{sk}}$, *payer* obtains it before the payment (see Sect. 2); C_v is a ciphertext of the four random value $(v_i^{new}, \rho_i^{new}, r_i^{new}, s_i^{new})$ that generated by *payer* in Pour and encrypt by pk_{enc} of *recipient*.
2. *Payer* deploys such a contract: if someone could provide a preimage of $hash(\mathrm{E_{sk}})$, the provider will be return C_v; the preimage (i.e., $\mathrm{E_{sk}}$) will be encrypted by pk_{enc} of *payer*, and be sent to *payer*.

This smart contract be triggered when someone input a preimage of $hash(\mathrm{E_{sk}})$, namely, $\mathrm{E_{sk}}$. Obviously, because of the property of hash function, only *recipient* can give the preimage, which means only *recipient* could get C_v, and only *recipient* could decrypt C_v, because *she* is the only one who owns the decryption key sk_{enc}. Meanwhile, $\mathrm{E_{sk}}$ is sent to payer after encryption. So far, payer gets *data encryption private key* $\mathrm{E_{sk}}$, *he* can download the encryption data from Super Nodes, and uses $\mathrm{E_{sk}}$ to decrypt it, thus obtaining the row data; As for *recipient*, with C_v, *she* can get the secret values by decrypt C_v using *her* sk_{enc}, in this way, recipient finally get the coin that *payer* mints for *her*, because *she* has all required items: four random values and $addr_{sk}$ targeted at the coin. We show the flow of smart contract in Fig. 3.

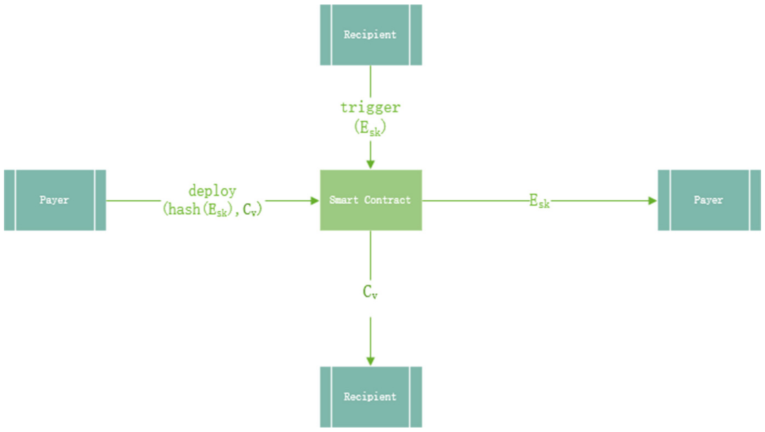

Fig. 3. Smart contract

We use asymmetric encryption in our smart contract to ensure safety. As we describe above, all message be returned and sent are encrypt by the receiver' encryption public key, which means even if the message is eavesdropped by a probabilistic polynomial-time adversary, this adversary has probability Pr[successful decryption] $\leq$ negl.

5 Experiment

To test validity of our scheme, we design several experiments. First, we test the basic algorithm of our scheme, including CreateAddress, Mint, Pour, smart contract and GenerateID. Second, we give three different size of data to test the performance of GenerateID algorithm, because this algorithm's performance with respect to the size of data. Our code is written in java, and all of our experiments were conducted on same machine (Inter Core i5-6300 @ 2.30 GHz with 12 GB of RAM).

Table 1 shows performance of specific algorithms in our scheme, and Table 2 shows performance of correspondent algorithms in Zcash. Similar to Zerocash, we didn't maintain the Merkle tree in our experiment, because that is not responsibility to our algorithm. And note we generate a big result in every basic algorithm of Zerocash, which brings big latency. In reality, the time consumption could be lower. Our smart contract is self-triggered, concretely, we automatically input the preimage of data encryption secret key's hash.

Table 1. Performance of our algorithm

Create address	Time	360 ms
	addr_{pk}	816B
	addr_{sk}	947B
Mint	Time	1.5 ms
	Coin c	1068B
	TX_{mint}	196B
Pour	Time	6 min 1.2 s
	TX_{pour}	2856B
SmartContract	Time	0.3 ms
	E_{sk}	846B
GeneralID	Time	5.6 s
	data	1M

Table 2. Performance of Zcash algorithm

Create address	Time	326.0 ms
	addr_{pk}	343B
	addr_{sk}	319B
Mint	Time	23 µs
	Coin c	463B
	tX_{mint}	72B
Pour	time	2 min 2.01 s
	tX_{pour}	996B[16]

Figure 4 shows performance of GenerateID with different size of data inputting. As it shows, latency of our algorithm grows with the data size grows. Property of hash function results in this performance, the latency will get lower if we change the way to generate dataID.

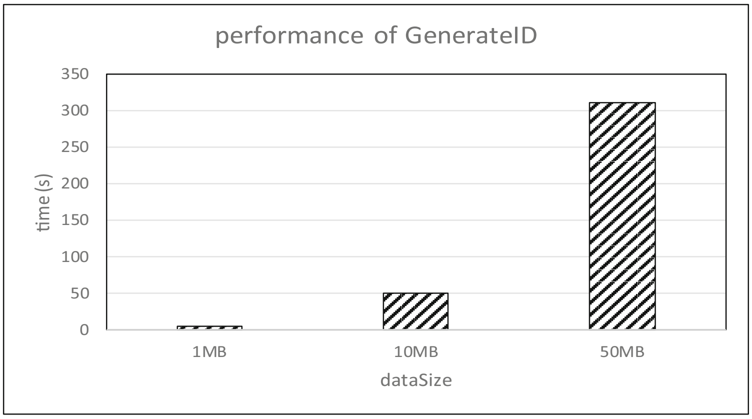

Fig. 4. Performance of algorithm GenerateID

6 Conclusion

We propose a new data transaction scheme that brings decentralization and anonymity for data sellers and data buyers. Concretely, our scheme enables the data transaction be completed only with an information sending by seller and a smart contract deployment by buyer, which is simple and convenient to both sides. Moreover, we introduce the DAP scheme to provide anonymity for transaction, experiment shows that our scheme has almost same performance as Zerocash, which means this is a practical scheme in data transaction.

In the future, we want to improve the performance of our scheme. The first aspect is simplifying transaction process such as eliminating the information sending step for sellers. The second aspect is zk-SNARKs, it takes a long time to compute a proof for each pour transaction, if we can shorten time of this part, we can give a more practical scheme for users among IoV. Moreover, our scheme builds a verification relationship (using zk-SNARKs) rather than trust relationship, which means we can't avoid a failed transaction caused by human behavior. The establishment of trust relationship in system is our important research direction in future.

References

1. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
2. Buterin, V.: A next-generation smart contract and decentralized application platform. white paper (2014)
3. CarBlock: A global transportation data protocol with decentralized applications. white paper (2018)
4. Sasson, E.B., Chiesa, A., Garman, C., et al.: Zerocash: decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy, pp. 459–474. IEEE (2014)
5. Castro, M., Liskov, B.: Practical Byzantine fault tolerance. OSDI, vol. 99 (1999)
6. https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949
7. Parno, B., Howell, J., Gentry, C., et al.: Pinocchio: nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy, pp. 238–252. IEEE (2013)
8. Ben-Sasson, E., Chiesa, A., Tromer, E., et al.: Succinct non-interactive zero knowledge for a von Neumann architecture. In: 23rd USENIX Security Symposium (USENIX Security 14), pp. 781–796 (2014)
9. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_37
10. Parno, B., Raykova, M., Vaikuntanathan, V.: How to delegate and verify in public: verifiable computation from attribute-based encryption. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 422–439. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28914-9_24
11. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 566–582. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_33