Building Private Blockchains over Public Blockchains (PoP): An Attribute-Based Access Control Approach

Dijiang Huang
Arizona State University
Tempe, Arizona
dijiang.huang@asu.edu

Chun-Jen Chung
Arizona State University
Tempe, Arizona
cchung20@asu.edu

Qiuxiang Dong
Arizona State University
Tempe, Arizona, USA
qiuxiang.dong@asu.edu

Jim Luo
US Naval Research Lab
Washington DC, USA
jim.luo@nrl.navy.mil

Myong Kang
US Naval Research Lab
Washington DC, USA
myong.kang@nrl.navy.mil

ABSTRACT

Blockchain technology is increasingly being adopted as a trusted platform to support business functions including trusted and verifiable transactions, tracking, and validation. However, most business use-cases require privacy and confidentiality for data and transactions. As a result, businesses are forced to choose private blockchain solutions and unable to take full advantage of the capabilities, benefits and infrastructure of public blockchain systems. To address this issue, we present an Attribute-Based Encryption (ABE) security solution built on a private-over-public (PoP) blockchain approach. The policy based distributed operation of ABE conforms well to the blockchain concept. The cross-chain PoP approach provides the benefits from both public blockchains and private blockchains. Businesses will be able to restrict access, maintain privacy, and improve performance while still being able to leverage the distributed trust of public blockchains. This paper present the ABE-based security framework and protocol for securing data, transactions as well as smart contracts. Security analysis and performance evaluation show the proposed solution to be effective, efficient and practical. It can greatly reduce the cost and complexity for businesses compared to running isolated private blockchain solutions.

KEYWORDS

Blockchain, Smart Contract, Data Privacy, Access Control, Attribute-Based Encryption

ACM Reference Format:
Dijiang Huang, Chun-Jen Chung, Qiuxiang Dong, Jim Luo, and Myong Kang. 2019. Building Private Blockchains over Public Blockchains (PoP): An Attribute-Based Access Control Approach. In *Proceedings of ACM SAC Conference (SAC'19).* ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/https://doi.org/10.1145/3297280.3297317

1 INTRODUCTION

Cross-chain functionality aims to combine the best features of different blockchain systems [12], both private and public, for the purposes of exchanging value across disconnected ecosystems. Ripple [2] has made notable strides to this effect, with Inter-ledger already testing transactions across multiple ledgers simultaneously in different currencies. ZCash [8] provides privacy protection for Bitcoin [13] users. Hawk [10] and Ekiden [4] have been proposed using off-chain approaches to provide data privacy protection. However, none of existing solutions clearly addressed the problem of applying access control policies to enforce data privacy protection on transaction secrets. For example, when using smart contract solutions, e.g. Ethereum [5], for procurement in supply-chain, transaction parameters such as product name, quantity, price, purchasing terms, shipping options, address, etc. could all be sensitive business secrets. They should be only viewable for relevant stakeholders. Hyperledger [3] addresses this problem by relying on a trust authority approach to build permission groups for data access control. However, data access has to be predefined. It is not suitable for complex and dynamic businesses logic that require dynamic access control. Moreover, traditional infrastructure-based data access control model, e.g., Role-Based Access Control (RBAC) [16], is incompatible with the distributed nature of blockchain operations where transaction data are mobile and shared by multiple blockchain participants.

To address the data access control and privacy protection issues in public blockchain, e.g., Ethereum, we propose a distributed Attribute-Based Encryption (ABE) solution applied to private blockchains over Public blockchains (PoP blockchain, PoP block, or PoP for short) approach. The PoP architecture is presented in Figure 1, where we use ABE-protected state channel and attach multiple private blockchains on a public blockchain. The PoP approach for deploying our ABE solution provides the best of both worlds. Applying ABE on an off-chain basis means it can interoperate with the public blockchain without interference. Private blockchains transactions can be much less computationally intensive and provide superior performance [9] since they do not have to be verified by all participants. Businesses are able to choose the private blockchain solution that best suits their needs independently from the public blockchain. Each private blockchain can be viewed as a protected state channel. The integrity of a private blockchain can be validated

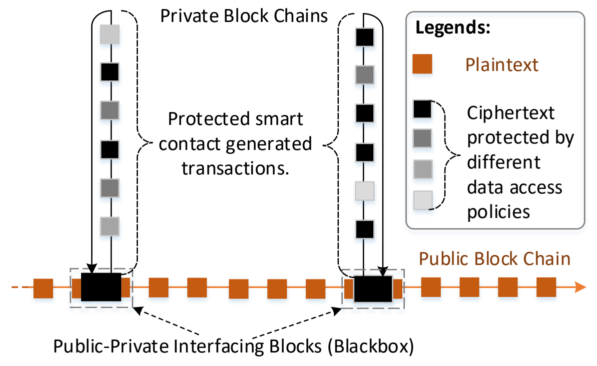
Private Block Chains
Legends:
Plaintext
Ciphertext protected by different data access policies
Protected smart contact generated transactions.
Public Block Chain
Public-Private Interfacing Blocks (Blackbox)

Figure 1: Illustration of PoP blockchain Architecture.

and checked in ciphertext and in aggregate by all public blockchain participants. The public blockchain infrastructure is leveraged to provide validation and immutability for the entirety of the private blockchain state channel. This can take the form of the final private blockchain transaction result, or a hash of the entire private blockchain. Therefore, distributed trust on the publich chain is not necessary for the private blockchain. At the same time, ABE provides data privacy for the private blockchain state channel. Only participants with the appropriate permissions and corresponding ABE attribute private keys can view and validate their relevant blocks in the private block chain. It provides the benefits of private block chains in terms of privacy without requiring the deployment of trusted nodes or multiple verification nodes. It essentially minimizes the entry cost businesses in adopting blockchain solutions.

In summary, the presented PoP solution has the following main contributions:

- We design a decentralized trust model for key management of ABE-based data access control. Using this approach, we can incorporate access control policies into ciphertext to protect content of smart contracts.
- We design a privacy preserving messaging protocol to allow private blockchain participants to interact with the smart contract that can generate a private blockchain. We illustrate how to use this protocol based on a supply-chain procurement application.
- We design two smart contracts: PPP to establish attribute based security trust model and ppSCM to provide secure data access control based on ABE scheme.
- We provide a comprehensive security and performance analysis based on the presented PPP scheme. The presented solution is practical that can significantly reduce the effort and cost to establish dedicated and isolated private blockchains.

The rest of this paper is arranged as follows: Section 2 describes system and models that serve as foundation for this presented solution, and a supply-chain based blockchain solution is highlighted in this section; Section 3.1 presents the details of the proposed PoP solution; the performance evaluation is presented in Section 4; finally, we conclude the work in Section 5.

2 SYSTEM AND MODELS

2.1 BCT for supply-chain

To illustrate the presented solution, in Figure 2, we present a supply chain example based on Block-Chain Technology (BCT), which involves multiple parties, i.e., suppliers, buyers, carriers, IoT Companies, and banks. In the figure, the middle box maintains the constructed blockchains. The potential of having all the information written in a Blockchain allows the creation of an authoritative record that can be used to automatically establish smart contracts. Without such an authoritative record, smart contracts written on a Blockchain could hardly be executed, because parties need to agree on data and information that, like smart contracts themselves, are agreed to by a whole network through a consensus mechanism. The one-layer Blockchain solution sees as such a fully integrated and automated trade network where documents and goods are transparently identified and tracked along the supply chain. Because the information is registered on a distributed database, it makes it tamper-resistant and fosters greater trust in the trade network. The left side of the figure present a purchase related transaction by using Ethereum's Decentralized App (DApp) [1] solution and it involves 4 main procedures based on supply-chain operation procedures:

***Order Processing*:** The order-processing workflow starts with a PO from the buyer. Within the blockchain, once created, the PO is time-stamped and can become a valid document whose clauses can be executed only if valid, due to the programming features of smart contracts. Assuming delivery documents can also be registered on it, the metadata of the invoice, PO and bill of lading could be matched automatically due to the smart contracts feature, which ensures consistency between price and quantity in all three documents (i.e. three-way-match), permitting an automated and fast invoice approval. The entire history of the transactions offers perfect audibility, and trust between parties is provided by the immutability of the data entered in a Blockchain.

***Shipment*:** IoT-based tracking capability is a critical component for this procedure. Keeping track of the material flow at each step, along with the corresponding paper flow, is a major undertaking that requires manual processes that are subject to human error, loss, damage or even theft and fraud. In such a Blockchain-based IoT, there is the possibility of maintaining product information, its history, product revisions, warranty details and end of life, transforming the Blockchain into a distributed and trusted blockchain.

***Invoicing*:** Blockchain-based services can register the invoice-related information on a Blockchain in order to avoid duplicates and fraud across the network. As explained by [7], each invoice would be distributed across the network, hashed and time-stamped in order to create a unique identifier. If a supplier tried to sell same invoice again through the network, that invoice would indicate a previous instance of financing to all parties, and the double financing would be avoided. The integration with the payment system is given by the ability of smart contracts to take control over an asset registered on a Blockchain (e.g. crypto-cash) and automatically trigger the payment.

***Payment*:** Developed to create a purely peer-to-peer version of electronic cash to allow online payments, payments are the first application of BCT. With the use of Bitcoin or similar cryptocurrencies in a B2B scenario, buyer and supplier could transact without any intermediaries (e.g. banks) and with very small transaction fees. Blockchain solutions could create more efficient payment processes between banks, eliminating the need for each institution to maintain and reconcile their own ledger.

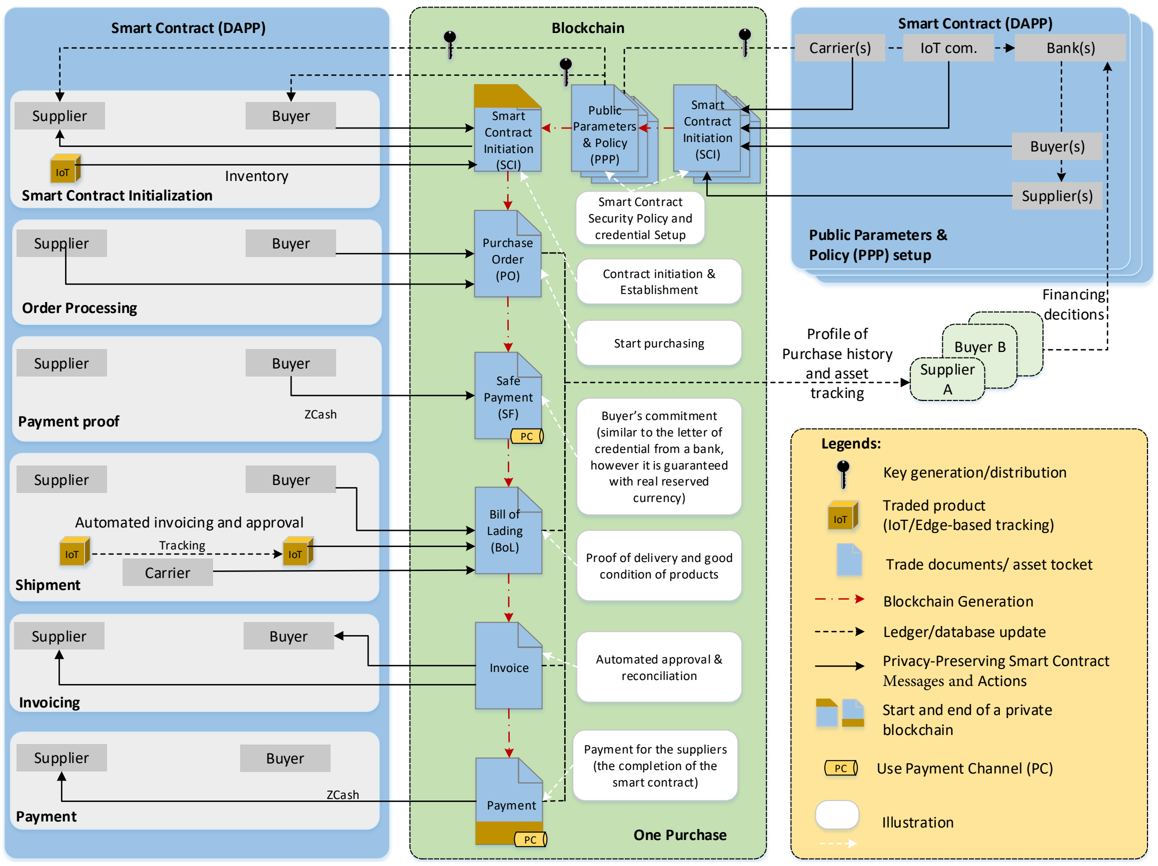
Smart Contract (DAPP)
Blockchain
Smart Contract (DAPP)
Supplier
Buyer
Inventory
Smart Contract Initialization
Supplier
Buyer
Order Processing
Supplier
Buyer
ZCash
Payment proof
Supplier
Buyer
Automated invoicing and approval
Tracking
Carrier
Shipment
Supplier
Buyer
Invoicing
Supplier
Buyer
ZCash
Payment
Smart Contract Initiation (SCI)
Public Parameters & Policy (PPP)
Smart Contract Initiation (SCI)
Smart Contract Security Policy and credential Setup
Purchase Order (PO)
Contract initiation & Establishment
Start purchasing
Safe Payment (SF)
PC
Buyer's commitment (similar to the letter of credential from a bank, however it is guaranteed with real reserved currency)
Bill of Lading (BoL)
Proof of delivery and good condition of products
Invoice
Automated approval & reconciliation
Payment
PC
Payment for the suppliers (the completion of the smart contract)
One Purchase
Carrier(s)
IoT com.
Bank(s)
Buyer(s)
Supplier(s)
Public Parameters & Policy (PPP) setup
Financing decitions
Profile of Purchase history and asset tracking
Buyer B
Supplier A
Legends:
Key generation/distribution
Traded product (IoT/Edge-based tracking)
Trade documents/ asset tocket
Blockchain Generation
Ledger/database update
Privacy-Preserving Smart Contract Messages and Actions
Start and end of a private blockchain
Use Payment Channel (PC)
Illustration

Figure 2: A supply chain scenario using IoT devices, block chain, and data encryption protections.

The above described smart contract is based on traditional supply-chain procurement procedures [7]. However, it does not provide privacy protection for transaction contents processed by smart contracts. In order to provide data privacy protection, we present two additional modules that are incorporated into original supply-chain procedures: (a) *Smart contract initialization*: it sets up the initial smart contract credentials such as agreed data access control policies for each step of smart contract and initiates the off-chain operation, in which we start a private blockchain at this point. (b) *Payment proof*: PoP is a hybrid blockchain solution, in which private block chains are interfaced into public chains. Moreover, the private chain can also incorporate public blockchain evidence into the private blockchain. The addition of the payment proof procedure is to utilize the payment channel [14] feature of public blockchains to prove the buyer has sufficient money to pay for the purchased product. The buyer first pays for the product to a Escrow account, and once the product is landed, the cashed money will be delivered to the supplier to close the blockchain-based purchase.

In the presented model, both IoT companies and Banks play a crucial role. Relying on IoT companies' tracking capability, Banks can use the traceability nature of blockchains. By gaining access control to protected business transaction data, Banks can monitor the healthiness of business entities for credit evaluation loan decision making. Due to the page limits, this research focuses on the data privacy protection and skips the details of Banks and IoT companies involved smart contract and blockchain related activities.

2.2 Smart Contract

Ethereum smart contract is to build a decentralized application to create a blockchain with a build-in Turing complete programming language. Therefore, smart contract means is defined to be a cryptographic "boxes" that contain value and only unlock it if certain conditions are met. The same as a transaction, a smart contract will also be stored in the blockchain and can be retrieved by its address and integrity can be guaranteed as well. To trigger a smart contract is just like a remote processor call. The input would be included in the transaction. That is, smart contract creation, smart contract function call and smart contract destroy are all included in a transaction. With smart contract, one can express logics such as "only after April 17th, 2018, can the document be sent to A". The smart contract is running in an Ethereum Virtual Machine (EVM), and the smart contract involved user interactions and data processing modules are usually running by the DApp.

In the presented supply-chain example shown in Figure 2, there are two smart contracts involved: (1) *public blockchain smart contract*: the smart contract on the right side box includes multiple stake holders providing supply-chain services to settle down a PPP (Public Parameters and Policies). A PPP describes what encryption public parameters will be used for data privacy protection, who may serve as a trusted party for data access control management for running private blockchains, and what security policies to be enforced in the private blockchain. We can treat a PPP as a "template" that can be reused to build a private blockchain. Thus, multiple PPPs can be generated for different use cases of private blockchains. (2) *Private blockchain smart contract*: the second smart contract on the left side box in Figure 2 represents a one purchase between a supplier and a buyer. In addition, an IoT company can be involved to provide product tracking and inventory.

2.3 Attribute-Based Encryption (ABE) Enabled ABAC

In the literature, a large numbers of Attribute-Based Encryption (ABE) solutions have been proposed. In this paper, we propose an extended Lewko's scheme [11] by adding distributed trust management to allow multiple parties to collaboratively establish the trust and distribute secret keys. The following described federated Authority Setup and Federated KeyGen protocols are newly proposed. The details of the extended Lewko's approach is presented in Appendix A. Due to the page limit, interested reader can refer to Lewko's [11] for security proofs. Our approach has the following salient features compared to existing blockchain data privacy protection solutions:

- It is distributed and mobile, i.e., every participant in the system can serve as a trust authority to issue attributes and corresponding private keys for private blockchain participants; the access control policy is associated with ciphertext, which can be freely shared among blockchain stakeholders without needing an access control infrastructure for data management.
- It is federated, i.e., attributes can be shared among private blockchain participants. This also means that the scheme allows a coalition to be established for a private blockchain for attributes and corresponding private keys generation. The coalition can prevent single point failure issue as well resisting to $n-1$ collusion problem, where n is the size of the coalition.
- It provides interoperability feature, i.e., attributes and corresponding private keys generated from different trust authorities can be used together to form a data access control policy. For example, Alice can use her own generate private key for attribute A_1 and another attribute A_2, which is generated by Bob to decrypt a data protected by data access policy enforced by the policy $\{A_1 \ \ AND \ \ A_2\}$.

A typical security policy should include multiple descriptive terms (i.e., attributes) such as:

$$\mathcal{P}1 = \text{The } \underline{\text{pricing}} \text{ and } \underline{\text{quantity}} \text{ can be accessed by}$$
$$\text{the } \underline{\text{supplier}} \text{ and the } \underline{\text{buyer}}.$$

In this policy, 'pricing' and 'quantity' are accessing objects, and 'supplier' and 'buyer' are attributes describing accessing subjects. These attributes can be used as public encryption keys. In each block created by the private blockchain, data are encrypted by using one or multiple data access policies.

The policy $P1$ is presented as a tree structure in Figure 3, which is called Policy Tree (PT). A PT is constructed by attributes at

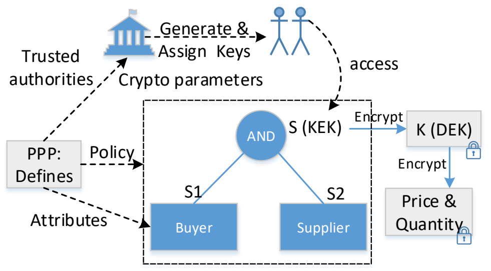

Figure 3: Attribute (or Policy) based access control setup.

the leaves and intermediate nodes are logical gates. Using secret sharing scheme, a tree-root level secret s can be used as a Key-Encrypting-Key (KEK) to protect at symmetric key K as the Data Encrypting-Key (DEK) to protect data such as the values of price and quantity. The smart contract generated PPP defines attributes and policies for a particular application, and trusted parties, and crypto parameters used for key generation and encryption in the private blockchain.

The following ABE functions are used in the presented ABAC data protection models, and their detailed constructions are presented in Appendix A.

Global Parameters Setup$(\lambda) \rightarrow GP$: The Global Parameters (GP) can be established in advance by a well-known organization, e.g., in the supply-chain industry. Since the GP is publicly known, it is not critical for which party to generate the GP. The organization selects a composite bilinear group G or order $N = p_1p_2p_3$. $GP = \{N, g_1, H : \{0,1\}^* \rightarrow G\}$, where g_1 is a generator of group G_{p_1} and the hash function H is mapping function that maps a global identifier to an element of group G. This algorithm might be run multiple times by different entities so as to generate multiple candidate global parameters in the candidate public parameters and policies (PPP). □

Authority Setup$(GP) \rightarrow MPK, MSK$: Each blockchain participant can serve as a trusted authority for key generation. Based on the GP, they need to choose and publish a set of public parameters, i.e., a master public key MPK, which can be later used for private key generation. For each attribute A_i that is managed by the authority, the authority j chooses randomly $\alpha_i, y_i \in \mathcal{Z}_N$ and publishes $MPK_j = \{e(g_1, g_1)^{\alpha_i}, g_1^{y_i}, \forall i\}$ as the public key. The corresponding master private key is $MSK_j = \{\alpha_i, y_i \ \forall i\}$. □

Federated Authority Setup$(GP, AAS) \rightarrow \hat{MPK}, \hat{MSK}$: Using Lewko's scheme, each authority can generate a private key for a given attribute. However, for each attribute, it requires every user to derive their private keys from the same authority. Thus, the authority must be fully trusted. To relax this requirement, we need to involve multiple authorities for private key generation to prevent single point failure issue. If there are n federated authorities, then this scheme is resistant to $n-1$ authority collusion problem. The federated authority setup algorithm is run when multiple attribute authorities need to generate the public key and private key for their shared attribute(s). For simplicity, we assume there are n attribute authorities in the set AAS, *i.e.,* $AAS = \{AA_1, AA_2, \cdots, AA_n\}$. AA_i will generate $\alpha_i, y_i \in \mathcal{Z}_N$. Each AA_i will generate an individual master private key $MSK_i = \{\alpha_i, y_i \forall i\}$. AA_{i-1} will send the individual master public key to AA_i as:

$$MPK_{i-1\to i} = \{e(g_1,g_1)^{\sum_{j=2}^{i}\alpha_{j-1}}, g_1^{\sum_{j=2}^{i} y_{j-1}}\},$$

AA_i will calculate

$$MPK_{i\to i+1} = (e(g_1,g_1)^{\sum_{j=2}^{i}\alpha_{j-1}})^{\alpha_i}, (g_1^{\sum_{j=2}^{i} y_{j-1}})^{y_i}.$$

The final federated master public key and private key for an attribute is defined as follows:

$$M\hat{P}K = \{e(g_1,g_1)^{\sum_{j=1}^{n}\alpha_j}, g_1^{\sum_{j=1}^{n} y_j}\},$$

$$M\hat{S}K = \{\sum_{j=1}^{n}\alpha_j, \sum_{j=1}^{n} y_j\}.$$

□

Encrypt$(M,(A,\rho),GP,\{MPK\},M\hat{P}K) \to CT$: M is a message, A is an $n\times\ell$ access matrix and ρ maps its rows to attributes. For each row in A, the algorithm chooses a random number $r_x \in \mathcal{Z}_N$. A random vector $w \in \mathcal{Z}_N^\ell$ with 0 being the first entry is chosen randomly. ω_x denotes $A_x \cdot \omega$. The data owner chooses $s \in \mathcal{Z}_N$ and a vector $v \in \mathcal{Z}_N^\ell$ randomly where s is its first entry. $\lambda_x = A_x \cdot v$ with A_x being the x^{th} row of the matrix A. The ciphertext is as follows:

$$CT =< C_0, C_{1,x}, C_{2,x}, C_{3,x} >, \ where$$
$$C_0 = Me(g_1,g_1)^s, C_{1,x} = e(g_1,g_1)^{\lambda_x} e(g_1,g_1)^{\alpha_{\rho(x)} r_x},$$
$$C_{2,x} = g_1^{r_x}, C_{3,x} = g_1^{y_{\rho(x)} r_x} g_1^{\omega_x}, \forall x.$$

CT is the ciphertext of DEK, which is used to encrypt data object o in the blockchain protocol. In this encryption protocol, the encryptor needs to identify which master public key parameters are used for each involved attribute. Later, a decryptor can use private keys generated from corresponding public keys. □

KeyGen$(GID, i, \{MSK\}, GP) \to SK_{i,GID}$: In PoP, the GID can be an address that is used to identify the blockchain participant. For a global identifier GID with attribute i belonging to an authority, the authority generates the following private key

$$SK_{i,GID} = g_1^{\alpha_i} H(GID)^{y_i}.$$

Using the *KeyGen* scheme, an authority can generate private keys for other blockchain participants. □

Federated KeyGen$(GID, i, \{MSK\}, M\hat{S}K, GP) \to SK_{i,GID}$: The federated key generation algorithm is run when multiple attribute authorities need to generate the private key for an attribute shared among multiple users. Assume that $AAS = \{AA_1, AA_2, \cdots, AA_n\}$. AA_i will generate $g_1^{\alpha_i} H(GID)^{y_i}$. AA_{i-1} will send AA_i the private key component:

$$SK_{i-1\to i} = g_1^{\sum_{j=2}^{i}\alpha_{j-1}} H(GID)^{\sum_{j=2}^{i} y_{j-1}},$$

then, AA_i will calculate

$$SK_i = (g_1^{\sum_{j=2}^{i}\alpha_{j-1}})^{\alpha_i} (H(GID)^{\sum_{j=2}^{i} y_{j-1}})^{y_i}.$$

The final secret key of the shared attribute for the user GID is as follows.

$$SK_{i,GID} = g_1^{\sum_{j=1}^{n}\alpha_j} H(GID)^{\sum_{j=1}^{n} y_j}.$$

□

Decrypt$(CT, \{SK_{i,GID}\}, GP) \to M$: Assume that the ciphertext is encrypted under an access matrix (A,ρ). If the decrypt holds the private key $\{SK_{\rho(x),GID}\}$ for a subset of rows A_x of A satisfying that $(1,0,\cdots,0)$ is in the span of these rows, then the plaintext message M can be obtained in the following way.

$$C_{1,x} \cdot e(H(GID), C_{3,x}) / e(SK_{\rho(x),GID}, C_{2,x})$$
$$= e(g_1,g_1)^{\lambda_x} e(H(GID), g_1)^{\omega_x}.$$

The decryptor chooses constants $c_x \in \mathcal{Z}_N$ so that $\sum_x c_x A_x = (1,0,\cdots,0)$ and computes

$$\prod_x (e(g_1,g_1)^{\lambda_x} e(H(GID),g_1)^{\omega_x})^{c_x} = e(g_1,g_1)^s.$$

To verify the transaction including encrypted data, the decryption algorithm will be called. □

2.4 Security Model

PoP assumes that blockchain participants are curious, selfishness, and greedy, and they want to learn business secrets incorporated into blockchains. They may collude to share their secrets to gain additional data access capabilities that should not be assigned to them. Moreover, they may drop off from blockchain creating procedure to take the goods without paying for it.

3 POP SYSTEM MODELS

3.1 Access System Construction

Definition 1 (Contract). *A smart contract C is defined by a procedure $C = \{T\}$ that specifies a set of interdependent transactions among contract subjects (or participants) in group G.* □

Definition 2 (Transaction). *A transaction defines a sequential atomic data actions $\{a\} \in A = \{read, write, change\}$, and each action a is restricted by a privilege $\alpha(a,T) : \mathcal{P}(a,T) \mapsto S_{a,T}$, where the privilege $\alpha(a,T)$ are defined by capabilities such as* {can, cannot, restricted by/to} *of the action a in the transaction T. The privilege is described in the security policy $\mathcal{P}(a,T)$, and the privilege can be mapped/translated to $S_{a,T}$ denoting the subset of subjects in the overall participating group G.* □

Definition 3 (Subject). *A subject $s \in S$ or G (here S is a subset of subjects and G is the overall group includes all the subjects) is an entity involved in smart contract that can perform actions, and each subject has a data access privilege described by a policy $\mathcal{P}(s) : \cup_{\forall(a,T)\rightarrowtail s}\mathcal{P}(a,T)$, in which $\cup_{\forall(a,T)\rightarrowtail s}$ denotes the union for all action and transaction pair (a,T) it involves ($\rightarrowtail$ is an "involve" operator) the subject s. It is the collection of policies for a smart contract involving the subject s. Here, we denote $\mathcal{P}(s) = \cup_{\forall(a,T)\rightarrowtail s}\mathcal{P}(s,a)$ and $\mathcal{P}(s,a)$ is the data privacy policy for action a. Correspondingly, $\mathcal{P}(S)$, $\mathcal{P}(S,a)$ are defined for S as a subset of subjects.* □

Definition 4 (Object). *An object $o \in O$ represents a data (or a file, a piece of information) that subjects want to access to perform actions such as* read, write *and* change. *The access policy to an object o for a smart contract $P = \{T\}$ is represented as $\mathcal{P}(o) = \cup_{\forall(a,T)\rightarrowtail o}\mathcal{P}(o,a)$, which represents the collection of data access policies involved with all actions on an object o.* □

Definition 5 (Data Access Capability). *The following access policies are defined:*

$T : s \longrightarrow o|_{\{a\}:*}$:	*in transaction T, subject s can operate action(s) $\{a\}$ limited by condition $*$ on object o under access policies $\mathcal{P}(o, a) \subseteq \mathcal{P}(s, a)$;*
$T : S \longrightarrow o|_{\{a\}:*}$:	*in transaction T, subset of subject S can operate action(s) $\{a\}$ limited by condition $*$ on object o under access policies $\mathcal{P}(o, a) \subseteq \mathcal{P}(S, a)$;*

where $*$ is a condition to confine action(s) such as in a transaction T or in a contract C. □

3.2 Privacy-Preserving Messaging Protocol (PPMP)

Goal: For a given smart contract transaction T, one or a set of subject(s) $S \subseteq G$ can perform an action a, where $\mathcal{P}(S, a) \neq \mathcal{P}(\bar{S}, a)$, in which the collective privilege (e.g., by colluding) of set $\bar{S}$ cannot satisfy the privilege given by subset S. This means that the data access policy $\mathcal{P}(S, a)$ can only be satisfied by the subgroup of participants in S. Thus, the *PPMP* protocol describes the data access privilege that only a subset of participants S can perform an action $a \in \{read, write, change\}$ in a transaction T. □

Messages: In order to implement the access control privilege associated to an action in a smart contract, we use cryptographic approaches. Let's define three cryptographic enforcement operations:

$$c = \{Encryption(E), Decryption(D), Signature(Sig)\},$$

which can be applied to an action a in the smart contract transaction to enforce a security policy $\mathcal{P}$ for one or multiple subject(s).

For an or multiple action(s) $\{a\}$ in a transaction T, a *PPMP* message is defined as:

$$PPMP(T, \{a\}, \mathcal{P}, c, s, o) = T : s \longrightarrow o|_{\{a\}:\mathcal{P},c}; \quad (1)$$

$$\text{or}$$

$$PPMP(T, \{a\}, \mathcal{P}, c, S, o) = T : S \longrightarrow o|_{\{a\}:\mathcal{P},c}. \quad (2)$$

Based on the above definition, the *PPMP* protocol is actually the implementation of data access capabilities defined in Definition 5 through conditions enforced by cryptographic operations $c = \{E, D, Sig\}$.

3.3 Smart Contract Protocols

In this section, we present two smart contract protocols presented in the supply-chain example of Figure 2, namely, PPP contract and ppSCM (privacy-preserving Scheme) contract.

3.3.1 *PPP Contract.*

Shown in Contract 1 in Appendix A.1, *PPP* is a smart contract created by an initiator on the public blockchain. The initiator could be a Trusted Authority who wants to negotiate attributes, global public parameters and policies with all other participants for using attribute-based encryption scheme and policy-based access control in a private blockchain. An attribute authority needs to call function *join* in the *PPP* first to join the negotiation and insert their attributes into the smart contract. The smart contract will form policies to be negotiated based on the inputs from all joined parties. The negotiation is a voting process on the collected policies and allow each joined party to vote at most once to select their preferred policy.

3.3.2 *Privacy-preserving scheme (ppSCM) Contract.*

ppSCM contract is shown in Contract 2 in Appendix A.2, which is a smart contract created by a supplier on a dedicated private blockchain. It uses the negotiated policy from *PPP* on public blockchain to create a contract for transactions involving supplier, buyer and carrier on the new created permission-based private blockchain. The permission to access the private blockchain is controlled by the access control policy.

ppSCM will maintain purchase orders and invoices for the same buyer and supplier on a dedicated private blockchain. The data in purchase orders and invoices are protected by the selected access policy from *PPP*. Only the parties with appropriated attributes can query or update the data via transactions.

When a buyer would like to purchase products from a supplier, the supplier deploys *ppSCM* smart contract exclusively for the buyer's account. The buyer then put the purchase order on the supplier's *ppSCM* with product name and quantity by calling *sendOrder* function. Through an event, so-called *OrderSend*, the supplier could receive the order data and process it.

After received the purchase order, the supplier looks for the best shipping price on the carrier's smart contract. He then sends the order price and shipment price to the buyer by calling *sendPrice* and the buyer receives this through the event called *PriceSent*.

The buyer performs the safe payment of the grand total (order price + shipment price) through the smart contract in the public blockchain by the *SafepaySent*() event in the *ppSCM*. These coins go to the smart contract account and wait there until the delivery.

After safe payment, the supplier sends the invoice with delivery date and some other data to the buyer by calling *sentInovice*. The buyer receives the invoice data through the event called *InvoiceSent*.

The carrier, after delivery the order to the buyer, marks the order as delivered on the *ppSCM* smart contract by calling *delivery* function. The event *OrderDelivery* then call a smart contract in the public blockchain payout the supplier for the order and payout the Courier for the shipment.

3.4 Policy-Based Data Privacy Protection

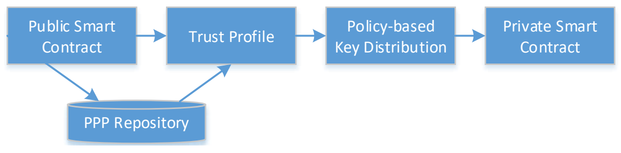

Figure 4: Trust and policy management Procedure.

3.4.1 *PPP: Public Parameters and Policies.*

As shown the PoP example in Figure 2, we can abstract the In the PoP's trust and policy management procedure in Figure 4. For each private blockchain construction, we need to set up or choose a trust profile, i.e., either derived from a public smart contract to generate a PPP or reuse a previously establish PPP. A PPP is built based on the following data structure:

- *D1*: A set of Global Parameters (GP, see the *global parameter setup*) provides the global parameters that all ABE users need to use for key generation, encryption, and decryption.

- *D2*: An array of identities of authorities ($\{GID\}$) and their associate public parameters $\{MPK\}$ and/or federated public parameters $\hat{MPK}$, and each parameter associated attributes $\{MPK\}, \hat{MPK} \longrightarrow \{A\}$. The mapping between public parameters and attributes allow each private blockchain participant to select which public parameters to use in the ABE **Encrypt** procedure.
- *D3*: A set of policy examples that can be used for each of smart contract transaction during the private blockchain construction.

PPP is built using smart contract over public blockchain (see Contract 1), and thus they are searchable. A public directory service can be used to store established PPP. For convenience, each PPP can be reused as a template for a new private blockchain construction. The *GID* is a public blockchain address, which is usually generated from a self-created public key. in D2, exposing *GID* will not reveal the real blockchain participant's identity. In many real business scenario, suppliers may prefer to expose their real identity for easier key management procedure after the PPP establishment. In addition, when creating a new private blockchain, the participant can initiate an ***update smart contract*** to update the authority list and associated attributes, which can be implemented using the PPP creation smart contract (see Contract 1).

3.4.2 Private Key Distribution.

Once a PPP is determined and trusted authorities are known, a private key distribution procedure is conducted as an off-chain procedure. The key distribution can be initiated by either a private blockchain participant or a trust authority. Using existing public key exchange protocols can allow the participants to derive private keys corresponding to each assigned attribute. Some of the attributes may need to get a capability certificate from a trusted authority when applying for a private key. A capability certificate is usually a digitally signed document to prove the requester has the capability to conduct a business function, e.g., professional certificates, bank certificates, business type certificates, etc. Each certificate should be digitally signed by wellknown certificate issuers on requestor's *GID*. Then, the trusted authorities can use **KenGen** or **Federated KeyGen** to generate private keys for distribution. The key distribution is an off-chain procedure and can be done offline. Any existing public key or shared key-based key distribution schemes can used, in which details are omitted in this paper.

3.4.3 Data Object Encryption and Decryption.

The data access protocol is specified in PPMP protocol. The data object operation diagram is presented in Figure 5. Both DApp (a web-based app) and smart contract (running within an Ethereum Virtual Machine (EVM)) run locally on a user's site, and the ABE encryption/decryption engine is also interfaced to the DApp locally. The encryption and decryption are performed between the DApp and the blockchain, and encryption/decryption engine.

In this project, we consider the data object granularity is determined by access control policies. Using the same data access control policy provided in Section 2.3: $\mathcal{P}_1$=*The pricing and quantity can be accessed by the supplier and the buyer.* $\mathcal{P}_1$ is an example to specify the data protection when creating the PO. The PPMP message will be used by DApp, which runs an ABE encryption and decryption

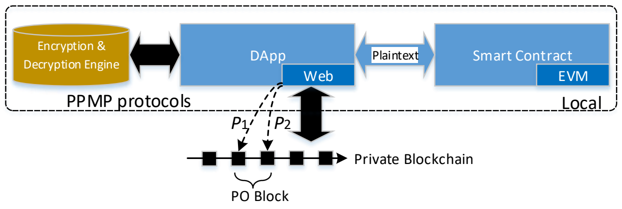

Figure 5: Data object encryption and decryption.

engine to create the PO blockchain block. Thus (2) can be written as:

$$\begin{aligned} & PPMP(T, \{a\}, \mathcal{P}, c, s, o) \\ = \; & T_{PO} : \{buyer, supplier\} \longrightarrow price\&quantity|_{create:\mathcal{P}_1, E}. \end{aligned}$$

PO should also contain an address information for shipment. If the data access policy is $\mathcal{P}_2$=*The shipment address information can be accessed by the buyer and the carrier.*, then the PPMP message can be:

$$\begin{aligned} & PPMP(T, \{a\}, \mathcal{P}, c, s, o) \\ = \; & T_{PO} : \{buyer, carrier\} \longrightarrow shipping\ address|_{create:\mathcal{P}_2, E}. \end{aligned}$$

The PO example presents two data access control policies $\mathcal{P}_1$ and $\mathcal{P}_2$ are involved, and thus on the blockchain, there should be at least two transactions corresponding to $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. The granularity of encrypted block on the blockchain is determined by using the same data access control policy without needing to create two different DEKs. Technically, we can combine $\mathcal{P}_1$ and $\mathcal{P}_2$ as one policies, however, there is no way to use one DEK to protect the data content to fulfill both of them. Other crypto actions such as decryption can be similarly created based on the PO example. We note that the crypto currency involved functions can be achieved by using public blockchain's payment channel approach [14]. Due to page limits, we do not provide details in this paper.

4 POP OPERATION AND PERFORMANCE ANALYSIS

4.1 Complexity Analysis

The following table summarizes the complexity of the algorithms of the proposed scheme. Here, only computation performed on each individual attribute authority will be counted. Therefore, the complexity of (setup, federated setup) and (key generation, federated key generation) is the same. Thus, we only show Setup and KeyGen to represent these two. **E** and **P** represents exponentiation

Table 1: Computation Complexity Comparison in terms of the Number of Pairing Operations

Schemes	Complexity
Setup	$2\lvert U_i\rvert$
KeyGen	$2\lvert S\rvert$
Encrypt	$5n\mathbf{E} + 1$
Decrypt	$2n\mathbf{P} + n\mathbf{E}$

and pairing respectively. U_i indicates the set of attributes managed by a certain attribute authority. S represents the set of attributes assigned to a user. n is the number of rows of the linear secret sharing matrix used in the encryption and decryption algorithm. The numerical evaluation of the presented ABE scheme is presented in Appendix ??, in which it shows the involved computations are at million seconds level and they can be done easily for modern computers.

4.2 Security Analysis

The proposed ABE scheme is based on Lewko's scheme from [11]. For interested readers, please refer to Lewko's work for security proof, in which the scheme is secure against both multiple (fewer than $n - 1$) trusted authority collusion attack and collusion among users. We extend Lewko's scheme from single authority setup and key management to **Federated Authority Setup** and **Federated KeyGen**. The remaining work we need to do is to prove the federated setup and key generation algorithm does not cause security issue and break the collusion problem provided by Lewko's scheme.

Basically, if an adversary wants to compromise the system during the federated setup and private key generation, what he/she wants to obtain is the value of $\sum_{j=1}^{n} \alpha_j$ and $\sum_{j=1}^{n} y_j$. Because the discrete logarithm problem is difficult in terms of a prime group that is big, the adversary cannot obtain each individual value α_j and y_j. The only way to do this is attribute authority collusion. However, it is only when all of the attribute authority collude together can the private secret get leaked. Therefore, if the number of colluding attribute authority is $n-1$ or fewer than that, our proposed federated algorithms are resistant against collusion attacks.

4.3 Comparison with Previous Work

Several existing projects, e.g., Hyperledger [3], R3CEV's Corda [18], and the Gem Health network [15] provide private blockchain solution for business. The idea of cross-chain functionality is to enjoy the benefits from both public and private blockchains, in which solutions bent on delivering cross-chain functionality mean that many of the existing obstacles currently governing the exchange of value will gradually fade. In effect, cross-chain functionality could gather together the best features of blockchains [12], both private and public for the purposes of exchanging value across disconnected ecosystems. Ripple [2] has already made notable strides to this effect, with Interledger already testing transactions across multiple ledgers simultaneously in different currencies. ZCash [8] provides privacy protection for Bitcoin [13] users. Hawk [10] and Ekiden [4] have been proposed using off-chain approaches to provide data privacy protection. However, none of existing solutions clearly addressed how to apply access control policies to enforce data privacy protection on transaction secrets.

The following Table 2 summarizes the main feature comparisons with existing several major privacy-preserving solutions.

Table 2: Blockchain Feature Comparison

Solution	Protect	Method	Compu.	Type	Access Policy
PoP	smart contract	off-chain execution & ABE	Medium	no limit	Yes
Hawk [10]	smart contract	off-chain execution & on-chain zkSNARK	heavy	no limit	No
Ekiden [4]	smart contract	off-chain Hardware Tee	low	no limit	No
Maxwell [6]	amount	on-chain	n/a	Bitcoin	No
ZeroCash [17]	identity	on-chain	n/a	Bitcoin	No

5 CONCLUSION

In this paper, we presented a blockchain solution on how to build private blockchains over public blockchain, called PoP. A set of messaging and smart contract protocols are also presented to illustrate privacy-preserving functions of PoP. We use a supply-chain procurement procedure example to illustrate how PoP works.

Blockchain technologies for supply-chain and other business functions are emerging research and development areas. This presented work may lead to many research and development directions for the next step. First, more functional-rich policy-based access control solutions should be considered. The existing solution is based on Lewko's solution. Other features such as attribute and user's revocation should be considered; policy/attributes expiration should be also considered that allow more automatic policy-based access control features, etc. Second, the presented smart contracts only focus on PPP establishment and procurement. Other smart contracts such as cancellation/revocation of a contract should be also investigated. Third, we briefly discussed on how to use IoT device and how Bank can monitor blockchain based transactions to allow them to decide business loan credibility of business parties. More in-depth investigation is required in our future work.

ACKNOWLEDGMENT

All authors are gratefully thankful for research grants from Naval Research Lab N00173-15-G017, National Science Foundation âĂŞ US DGE-1723440, OAC-1642031, SaTC-1528099, and National Science Foundation âĂŞ China 61628201 and 61571375.

REFERENCES

[1] Andreas Bogner, Mathieu Chanson, and Arne Meeuw. 2016. A decentralised sharing app running a smart contract on the ethereum blockchain. In *Proceedings of the 6th International Conference on the Internet of Things.* ACM, 177–178.

[2] Vitalik Buterin. 2016. Chain interoperability. (2016).

[3] Christian Cachin. 2016. Architecture of the Hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers.*

[4] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2018. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contract Execution. *arXiv preprint arXiv:1804.05141* (2018).

[5] Ethereum. 2018. Ethereum Project. available at https://www.ethereum.org/. (2018).

[6] G. Maxwell. 2018. Project gmaxwell. available at https://github.com/gmaxwell. (2018).

[7] Erik Hofmann, Urs Magnus Strewe, and Nicola Bosia. 2017. *Supply Chain Finance and Blockchain Technology: The Case of Reverse Securitisation.* Springer.

[8] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2016. *Zcash protocol specification.* Technical Report. Tech. rep. 2016-1.10. Zerocoin Electric Coin Company.

[9] Rami Khalil and Arthur Gervais. 2017. Revive: Rebalancing off-blockchain payment networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 439–453.

[10] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Security and Privacy (SP), 2016 IEEE Symposium on.* IEEE, 839–858.

[11] Allison Lewko and Brent Waters. 2011. Decentralizing attribute-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 568–588.

[12] Joe Liebkind. 2018. Public vs Private Blockchains: Challenges and Gaps. available at https://www.investopedia.com/news/public-vs-private-blockchains-challenges-and-gaps/. (2018).

[13] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[14] Joseph Poon and Thaddeus Dryja. 2016. The bitcoin lightning network: Scalable off-chain instant payments. *draft version 0.5* 9 (2016), 14.

[15] J Prisco. 2016. The Blockchain for healthcare: Gem launches Gem Health Network with Philips Blockchain Lab. *Bitcoin Magazine* (2016).

[16] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. 1996. Role-based access control models. *Computer* 29, 2 (1996), 38–47.

[17] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 459–474.

[18] Eric Wall and Gustaf Malm. 2016. Using blockchain technology and smart contracts to create a distributed securities depository. (2016).

A SMART CONTRACT SUDO CODES

In this appendix, we present sudo codes of two smart contract protocols presented in the supply-chain example of Figure 2, namely, PPP contract and ppSCM contract.

A.1 Pseudo-code of PPP Contract

The *PPP* contract is created by an initiator on the public blockchain. An attribute authority needs to call function *join* in the *PPP* first to join the negotiation and insert their attributes into the smart contract. The smart contract will form policies to be negotiated based on the inputs from all joined parties. The negotiation is a voting process on the collected policies and allow each joined party to vote at most once to select their preferred policy.

Contract 1 Pseudo-code of PPP Contract

```
1:  struct $AA = \{\ldots\}$                         ▷ Define Attribute Authority
2:  $AAS = \{AA_1, \ldots, AA_n\}$                 ▷ Address of each joined $AA$
3:  struct $P = \{\ldots, count\}$                 ▷ Define Policy and their count
4:  $PS = \{P_1, \ldots, P_n\}$                   ▷ A set of policy to be negotiated
5:  function PPP(P $p$)
6:      $PS$.push(new $P(p)$)                      ▷ Initiator create a new policy
7:  function Join($Attribute\ attr$)
8:                    ▷ Initiator grants a new AA to join the negotiation
9:      require(isAllow($msg.sender$)
10:     require($msg.sender \notin AAS$)
11:     $AAS$.push(new $AA(msg.sender, attr)$)
12:     if ($AA.attributes \notin PS$) then
13:         add $AA.attributes$ to $PS$
14: function Vote($P\ p$)
15:     require(($msg.sender \in AAS$)
16:     require(not isVoted($addr\_AA$))
17:     $PS$.find($p$).$count^{++}$
18: function getPolicies( )
19:     require($msg.sender \in AAS$)
20:     $Policies = \emptyset$
21:     for each $p \in PS$ do
22:         $Policies$.push($p$)
23:     return $Policies$
24: function getPPP( )
25:     require($msg.sender \in AAS$)
26:     return $PS$.findMaxVote()
```

A.2 Pseudo-code of ppSCM Contract

The *ppSCM* contract is created by a supplier on a dedicated private blockchain. It uses the negotiated policy from *PPP* on public blockchain to create a contract for transactions involving supplier, buyer and carrier on the new created permission-based private blockchain. The permission to access the private blockchain is controlled by the access control policy. *ppSCM* will maintain purchase orders and invoices for the same buyer and supplier on a dedicated private blockchain. The data in purchase orders and invoices are protected by the selected access policy from *PPP*. Only the parties with appropriated attributes can query or update the data via transactions.

Contract 2 Pseudo-code of ppSCM Contract

```
1:  address $addr\_Seller$, $addr\_Buyer$
2:  struct $PO = \{goods, quantity, number, price, safepay, shipment\}$
        ▷ Purchase Order Data
3:  struct $Shipment = \{courier, price, safepay, payer, date\}$
4:  struct $Invoice = \{orderno, invoiceno\}$
5:  $AccessPolicy = \emptyset$
6:  $Orders = \emptyset$, $Invoices = \emptyset$
7:  $orderseq = 0$, $invoiceseq = 0$
8:  function ppSCM(address $buyerAddr$, $Policy\ AccP$)
9:                                         ▷ Initialize contract by a seller
10:     $addr\_Seller = sender$
11:     $addr\_Buyder = buyerAddr$
12:     $AccessPolicy = AccP$
13: function sendOrder(string $good$, unit $quantity$)
14:                                          ▷ Buyer send a PO to the seller
15:     require($msg.sender == addr\_Buyer$)
16:     $Orders$.push(new $PO(good, quantity, orderseq^{++})$)
17:     OrderSent()
18: function sendPrice($PO\ po$, unit $priceP$, unit $priceS$)
19:               ▷ Seller send prices (order and shipment) to buyer
20:     require($msg.sender == addr\_Seller$)
21:     require(isValid($po$))
22:     $po.price = priceP$
23:     $po.shipment.price = priceS$
24:     PriceSent()
25: function sendSafepay($PO\ po$)
26:                                   ▷ Buyer send safe payment to seller
27:     require($msg.sender == addr\_Buyer$)
28:     require(isValid($po$))
29:     SafepaySent()
30: function sendInvoice($PO\ po$, unit $date$, address $courier$)
31:          ▷ Seller send invoice to Buyer and trigger the shipment
32:     require($msg.sender == addr\_Seller$)
33:     require(isValid($po$))
34:     $Invoices$.push(new $Invoice(po.number, invoiceseq^{++})$)
35:     $po.shipment.date = date$
36:     $po.shipment.courier = courier$
37:     InvoiceSent()
38: function delivery(unit $invoiceno$, unit $timestamp$)
39:            ▷ Courier delivers the goods and trigger the payment
40:     require(isValid(Invoices[invoiceno]))
41:     $po = PO[Invoices[invoiceno].orderno]$
42:     require($po.shipment.courier == msg.sender$)
43:     OrderDelivered()
```