

Using ABE to Secure Blockchain Transaction Data

Andrei Cristian

June 8, 2021

Table of Contents

- 1 Introduction
- 2 System and Models
- 3 PoP System Models
- 4 PoP Operation and Performance Analysis
- 5 Summary

Introduction

- Blockchain technology is increasingly being adopted as a trusted platform to support business functions including trusted and verifiable transactions, tracking, and validation.
- Most business use-cases require privacy and confidentiality for data and transactions \Rightarrow **Private blockchain solutions** \Rightarrow Unable to take full advantage of the capabilities, benefits and infrastructure of public blockchain systems.
- **Attribute-Based Encryption** security solution built on private-over-public (PoP) blockchain \Rightarrow Businesses will be able to **restrict access, maintain privacy, improve performance**, while still being able to benefit from the distributed trust of public blockchains.

Public and Private Blockchains I

- Similarities[1]:
 - ① both are decentralized peer-to-peer networks, where each participant maintains a replica of a shared append-only ledger of digitally signed transactions;
 - ② both maintain the replicas in sync through a protocol referred to as consensus;
 - ③ both provide certain guarantees on the immutability of the ledger, even when some participants are faulty or malicious.
- The main distinction between public and private blockchain is related to who is allowed to participate in the network.

Public and Private Blockchains II

- One of the drawbacks of the public blockchain is the substantial amount of computational power to maintain a distributed ledger at a large scale to achieve consensus, in which each node in a network must solve a complex, resource-intensive cryptographic problem - called Proof of Work (PoW)[2] to ensure all are in sync.
- Another disadvantage is the openness of public blockchain, which implies little to no privacy protection for transactions and only supports a weak notion of security.

Public and Private Blockchains III

- Many people believe private blockchains could provide solutions to many financial enterprise problems, that public blockchains do not, such as abiding by regulations such as Health Insurance Portability and Accountability Act (HIPPA), anti-money laundering (AML) and know-your-customer (KYC) laws, etc.
- Private blockchain is usually much faster, cheaper and respects the company's privacy.
- Private blockchains also provide more control power over the participants in the blockchain.

Public and Private Blockchains IV

- **Cross-chain** functionality aims to combine the best features of different blockchain systems[3], both private and public, for the purposes of exchanging value across disconnected ecosystems.
- None of existing solutions clearly addressed the problem of applying access control policies to enforce data privacy protection on transaction secrets.

Public and Private Blockchains V

- For example, when using smart contract solutions, e.g. Ethereum[4], 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[5] addresses this problem by relying on a TA approach to build permission groups for data access control. However, data access must be **predefined** which is not suitable for complex and dynamic businesses logic that require dynamic access control.

Public and Private Blockchains VI

- **RBAC** is **incompatbile** with the distributed nature of blockchain operations where transaction data are mobile and shared by multiple blockchain participants.

POP architecture

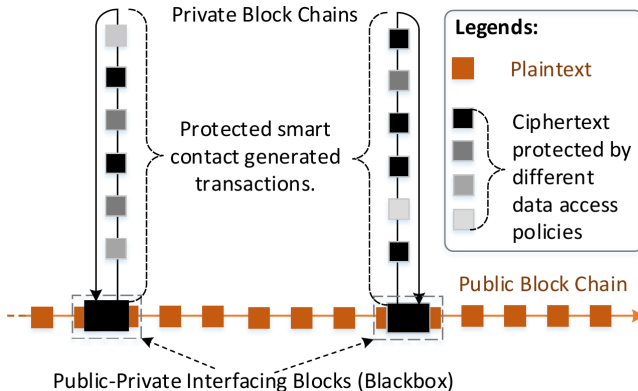


Figure: Illustration of PoP blockchain architecture.

ABE over PoP Blockchain I

- PoP architecture is presented in Figure 1.
- Applying ABE on an off-chain basis means it can inter-operate with the public blockchain without interference.
- Private blockchains transactions can be much **less computationally intensive** and provide **superior performance** 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.

ABE over PoP Blockchain II

- Each private blockchain can be viewed as a **protected state channel**.
- The integrity of a private blockchain can be validated 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 \Rightarrow distributed trust on the public chain is **not necessary** for the private blockchain.

ABE over PoP Blockchain III

- **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 blockchains 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.

The PoP Solution I

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

- It is a decentralized trust model for key management of ABE-based data access control. Using this approach, it can incorporate access control policies into ciphertext to protect content of smart contracts.
- It is a privacy-preserving messaging protocol to allow private blockchain participants to interact with the smart contract that can generate a private blockchain. This chapter illustrates how to use this protocol based on a supply-chain procurement application.

The PoP Solution II

- The solution provides two smart contracts: **PPP** (Public Parameters and Policies) to establish attribute based security trust model and **ppSCM** to provide secure data access control based on ABE scheme.
- A comprehensive security and performance analysis is presented 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.

System and Models

- To illustrate the presented solution, in Figure 2 it is used a supply chain example based on **Block-Chain Technology**(BCT), which involves multiple parties.
- 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.
- Because the information is registered on a distributed database, it makes it **tamper-resistant** and **fosters greater trust** in the trade network.

Supply chain example

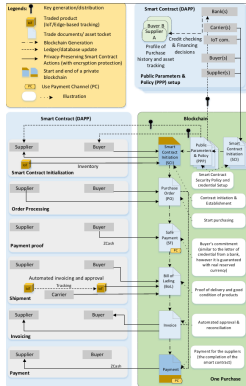


Figure: A supply chain scenario using IoT devices, blockchain, and data encryption protections.

Legends and Smart Contract interaction with Blockchain

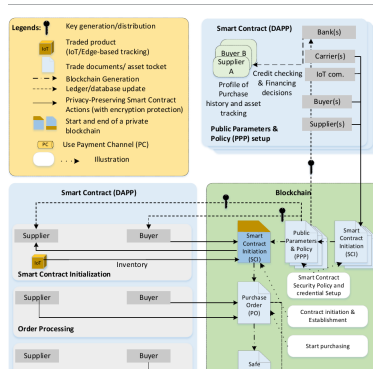


Figure: Zoom-in on Legends and interaction of the first smart contract with the blockchain

Smart Contract interaction with Blockchain

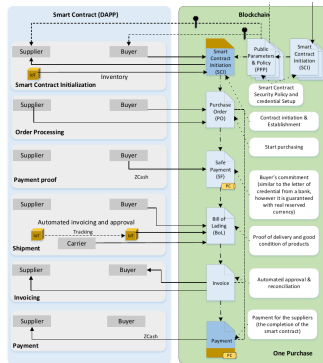


Figure: Zoom-in on the interaction of the second smart contract with the blockchain

BCT-supported purchase related transaction using DApp I

The left side of the figure present a BCT-supported purchase related transaction by using Ethereum's Decentralized App (DApp) solution 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.

BCT-supported purchase related transaction using DApp II

- 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.

BCT-supported purchase related transaction using DApp III

- 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**.
- Another potential application is provided by smart contracts and cryptographic multi-signatures and product content protection for all the various documentation and processing stages involved in a trade transaction.
- 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:**

BCT-supported purchase related transaction using DApp IV

- 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 [6], 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**.

BCT-supported purchase related transaction using DApp V

- 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**.

BCT-supported purchase related transaction using DApp VI

- Blockchain solutions could create **more efficient payment processes** between banks, eliminating the need for each institution to **maintain** and **reconcile their own ledger**.

Privacy problem I

- The described smart contract **does not provide privacy protection** for transaction contents processed by smart contracts.
- Two additional modules (incorporated into original supply-chain procedures):
 - ① **Smart contract initialization:**
 - sets up the initial smart contract credentials such as agreed data access control policies for each step of smart contract;
 - initiates the off-chain operation, in which we start a private blockchain at this point.
 - ② **Payment proof:**
 - the private chain can also incorporate public blockchain evidence into the private blockchain;

Privacy problem II

- the addition of the payment proof procedure is to utilize the payment channel 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.

Smart Contracts I

- In Bitcoin, the concept of “*scripting*” has already existed, which is actually **a weak version** of smart contract.
 - it lacks Turing-completeness, thus does not nearly support everything;
 - it is value-blinded;
 - it lacks state, UTXO can either be spent or not, there is no way to keep other states except for these two;
 - it is blockchain blinded.
- Ethereum smart contract is to build a decentralized application to create a blockchain with a build-in Turing complete programming language.

Smart Contracts II

- Smart contract means is defined to be a cryptographic “boxes” that contain value and only unlock it if certain conditions are met.
- A smart contract will also be stored in the blockchain and can be retrieved by its address and integrity can be guaranteed as well.
- With smart contract, one can express logics such as “only after April 17th, 2018, can the document be sent to A”.
- In the presented supply-chain example in Figure 2, two smart contracts are involved:

Smart Contracts III

- 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 PPP as a template**;

- 2 **private blockchain smart contract**: the smart contract on the left side of 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**.

ABE-Enabled ABAC

- ABE is a way to implement attribute-based access control, in which, data will be encrypted and a data owner could define an access policy describing what attributes the data users need to own in order to get access of the data.
- Presentation of an extended Lewko's scheme [7] 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**.

Comparing existing blockchain data privacy protection solutions I

The presented approach has the following important features compared to existing blockchain data privacy protection solutions:

- **Distributed and mobile:** 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.

Comparing existing blockchain data privacy protection solutions II

- **Federated:** 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.

Comparing existing blockchain data privacy protection solutions III

- **Provides interoperability feature:** 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 $\{A_1 \text{ AND } A_2\}$.

Security Policy I

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

$\mathcal{P}1$ = The **pricing** and **quantity** can be accessed by the **supplier** and the **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 using one or multiple data access policies.

Security Policy II

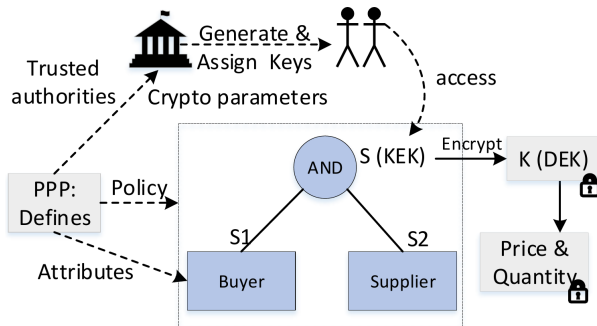


Figure: Attribute(or policy)-based access control setup.

Security Policy III

- 1 The policy $\mathcal{P}1$ is presented in Figure 5, which is called Policy Tree (PT).
- 2 A PT is constructed by attributes at the leaves and intermediate nodes are logical gates.
- 3 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.
- 4 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.

Scheme Construction I

- Lewko's scheme requires users to derive their private key from one trusted authority to allow them to use the same attribute and corresponding private key to decrypt a ciphertext.
- Extension of Lewko's solution by adding multi-authority key generation scheme - *federated authority setup* and *federated key generation*, for an attribute and corresponding private key is generated by multiple authorities.
- Only if **all** involved trusted authorities got compromised, e.g., using collusion attack, can they derive the private key for a user.

Scheme Construction II

- Considering the heavy computation overhead, part of the encryption and decryption computation is outsourced to the edge nodes. The following is the outsourced version of the scheme.
- **Global Parameters Setup**(λ) \rightarrow GP:
 - The Global Parameter (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 $N = p_1 p_2 p_3$.

Scheme Construction III

- $GP = \{N, g_1, H : \{0, 1\}^* \rightarrow G\}$, 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.

Scheme Construction IV

- 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 $MPK_j = \{e(g_1, g_2)^{\alpha_i}, g_1^{y_i}, \forall i\}$ 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.
 - To relax the requirement that states that the same authority must be used for the private keys to be derived, multiple authorities must be involved 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.

Scheme Construction V

- 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 .
- $AAS = \{AA_1, AA_2, \dots, AA_n\}$.
- 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:

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

Scheme Construction VI

- AA_i will calculate:

$$MPK_{i \rightarrow i+1} = (e(g_1, g_1)^{\sum_{j=2}^i \alpha_{j-1}})_i^\alpha, (g_1^{\sum_{j=2}^i y_{j-1}})_i^y.$$

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

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

$$\hat{SK} = \{\sum_{j=1}^n \alpha_j, \sum_{j=1}^n y_j\}.$$

- **Encrypt** $(M, (A, \rho), GP, \{MPK\}, \hat{MPK} \rightarrow CT$:
 - M is a message.

Scheme Construction VII

- A is an $n \times \ell$ access matrix.
- 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 denotes $A_x \cdot w$.
- 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 .

Scheme Construction VIII

- The ciphertext is as follows:

$$CT = \langle C_0, C_{1,x}, C_{2,x}, C_{3,x} \rangle, \quad \text{where}$$

$$C_0 = Me(g_1, g_1)^s, \quad C_{1,x} = e(g_1, g_1)^{\lambda_x} e(g_1, g_1)^{\alpha_{\rho(x)} r_x},$$

$$C_{2,x} = g_1^{r_x}, \quad C_{3,x} = g_1^{y_{\rho(x)} r_x} g_1^{\omega_x}, \forall x.$$

- 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.

Scheme Construction IX

- **KeyGen**($GID, i, \{MSK\}, GP$) $\rightarrow SK_{i,GID}$:
 - 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**($GID, i, \{MSK\}, \hat{MSK}, GP$) $\rightarrow SK_{i,GID}$:
 - The federated key generation algorithm is run when multiple attribute authorities need to generate the private key for an attribute shared among multiple users.

Scheme Construction X

- Assume that $AAS = \{AA_1, AA_2, \dots, AA_n\}$.
- AA_i will generate $g_1^{\alpha_i} H(GID)^{y_i}$.
- AA_{i-1} will send AA_i the private key component:

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

- 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:

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

Scheme Construction XI

- **Decrypt**($CT, \{SK_{i,GID}\}, GP) \rightarrow M$:
 - 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 $(1, 0, \dots, 0)$ is in the span of these rows, then the plaintext message M can be obtained in the following way:

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

Scheme Construction XII

- The decryptor chooses constants $c_x \in \mathbb{Z}_N$ so that $\sum_x c_x A_x = (1, 0, \dots, 0)$ 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.

Offloaded Encryption and Decryption I

- Considering participant running DApp on mobile devices, ABE based computation can be potentially intensive for mobiles.
- \Rightarrow An ABE offloading model that can significantly reduce the computation overhead on mobiles.
- Assumption that an edge cloud node can assist mobiles to perform offloading functions.
- **Encrypt**: similar to the encryption presented in *Scheme Construction*, but only C_0, C_1 are calculated by the data owner (mobile device), and C_2, C_3 are calculated by the edge node.

Offloaded Encryption and Decryption II

- **Decrypt:** similar to the decryption presented in *Scheme Construction*, but the private key holder only needs to calculate $e(SK_{\rho(x), GID}, C_{2,x})$. The edge node will calculate $C_{1,x} \cdot e(H(GID), C_{3,x})$.

Security Model

- PoP assumes that blockchain participants are curious, selfish, 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.

Access System Construction I

- **Definition 1**(Contract)

- A smart contract C is defined by a procedure $C = \{T\}$.
- The procedure 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 $\{a\} \in A = \{read, write, change\}$.
- Each action a is restricted by a privilege $\alpha(a, T) : \mathcal{P}(a, T) \mapsto S_{a,T}$.
- The privilege $\alpha(a, T)$ are defined by capabilities such as $\{can, cannot, restricted\ by/to\}$.

Access System Construction II

• Definition 3(Subject)

- A subject $s \in S$ or G (here S is a subset of subjects and G is the overall group and includes all the subjects) is an entity involved in smart contract that can perform actions.
- Each subject has a data access privilege described by a policy $\mathcal{P}(s) : \cup_{\forall(a, T) \rightarrow s} \mathcal{P}(a, T)$.
- $\cup_{\forall(a, T) \rightarrow s}$ denotes the union for all action and transaction pair (a, T) it involves the subject s .
- The collection of policies for a smart contract involving the subject s is denoted by $\mathcal{P}(s, a) = \cup_{\forall(a, T) \rightarrow s} \mathcal{P}(s, a)$.
- $\mathcal{P}(s, a)$ 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.

Access System Construction III

- **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 $\mathcal{P}(o) = \bigcup_{\forall(a, T) \mapsto o} \mathcal{P}(o, a)$.
- $\mathcal{P}(o)$ 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:

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

Access System Construction IV

- $T : S \rightarrow o|_{\{a\}}^*$: in transaction T , subset of subjects S can operate action(s) $\{a\}$ limited by condition $*$ on object o under access policies $\mathcal{P}(o, a) \subset \mathcal{P}(S, a)$;
- $*$ is a condition to confine action(s) such as in a transaction T or in a contract C .

Privacy-Preserving Messaging Protocol (PPMP) I

Goal: For a given transaction T , one or a set of subject(s) $S \subset G$ can perform an action a , where

- $\mathcal{P}(S, a) \neq \mathcal{P}(\bar{S}, a)$.
- The collective privilege (e.g. by colluding) of set \bar{S} cannot satisfy the privilege given by subset S .
- $\Rightarrow \mathcal{P}(S, a)$ can only be satisfied by the subgroup of participants in S .
- 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 .

Privacy-Preserving Messaging Protocol (PPMP) II

Messages: In order to implement the access control privilege associated to an action in a smart contract, we use cryptographic approaches. The following three cryptographic enforcement operations are defined:

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

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).

Privacy-Preserving Messaging Protocol (PPMP) III

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

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

or

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

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

Smart Contract Protocols

Presentation of the two smart contracts that are present in the supply-chain example of Figure 2:

PPP Contract I

- *PPP*(Public Parameters and Policies) is a smart contract created by an initiator on the public blockchain.
- The initiator could be a TA 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.

PPP Contract II

- 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.

PPP Contract III

```

1: struct AA = { ... }
2: AAS = {AA1, ..., AAn}
3: struct P = { ..., count }
4: PS = {P1, ..., Pn}
5: function PPP(P p)
6:     PS.push(new P(p))
7: function JOIN(Attribute attr)
8:
9:     require(isAllow(msg.sender)
10:    require(msg.sender ∉ AAS)
11:    AAS.push(new AA(msg.sender, attr))
12:    if (AA.attributes ∉ PS) then
13:        add AA.attributes to PS
14: function VOTE(P p)
15:     require((msg.sender ∈ AAS)
16:     require(not isVoted(addr AA))
17:     PS.find(p).count++
18: function GETPOLICIES( )
19:     require(msg.sender ∈ AAS)
20:     Policies = ∅
21:     for each p ∈ PS do
22:         Policies.push(p)
23:     return Policies
24: function GETPPP( )
25:     require(msg.sender ∈ AAS)
26:     return PS.findMaxVote()
```

- ▷ Define Attribute Authority
- ▷ Address of each joined AA
- ▷ Define Policy and their count
- ▷ A set of policy to be negotiated
- ▷ Initiator create a new policy

- ▷ Initiator grants a new AA to join the negotiation

PPP Contract IV

Figure: Pseudo-code of PPP Contract.

ppSCM Contract I

- *ppSCM* contract is a smart contract created by a supplier on a dedicated private blockchain.
- It uses the negotiated policy from *PPP* on a 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.

ppSCM Contract II

- 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 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 *OrderSent*, the supplier could receive the order data and process it.

ppSCM Contract III

- After received the purchase order, the supplier looks for the best shipping price on the carrier's smart contract.
- The supplier 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 goes to the smart contract account and waits there until the delivery.

ppSCM Contract IV

- After safe payment, the supplier sends the invoice with delivery date and some other data to the buyer by calling *sentInvoice*. The buyer receives the invoice data through the event called *InvoiceSent*.
- The carrier, after delivery, marks the order as delivered on the *ppSCM* smart contract by calling *delivery* function.
- The event *OrderDelivery* then calls a smart contract in the public blockchain to payout the supplier for the order, and payout the carrier for the shipment.

Pseudo-code of ppSCM Contract pt. 1

```

1: address addr' Seller, addr' Buyer
2: struct PO = { goods, quantity, number, price, safe pay, shipment } ▷ Purchase Order Data
3: struct Shipment = { courier, price, safe pay, payer, date }
4: struct Invoice = { orderno, invoiceno }
5: AccessPolicy =  $\emptyset$ 
6: Orders =  $\emptyset$ , Invoices =  $\emptyset$ 
7: orderseq = 0, invoicseq = 0
8: function PPSCM(address buyerAddr, Policy AccP)
9:                                     ▷ Initialize contract by a seller
10:   addr' Seller = sender
11:   addr' Buyer = 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()

```

Figure: Pseudo-code of first part of ppSCM Contract.

Pseudo-code of ppSCM Contract pt. 2

```
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()
```

Figure: Pseudo-code of second part of ppSCM Contract.

Policy-Based Data Privacy Protection I

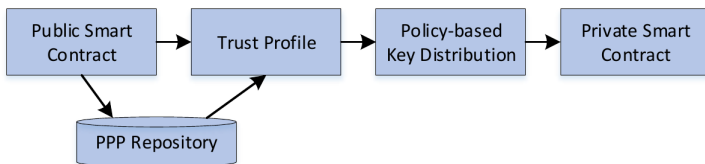


Figure: Trust and policy management procedure

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

Policy-Based Data Privacy Protection II

PPP is built based on the following data structure:

- D1: A set of Global Parameters (GP) 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 $\{MPK\}, \hat{MPK} \rightarrow \{A\}$. The mapping between public parameters and attributes allow each private blockchain participant to select which public parameters to use in the ABE **Encrypt** procedure.

Policy-Based Data Privacy Protection III

- 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 contracts over public blockchain (Figure 6), and thus they are searchable. A public directory service can be used to store established PPP. The PPP can be **resued** as a template for a new private blockchain.

In addition, when creating a new private blockchain, the participant can initiate an **update smart contract** to update the authority list and associated attributes.

Private Key Distribution I

- 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.

Private Key Distribution II

- 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 well-known 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 be used.

Data Object Encryption and Decryption I

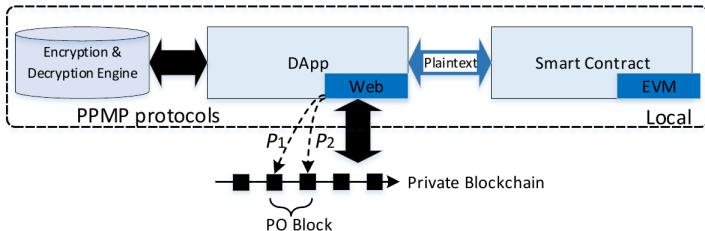


Figure: Data object encryption and decryption.

- The data access protocol is specified in PPMP protocol.
- The data object operation diagram is presented in Figure 10.

Data Object Encryption and Decryption II

- Both DApp (a web-based app) and smart contract (running within an Ethereum Virtual Machine (EVM)) run locally on a user's site.
- 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 the encryption/decryption engine.
- The data object granularity is determined by access control policies.
- Using the policy provided in *Security Policy 1*: $\mathcal{P}1 = \text{The pricing and quantity can be accessed by the supplier and the buyer.}$

Data Object Encryption and Decryption III

- $\mathcal{P}1$ is an example to specify the data protection when creating the PO.
- The PPMP message that will be used on DApp, which runs on ABE enc/dec engine to create the PO blockchain block and can be written as:
$$PPMP(T, \{a\}, \mathcal{P}, c, s, o) =$$
$$= T_{PO} : \{buyer, supplier\} \rightarrow price\&quantity|_{create:\mathcal{P}1,E}.$$
- PO should also contain an address information for shipment.

Data Object Encryption and Decryption IV

- If the data access policy is $\mathcal{P}2 = \textit{The shipment address information can be accessed by the buyer and the carrier.}$, then the PPMP message can be:
$$PPMP(T, \{a\}, \mathcal{P}, c, s, o) =$$
$$= T_{PO} : \{buyer, carrier\} \rightarrow shippingaddress|_{create:\mathcal{P}2,E}.$$
- 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.

Data Object Encryption and Decryption V

- Technically, we can combine $\mathcal{P}1$ and $\mathcal{P}2$ as one policy, 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.
- The crypto currency involved functions can be achieved by using public blockchain's payment channel approach .

Complexity Analysis I

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

Table: Computation complexity comparison in terms of the number of pairing operations.

- **E** and **P** represents exponentiation and pairing respectively.
- U_i indicates the set of attributes managed by a certain attribute authority.

Complexity Analysis II

- 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.

| Schemes | DABE | device | edge node |
|---------|----------------------------|---------------------------|----------------|
| Setup | $2 U_i $ | $2 U_i $ | 0 |
| KeyGen | $2 S $ | $2 S $ | 0 |
| Encrypt | $5n\mathbf{E}+1$ | $2n\mathbf{E}+1$ | $3n\mathbf{E}$ |
| Decrypt | $2n\mathbf{P}+n\mathbf{E}$ | $n\mathbf{P}+n\mathbf{E}$ | $n\mathbf{P}$ |

Table: Complexity of the algorithms of the presented scheme.

Complexity Analysis III

- Table 2 compares the complexity of the original algorithms and the scheme with Offloaded encryption and decryption.
- It is seen that for the encryption the edge node is able to do more than half of the computational work.
- As for the decryption, the edge node will share half number of the pairing operations.
- Since the main complexity is caused by exponentiation and pairing, we only count the number of these two operations

Computation Evaluation

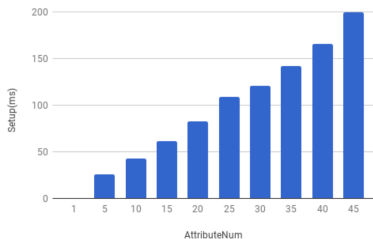
The evaluation is based on Charm, i.e., a python library for pairing-based cryptography. The evaluation environment is a VM on top of Intel Xeon CPU E5-2650 v2 @ 2.60hZ. Two virtual CPU, 4GB RAM and 80GB hard disk are assigned for this VM.

The performance evaluation of setup, keygen, encrypt and decrypt is presented from Figure **11a** to Figure **12b**.

Figure **13a** and Figure **13b** compare the computation overhead between the edge node and the user's end.

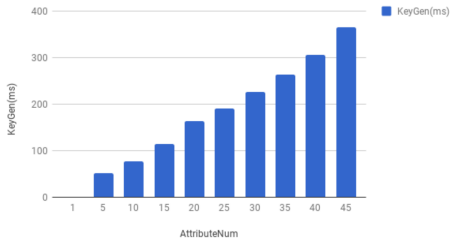
Setup and KeyGen times

Setup(ms) vs. AttributeNum



(a) Setup time.

KeyGen(ms) vs. AttributeNum

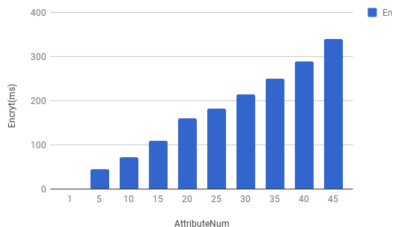


(b) KeyGen time.

Figure: Setup and KeyGen times.

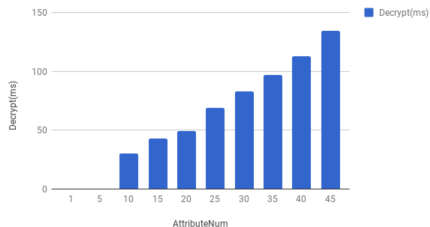
Encryption and Decryption times

Encrypt(ms) vs. AttributeNum



(a) Encryption time.

Decrypt(ms) vs. AttributeNum

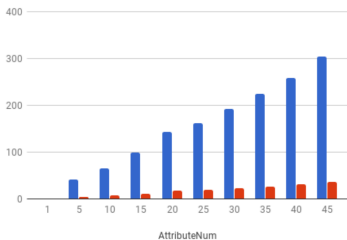


(b) Decryption time.

Figure: Encryption and Decryption times.

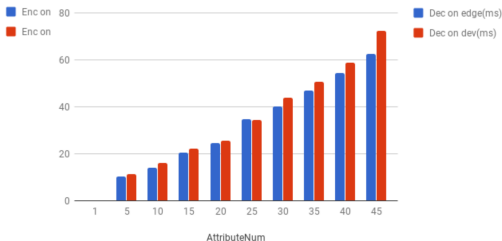
Computation Overhead with Offloading

Enc on edge(ms) and Enc on dev(ms)



(a) Encryption time.

Dec on edge(ms) and Dec on dev(ms)



(b) Decryption time.

Figure: Computation overhead between the edge node and the user's end.

Security Analysis I

- The presented ABE scheme is based on Lewko's scheme from [7].
- Lewko's scheme is proved to be secure against
 - multiple $(n - 1)$ trusted authority collusion attack;
 - collusion among users.
- The solution extends Lewko's scheme to **Federated Authority Setup** and **Federated KeyGen**.
- The federated setup and key generation algorithm does not cause security issue and break the collusion problem provided by Lewko's scheme.
- We have the following theorem:

Security Analysis II

Theorem

The presented scheme is resistant against attacks from colluded attribute authorities.

Security Analysis III

Proof.

Assume that $n - 1$ attribute authorities in AAS except for one attribute authority AA' colluded, each of which provides the value α_j and y_j ($AA_j \in AAS$ and $AA_j \neq AA'$). We denote the set of colluded attribute authorities as CAA . To compromise the presented scheme, the colluded authorities need to obtain the value of $\alpha = \sum_{j \in AAS} \alpha_j$ and $y = \sum_{j \in AAS} y_j$. However, they can only get the value of $\alpha' = \sum_{j \in CAA} \alpha_j$ and $y' = \sum_{j \in CAA} y_j$. Assume that they successfully calculate the value α and y with access to $\alpha', y', e(g_1, g_1)^{\alpha_j}$ and $g_1^{y_j}$ ($AA_j = AA'$), \Rightarrow the colluded attribute authorities solved well known computationally difficult problem - discrete logarithm problem \Rightarrow contradiction \Rightarrow the presented scheme is resistant against the colluded attribute authorities. □

Collusion Resistance Summary

- For an adversary to compromise the system during federated setup and private key generation, the values $\sum_{j=1}^n \alpha_j$ and $\sum_{j=1}^n y_j$ must be obtained.
- 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 colludes together can the private secret get leaked.
- If the number of colluding attribute authority is $n - 1$ or fewer than that, the presented federated algorithms are resistant against collusion attacks.

Security Analysis of the Scheme with Offloaded Encryption and Decryption I

- In the Offloaded encryption algorithm, both the message M and related information of the shared secret s is leaked to the edge node.
- The edge node is capable to help offload the computation overhead and also knows nothing about the encrypted message.
- In the decryption algorithm, the edge node is only responsible for the private key unrelated computation, i.e.,
 $C_{1,x} \cdot e(H(GID), C_{3,x})$.
- Therefore, the private key is not leaked to the edge node, while the edge node can still help do the decryption.

Security Analysis of the Scheme with Offloaded Encryption and Decryption II

- Easily proved that offloaded scheme is secure if the original scheme is secure.

Main feature comparison with existing major privacy preserving solutions




Blockchain feature comparison

| Solution | Protect | Method | Compu. | Type | Access Policy |
|--------------------------------|----------------|--|--------|----------|---------------|
| PoP | smart contract | off-chain execution & ABE | Medium | no limit | Yes |
| Hawk [132] | smart contract | off-chain execution & on-chain zkSNARK | heavy | no limit | No |
| Ekiden [54] | smart contract | off-chain Hardware Tee | low | no limit | No |
| Maxwell [156] | amount | on-chain | n/a | Bitcoin | No |
| ZeroCash [190] | identity | on-chain | n/a | Bitcoin | No |

Summary

- Presented a blockchain solution on how to build private blockchains over public blockchains - PoPs.
- Presented a set of messaging and smart contract protocols - to illustrate privacy-preserving functions of PoP.
- Use of a supply-chain procurement procedure example to illustrate how PoP works.
- A starting point for research and development for the next step:
 - More functional-rich policy-based access control solutions should be considered: ex. policy/attributes expiration.
 - The presented smart contract only focuses on PPP establishment and procurement. Other smart contracts such as cancellation/revocation of a contract should be investigated.

References I

-  P. Jayachandran, “The difference between public and private blockchain,” , 2017. [Online]. Available: <https://www.ibm.com/blogs/blockchain/2017/05/the-difference-between-public-and-private-blockchain/>.
-  S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” , 2008. [Online]. Available: <https://bitcoin.org/en/bitcoin-paper>.
-  J. Liebkind, “Public vs private blockchains: Challenges and gaps,” , 2018. [Online]. Available: <https://www.investopedia.com/news/public-vs-private-blockchains-challenges-and-gaps/>.

References II



Ethereum project, [Online]. Available:
<https://www.ethereum.org>.



C. Cachin, “Architecture of the hyperledger blockchain fabric,” , vol. 310, 2016. [Online]. Available: <https://www.zurich.ibm.com/dccl/papers/cachindccl.pdf>.



H. Erik, S. Urs Magnus, and N. Bosia, “Supply chain finance and blockchain technology: The case of reverse securitisation,” , Springer, 2017.

References III



L. Allison and W. Brent, “Decentralizing attribute-based encryption,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2011, pp. 568,588.