Journal of Network and Computer Applications 116 (2018) 42–52

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

BSeIn: A blockchain-based secure mutual authentication with fine-grained access control system for industry 4.0☆

Chao Lin[a], Debiao He[a,*], Xinyi Huang[b], Kim-Kwang Raymond Choo[c], Athanasios V. Vasilakos[d]

[a] Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China
[b] School of Mathematics and Computer Science, Fujian Normal University, Fuzhou 350007, China
[c] The University of Texas at San Antonio, San Antonio, TX 78249, USA
[d] Department of Computer Science, Electrical and Space Engineering, Lulea University of Technology, Skellefte SE-931 87, Sweden

ARTICLE INFO

Keywords:
Industry 4.0
Smart factory
Blockchain
Smart contract
Authentication
Access control
Blockchain-based secure mutual authentication

ABSTRACT

To be prepared for the 'Industry 4.0'-era, we propose a hierarchical framework comprising four tangible layers, which is designed to vertically integrate inter-organizational value networks, engineering value chain, manufacturing factories, etc. The conceptual framework allows us to efficiently implement a flexible and reconfigurable smart factory. However, we need to consider security inherent in existing (stand-alone) devices and networks as well as those that may arise in such integrations. Especially the existing solutions are insufficient to address these fundamental security concerns. Thus, we present a blockchain-based system for secure mutual authentication, BSeIn, to enforce fine-grained access control polices. The proposed system (with integrated attribute signature, multi-receivers encryption and message authentication code) is designed to provide privacy and security guarantees such as anonymous authentication, auditability, and confidentiality. BSeIn also scales well due to the utilization of smart contract. We then evaluate the security and performance of BSeIn. For example, findings from the performance evaluation demonstrate that Initialization/Request Issuance/Chain Transaction/State Delivery/Permission Update phase only cost 12.123/4.810/6.978/0.013/2.559s, respectively.

1. Introduction

While some regard 'Industry 4.0' as one of several recent flashy catchphrases, in our digitalized society, automation and data exchange in the manufacturing sector (as well as many other industry sectors) are increasingly interconnected and integrated. This is partly facilitated by the extension of Internet of Things (IoT) and services to the industry (Wang et al., 2016). A hierarchical layout, such as Fig. 1, can be used to facilitate high flexibility and reconfigurability in an Industry 4.0 smart factory, say by enhancing data/information flow better via the interconnected cyber-physical system.

In such a layout, there are at least four layers, namely: some *Physical Resources*, one or more *Industrial Networks*, the *Cloud*, and some *Terminals*. *Physical Resources* can connect with each other via an Industrial Network (which can be wired and/or wireless). *Cloud*-based information systems such as enterprise resource planning (ERP) can then be deployed to pull or push data from *Physical Resources* and interact with other devices or users via the *Terminals*. For example, services or requests (e.g. physical resources' status info) can be submitted and processed remotely and online.

Security is one of the most important concerns in an Industry 4.0 implementation, given that vulnerabilities introduced during manufacturing (e.g. hardware Trojans and backdoors) can be hard to detect. Also a smart factory may be more exposed than conventional Internet and electronic commerce applications, due to broader and more diverse attack vectors (Qiu et al., 2011, 2012). For example, users with different motivations, roles and agenda (e.g. customers, suppliers and manufacturers) can utilize the terminals to interact with the cloud-based systems. They may query data or attempt to take control of some physical resources (i.e. terminals and the communication channel are the target). Or an attacker can seek to collect available data through harvesting and exfiltration of sensitive information from these terminals.

Thus, it is not possible to design a solution that can comprehensively address and mitigate all security threats. In this paper, we focus only on the access control mechanism, as this is what many would regard the

☆ Fully documented templates are available in the elsarticle package on CTAN.

* Corresponding author.
E-mail addresses: linchao91@qq.com (C. Lin), hedebiao@whu.edu.cn (D. He), xyhuang81@gmail.com (X. Huang), raymond.choo@fulbrightmail.org (K.-K.R. Choo), th.vasilakos@gmail.com (A.V. Vasilakos).

https://doi.org/10.1016/j.jnca.2018.05.005
Received 28 February 2018; Received in revised form 28 March 2018; Accepted 7 May 2018

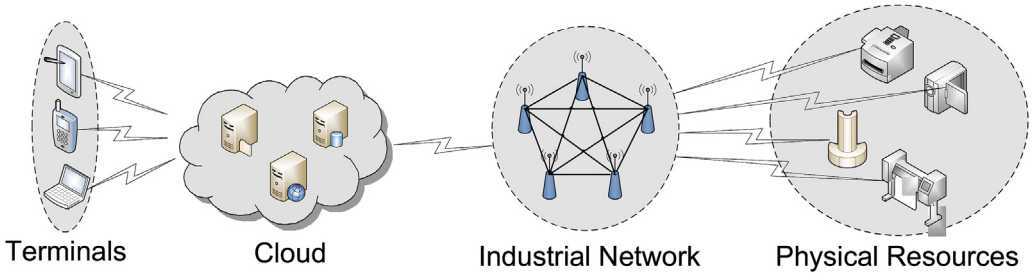

Fig. 1. An example Industry 4.0 smart factory layout.

'gatekeeper' to information systems and our data. With a secure and efficient authentication mechanism with access control, we can significantly reduce potential cyberthreats.

There exist many solutions to data security from multiple perspectives, such as (Li et al., 1109; Li et al., 2015; Chen et al., 2016; Huang et al., 2017). And a variety of authentication mechanisms such as those based on password-based authentication (Tsai et al., 2006) (simplest and most widely used in electronic commerce systems, but relatively weak), One Time Password authentication (Haller, 1995; M'Raïhi et al., 2005; Vaidya et al., 2011) (OTP, a widely used two-factor authentication scheme, particularly in the banking and financial sector), smart card based authentication (Liu et al., 2008) (relatively efficient in commercial applications), have been proposed in the literature. However, these mechanisms, as well as several others, are not able to ensure the following fundamental security properties (Li and Hwang, 2010; Li et al., 2013):

1. **Privacy Protection and Anonymity**: A simple password authentication approach, such as those discussed above, is not designed to preserve the privacy of the terminals' real identities, since a malicious or compromised cloud can trivially identify a terminal based on daily activities via the interactions.
2. **Fine-grain Access Control**: A fine-grain access control is commonly achieved by integrating some strategies, such as constructing user-role mapping table and role authorization table (Sandhu et al., 1996). However, approaches based on the use of these tables are vulnerable to attacks at the cloud end, since these tables are generally maintained by the semi-trusted cloud.
3. **Integrity and Confidentiality**: The terminals' access history, such as those discussed earlier, are recorded/stored by the cloud in a local database. Hence, there is a risk that these databases can be subject to unauthorized access and modification. Consequently, traceability and auditability of access records will be challenging.

1.1. Motivations

Seeking to contribute to the knowledge gap, we design a blockchain based solution for Industry 4.0 applications. In this paper, however, we use smart factory as a use case. We also remark that the potential of blockchain has been explored in a number of contexts, ranging from data storage (Kopp et al., 2017) to data provenance (Liang et al., 2017; Zyskind et al., 2015) to identity management (Azouvi et al., 2017) to crowdsouring (Li et al., 2017), and so forth. This is not surprising as blockchain's properties on decentralization, verifiability and immutability can be leveraged to enhance security (in conjunction with other cryptographic tools), service availability (avoiding the single point of failure), system scalability (e.g. due to programmable smart contracts).

1.2. Main contributions

The contribution of this research is the conceptual blockchain-based system (BSeIn) for remote mutual authentication with fine-grained access control, suited for Industry 4.0 deployments. In BSeIn, both blockchain and attribute signatures (Maji et al., 2010) are integrated to anonymously authenticate terminals, and message authentication code (Bellare et al., 2000) is utilized to efficiently authenticate gateways. In addition, the multi-receivers encryption (Islam et al., 2015) is utilized to provide confidentiality, that is, only authorized participates (e.g. permission nodes, cloud gateway, Industrial network gateway) can obtain access to the 'raw' context of request messages. To ensure scalability, fundamental in Industry 4.0 applications, we utilize smart contracts in BSeIn, where the entire request process is performed via interacting with smart contracts (i.e. Smart Contract on PDHT or Smart Contract on TX described in Section 3.6). We also present the security requirements of such a blockchain-based mutual authentication for Industry 4.0 deployments, prior to discussing how BSeIn can provide such security requirements. We then implement our proposal to evaluate the feasibility using a software prototype based on JUICE.[1]

1.3. Organization

The rest of this paper is organized as follows. Section 2 describes the blockchain-based system model and the security requirements. Concrete system building blocks are described in Section 3. In Sections 4 and Section 5, we present BSeIn its security analysis, respectively. Section 6 describes the implementation and presents the performance evaluation findings. The final part is the conclusion presented in Section 7.

2. Problem definition

In this section, we introduce the system model and the relevant security requirements.

2.1. System model

The participants involved in BSeIn include **Terminals**, **Blockchain Network**, **Cloud**, **Industrial Network** and **Physical Resources**, as shown in Fig. 2.

- **Terminals**: Terminals are often some remote telecommunication devices (e.g. mobile devices) that can remotely request for access or process the commands. When these participants wish to ask for a request, they need to publish a corresponding transaction (see Section 3.3) to the blockchain network.
- **Blockchain Network**: In our system, we adopt a permissioned fabric that comprises some processing and consensus nodes (i.e. permissioned nodes) responsible for maintaining the blockchain according to a consensus mechanism. In our system, we adopt a practical consensus mechanism (PBFT) unlike the Bitcoin blockchain (uses proof of work, PoW). Permissioned nodes are classified into "validation node" (responsible for verifying transactions, abbreviated as "vdn") and "bookkeeping node" (takes charge of

[1] https://www.juzhen.io/.

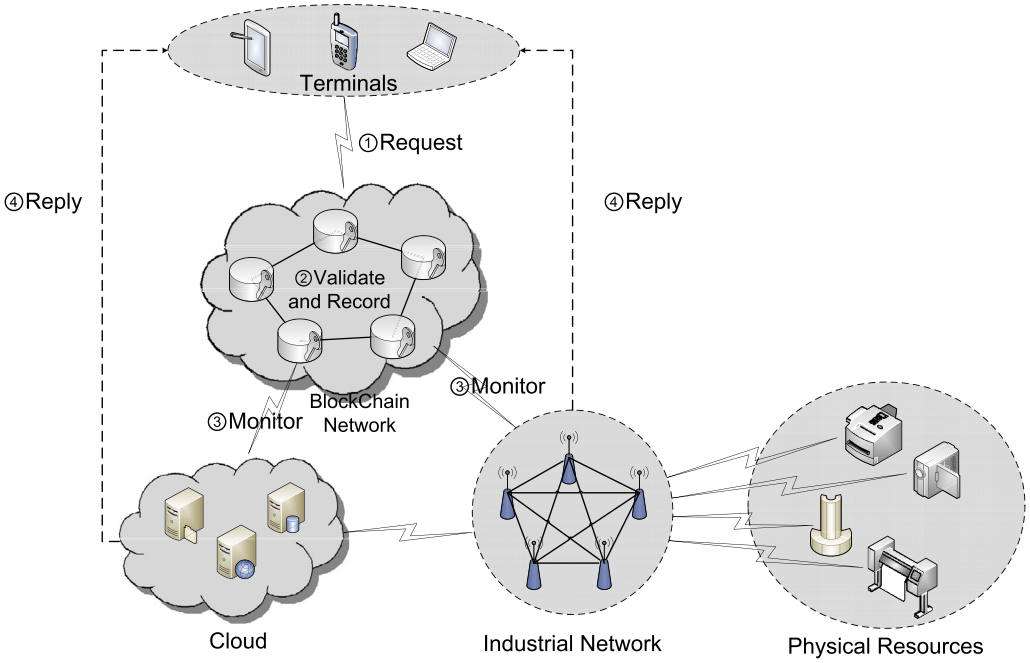

Fig. 2. Proposed BSeIn system.

chaining validated transactions into the blockchain, abbreviated as "bkn").

After permissioned nodes reach a consensus on the received requests from terminals, the requests will be recorded into the blockchain. The details of the blockchain network will be introduced in Section 3.4.

- **Cloud**: The cloud hosts a number of information systems (e.g. ERP) which collect and process massive data from various physical resources, and responds to the access data requests from terminals. That is, the cloud monitors the blockchain network and responds to the requested data or terminals.
- **Industrial Network**: In contrast to the cloud, the industrial network is mainly tasked with dealing with requests from terminals, and such requests are mainly control commands. Once the industrial network monitors the blockchain network and receives the control requests, it will communicate with the relevant physical resource to execute the control commands and sent the results to the relevant terminals.
- **Physical Resources**: The function of this participant in BSeIn is the same as that in the conventional system, in the sense that they need to transfer the (massive) data to the cloud via the industrial network and execute the control orders from the industrial network.

2.2. Security requirements

Based on existing literature, such as those of (Aitzhan and Svetinovic, 2016; He et al., 2016), a blockchain based mutual authentication and fine-grained access system need to satisfy the following fundamental security requirements.

1. **Single registration**: To be practical, the system should only require single registration that one needs to register only once to publish transactions.
2. **Mutual authentication**: The system should provide mutual authentication (i.e between terminals and gateways) to resist common attacks such as impersonation attacks and man-in-the-middle attacks.
3. **User anonymity**: To preserve privacy, the system should guarantee the anonymity of terminals. From the view of the blockchain based system, any attacker cannot determine the real identity of terminals via a simple analysis of transactions.
4. **Fine-grained access control**: The authority or manager can adaptively grant or revoke terminals' permission for the policy at a fine-grained level, and in real-time.
5. **Session key agreement**: To securely communicate with participants, the system should support session key agreement or establishment. Namely, one can build a secure connection with others to negotiate a session key for subsequent communications.
6. **Perfect forward secrecy**: To ensure the security of previously transmitted messages, perfect forward secrecy should be provided where any attacker who obtains both communicating parties' private/public keys will not be able to recover prior session keys.
7. **No verifier table**: To minimize communication overhead and resist stolen verifier and denial of service attacks, the system should avoid having any verifier table. In other words, mutual authentication should be realized in the system without relying on a verifier table.
8. **No online registration center**: To minimize communication costs, the system should avoid having an online registration center. That is, both communication entities can authenticate each other directly without relying on an online registration center.
9. **Relay current timestamp**: To maintain a trust blockchain with chronological order, the system should utilize trust nodes to relay current timestamp in the pending block. That is, the potential of modifying the timestamp without detection is minimal.
10. **Birthday collision resilience**: The potential of two same blocks being generated at the same time is minimal. That is, the system should be able to resist birthday collision and avoid disputes between sub-blockchains.
11. **Interception and modification resilience**: The integrity of transmitted message should be protected for interception and modification, without detection.
12. **Hijacking resilience**: To ensure a smooth transaction, the system should reduce the potential for an attacker to hijack transactions, without detection.
13. **Resilience to other attacks**: The system should also be designed to ensure the resilience to common attacks, such as impersonation

attacks, distributed denial of service attacks, modification attacks, replay attacks and man-in-the-middle attacks.

3. System building blocks

In this section, we review the cryptography materials used in BSeIn and introduce a permission data hash table to achieve fine-grained access control. We also present the design of core blockchain components (i.e., transaction, blockchain, consensus mechanism and smart contract).

3.1. Cryptography materials

3.1.1. Attribute-based signatures (ABS)

In an attribute-based signature (ABS) scheme, a set of attributes is utilized to define a signer, and the corresponding certificate of those attributes is generated by an attribute authority (Li et al., 2010). Besides the basic properties as a digital signature (i.e. authenticity, integrity and non-repudiation), ABS provides properties such as anonymous authentication and fine-grained access control. Therefore, we replace the ECDSA signature (used in Bitcoin system) with ABS in our BSeIn for privacy-preserving and security.

In the BSeIn system, we adopt the most practical instantiation in (Maji et al., 2010), whose signatures comprise exactly $s + 2$ group elements (where s is the size of the claim-predicate's monotone span program (MSP) (Chen and Tang, 2013)), and does not rely on the Groth-Sahai proof system (Groth et al., 2008). In order to describe this construction, we will review the relevant notations in the scheme. That is, $c_{\max}$ denotes the maximum width of monotone span programs in the supported claim-predicates and $\mathbb{A} = \mathbb{Z}_p^*$ the set of attributes, where p is the prime order of chosen cyclic groups. Then, we specify its components as follows.

- ABS.TSetup: This algorithm initializes two cyclic groups G_1 and G_2 of prime order p, the corresponding bilinear pairing: e: $G_1 \times G_2 \to G_T$ and a collision-resistant hash function $\mathscr{H}$: $\{0, 1\}^* \to \mathbb{Z}_p^*$. Then, it randomly chooses $g \leftarrow G_1$ and $h_i \leftarrow G_2$ where $i = 1,\ldots,c_{\max}$. Finally, it returns the public parameters $PP = (G_1, G_2, \mathscr{H}, g, (h_0,\ldots,h_{c_{\max}}))$.
- ABS.ASetup: This algorithm randomly chooses $x_0, x, y, z \leftarrow \mathbb{Z}_p^*$ and computes: $Z = g^z$, $X_0 = h_0^{x_0}$, and $X_j = h_j^x$, $Y_j = h_j^y$, where $j = 1, \ldots, c_{\max}$. It returns (ASK, APK) as the master/public key pair, where $ASK = (x_0, x, y)$ and $APK = (X_0,\ldots,X_{c_{\max}}, Y_1,\ldots,Y_{c_{\max}}, Z)$.
- ABS.AttrGen: This signing key generation algorithm takes as input the master key ASK and attribute set $\mathscr{A} \subseteq \mathbb{A}$ and randomly chooses $K_{base} \leftarrow G_1$, then computes $K_0 \leftarrow K_{base}^{x_0^{-1}}$ and $K_u = K_{base}^{(x+yu)^{-1}}$, for all $u \in \mathscr{A}$. Finally, it returns the signing key $SK_{\mathscr{A}} = (K_{base}, K_0, K_u | u \in \mathscr{A})$.
- ABS.Sign: This signing algorithm takes as input $(APK, SK_{\mathscr{A}}, m, \gamma)$ (γ is a monotone boolean function), where $\gamma(\mathscr{A}) = 1$ (meaning that an input is set to be true if its corresponding attribute is presented in $\mathscr{A}$). Then, it converts γ to the MSP $M \in (\mathbb{Z}_p)^{r\times c}$ with the row labels u: $[r] \to \mathbb{A}$, and solves the vector $\vec{v}$ that matches the assignment $\mathscr{A}$. Next, it computes $\mu = \mathscr{H}(m\|\gamma)$ and randomly selects $\alpha_0 \leftarrow \mathbb{Z}_p^*, \alpha_1, \ldots,\alpha_r \leftarrow \mathbb{Z}_p$ to compute $L = K_{base}^{\alpha_0}$, $S_i = (K_{u(i)}^{v_i})^{\alpha_0}\cdot(Zg^u)^{\alpha_i} (\forall i \in [r])$, $R = K_0^{r_0}$, and $P_j = \Pi_{i=1}^{l}(X_j B_j^{u(i)})^{M_{ij}\cdot\alpha_i} (\forall j \in [c])$. Finally, it returns the signature $\sigma = (L, R, S_1,\ldots,S_r, P_1,\ldots,P_c)$.
- ABS.Ver: This verification algorithm takes as input (APK, σ, m, γ), parses $\sigma = (L, R, S_1, \ldots, S_r, P_1, \ldots, P_c)$, and converts γ to the MSP $M \in (\mathbb{Z}_p)^{r\times c}$ with the row labels u: $[r] \to \mathbb{A}$, and computes $\mu = \mathscr{H}(m\|\gamma)$. If $L \neq 1$, $e(R, A_0) = e(L, h_0)$ and

$$\Pi_{i=1}^{r} e\left(S_i, (A_j B_j^{u(i)})^{M_{ij}}\right) = \begin{cases} e(L, h_1)e(Cg^{\mu}, P_1), & j = 1 \\ e(Cg^{\mu}, P_j), & j > 1, \end{cases} \quad (1)$$

where $j \in [c]$, then it returns 1 to accept σ.

The security of this scheme is proven in the generic group model, which is the cornerstone for practical applications. Interested readers are referred to (Maji et al., 2010).

3.1.2. Multi-receivers encryption (MRE)

Broadcast Encryption (BE) and Multi-Receivers Encryption (MRE) are used to build a secure communication between one entity and a group of pre-selected receivers (Islam et al., 2015). That is, one can securely broadcast a common message to several receivers over an open network using BE/MRE. Due to the special properties of BE/MRE (i.e. confidentiality of message and anonymity among receivers) and the security requirements of blockchain system design (e.g. privacy protection of transactions content), BE/MRE can be integrated into our proposal to protect transactions' confidentiality.

Although various BE schemes (e.g. (Fiat et al., 1993; Boneh et al., 2005)) and MRE schemes (e.g. (Kurosawa et al., 2002; Bellare et al., 2003)) have been proposed, a secure certificateless multireceiver encryption (CL-MRE) is more suitable for a decentralization instantiation. This can avoid the single point failure issue, as it does not need a trusted certificate authority (may be compromised) to issue certificates. Hence, we propose using a secure and efficient CL-MRE, such as the scheme presented by Islam et al. (2015), in our BSeIn, which is designed without the bilinear pairing and map-to-point hash function. Here, we give a description of the CL-MRE scheme, which consists of a sender $\mathscr{S}$, several receiver $\mathscr{R}_1, \mathscr{R}_2, \ldots, \mathscr{R}_t$ and a $\mathscr{PKG}$.

- CL-MRE.Setup: This algorithm invoked by $\mathscr{PKG}$ takes as input a security parameter 1^k, and selects a set of non-singular elliptic curve points $\mathbb{E}(\mathbb{F}_p)$ over prime field $\mathbb{F}_p$ and a cyclic group $\mathbb{G}_p$ of prime order p. P is a generator of $\mathbb{E}(\mathbb{F}_p)$ of order n. Then, it randomly chooses $s \in \mathbb{Z}_p^*$ as the master key and computes $P_0 = sP$ as the public key. Now, it selects three one-way hash function $\mathscr{H}_0, \mathscr{H}_1, \mathscr{H}_2$: $\{0, 1\}^* \to \mathbb{Z}_p^*$ and a secure symmetric cryptography AES with the encryption/decryption AES.Enc$_{key}(\cdot)$/AES.Dec$_{key}(\cdot)$, where key is a symmetric key. Finally, it returns $(\mathbb{F}_p, \mathbb{E}(\mathbb{F}_p), \mathbb{G}_p, P, P_0,$ AES. Enc$_{key}(\cdot)$, AES. Dec$_{key}(\cdot)$, $\mathscr{H}_0, \mathscr{H}_1, \mathscr{H}_2)$ as the public parameters, and $msk = s$ is kept by $\mathscr{PKG}$ secretly.
- CL-MRE.SVG: This secret value generation algorithm is invoked by a user i with identity ID_i, and will randomly choose $x_i \in \mathbb{Z}_p^*$ and compute $P_i = x_iP$. Then, it returns (x_i, P_i) as i's secret/public key pair.
- CL-MRE.PPKG: This partial private key generation algorithm invoked by $\mathscr{PKG}$ takes as input the public information (ID_i, P_i) of user i. Randomly chooses $t_i \in \mathbb{Z}_p^*$ and computes $T_i = t_iP$, $l_i = \mathscr{H}_0(ID_i, T_i, P_i)$, $d_i = (t_i + sl_i) \bmod p$. It returns (d_i, T_i) as the partial private/identity-based public key pair of user i. (Note that $\mathscr{PKG}$ replies (d_i, T_i) to user i through a secure channel).
- CL-MRE.KG: According to both CL-MRE.SVG and CL-MRE.PPKG algorithms, this key generation algorithm sets (sk_i, pk_i) as i's private/public key pair, where $sk_i = (d_i, x_i)$, $pk_i = (P_i, T_i)$.
- CL-MRE.Enc: This encryption algorithm is invoked by a sender $\mathscr{S}$ to generate a ciphertext on message m for $\mathscr{R}_1, \mathscr{R}_2, \ldots, \mathscr{R}_t$. Firstly, it randomly chooses $r \in \mathbb{Z}_p^*$ and computes $R = rP$, $l_i = \mathscr{H}_0(ID_i, T_i, P_i)$, $u_i = r(T_i + l_iP_0 + P_i)$, $\alpha_i = \mathscr{H}_1(R, u_i)$, where $i = 1, 2, \ldots, t$. Then, it randomly chooses $\theta \in \mathbb{Z}_p^*$ and calculates $f(x) = \Pi_{i=1}^{t}(x - \alpha_i) + \theta \pmod p = x^t + c_{t-1}x^{t-1} + \cdots + c_1x + c_0$, where $c_i \in \mathbb{Z}_p^*$. And, it computes $\beta = \mathscr{H}_2(R, \theta)$, $\delta =$ AES. Enc$_{\beta}(m)$, $\tau = \mathscr{H}_3(c_0, c_1,\ldots,c_{t-1}, m, \theta, R, \delta)$. Finally, it returns the ciphertext $(c_0, c_1, \ldots, c_{t-1}, R, \delta, \tau)$.
- CL-MRE.Dec: This decryption algorithm is invoked by each receiver $\mathscr{R}_i$ independently. It takes as input $(c_0,\ldots,c_{t-1}, R, \delta, \tau)$ and computes $u_i = (d_i + x_i)R$, $\alpha_i = \mathscr{H}_1(R, u_i)$, $f(x) = x^t + c_{t-1}x^{t-1} + \cdots + c_1x + c_0$ and $\theta = f(\alpha_i)$, $\beta = \mathscr{H}_2(R, \theta)$, $m =$ AES. Dec$_{\beta}(\delta)$ and $\tau' = H_3(c_0,\ldots,c_{t-1}, m, \theta, R, \delta)$. Then, it returns m as the message if $\tau' = \tau$ or returns $\perp$ to indicate the input ciphertext invalid.

Under the assumption that the computational Diffie-Hellman problem is computationally hard to solve, this scheme not only provides confidentiality and anonymity, but also both forward and backward secrecy. We refer interested reader to (Islam et al., 2015) for more details.

3.2. Permission data hash table PDHT

For a fine-grained access control, we introduce a PDHT, $\mathscr{H}$: $\{0,1\}^{256} \rightarrow \{0,1\}^{N} \times \{0,1\}^{N}$ (i.e. *hash_of predicate*: *predicate* × *policy*), where $N \gg 256$ and can store sufficiently large data. A *predicate* can map to several *policy*s, which represents that an access rule on attributes needs to satisfy in order to obtain the permission of *policy*s. For instance, "A and (B or C)" mapping to "*efID*‖*all*" means that someone owns attributes "A" and "B" (or "A" and "C") can have full control on a device (e.g. exhaust fan) with identity "*efID*". The implementation of PDHT is similar to the off-blockchain key value store, which can be stored securely in the cloud storage. Only the manager (i.e. owner of the smart contract) can update PDHT through communications with smart contract (see Section 3.6).

3.3. Transaction

A transaction in the Bitcoin system is a signature of transactional information that mainly includes sender's address, receiver's address and transfer amount. However, transactions in our system are used to carry instructions, such as querying, storing, and operating data. There are two transaction types in our system, as described below.

1. *Request Transaction (RTx)*: As shown in Table 1, a *RTx* in our proposal comprises two cryptographic mechanisms, namely: CL-MRE and ABS. The messages include the following:
   - "*from*" represents the request's ABS information like "predicate" for verifying a ABS signature,
   - "*to*" is the goal address (e.g. smart contract address),
   - "*type*" is the request type (e.g. "read" or "control"),
   - "*one-time_pk*" is the user's one-time public key used to encrypt message from the cloud or industrial network,
   - "*device_info*" is the identity of the target devices,
   - "*order*" is the actual control order request such as "turn on" or "suspend" (when the type is "read", this parameter will be set to NULL), and
   - "*data*" is the function or other parameters to trigger a smart contract.

   If a user wishes to request or control a device, then he/she computes a ciphertext on the relevant messages using CL-MRE together with a ABS signature. That is, *RTx* = ABS.Sign(*from*‖*to*‖*type*‖CL-MRE.Enc(*one_time_pk*‖*device_info*‖*order*‖*data*)), where ABS.Sign and CL-MRE.Enc are the signing and encryption algorithms of ABS and CL-MRE respectively.

2. *Permission-update Transaction (PuTx)*: A *PuTx* is utilized to update the policies of different predicates to grant or invoke a permission, which has a similar structure with a *RTx*. The different components are the relevant messages that consist of a predicate and the policy (as shown in Table 2). Generally, a predicate corresponds to a policy (e.g. a mapping from "(A or B) and (C or D)" to "read" means that someone's attributes satisfy "(A or B) and (C or D)" have the permission to read the data). Hence, the manager can update the policies of some predicates through publishing a *PuTx* = ABS.Sign(*from*‖*to*‖CL-MRE.Enc(*predicate*‖*policy*‖*data*)).

Table 1
Structure of a request transaction.

from	*to*	*type*	CL-MRE				ABS
			one_time_pk	*device_info*	*order*	*data*	

Table 2
Structure of a permission-update transaction.

from	*to*	CL-MRE			ABS
		predicate	*policy*	*data*	

3.4. Blockchain

Blockchain (BC) is the core data structure, consisting of blocks (chronologically chained by a hash). In some literature, blockchain and Bitcoin are used interchangeably, although Bitcoin is in essence a blockchain based system. Here, we consider a public BC comprising of publicly maintained nodes (i.e. anyone can join or quit the decision of creating new block at any time). There are also permissioned blockchain (i.e. new block creation decided by some trusted nodes), which have applications in copyright management, authentication, data storage service, etc.

In both public blockchains (e.g. Bitcoin and Ethereum) and permissioned blockchains (e.g. Hyperledger), a consensus mechanism is essential for electing/selecting a miner to create a new block. Typical consensus mechanisms in public blockchains are proof of work (PoW), proof of stake (PoS) and delegated proof of stake (DPoS), and those in permissioned blockchains are byzantine fault tolerance algorithm (PBFT) and RAFT. In our design, trusted participating nodes are chosen to maintain the blockchain with the PBFT consensus mechanism (see Section 3.5). That is, BSeIn is realized in a permissioned blockchain, which can conduct hundreds of thousands transactions within seconds. Moreover, the advantages of permissioned blockchain (including increased privacy control and the ability to modify the gas cost requirement) are more appropriate and sufficient for our BSeIn design.

The structure of blocks in BSeIn is similar to that in Bitcoin, which contains a block version number, hash of the previous block, Merkle tree root and a timestamp. Since we replace PoW with PBFT in our system, the difficulty target and corresponding nonce of PoW can be omitted. Note that the hash of the previous block and the timestamp can prevent the block from being modified without detection, and the Merkle tree root is the hash of all transactions in this block to guarantee transaction integrity.

3.5. Practical byzantine fault tolerance (PBFT)

PBFT can tolerate Byzantine fault once when more than two-third of the total number of nodes in the network are honest. That is, we assume that the number of total consensus nodes and compromised nodes is n and f respectively, a consensus can be reached successfully if $n >= 3f + 1$.

Here, we briefly introduce the consensus procedure of PBFT adopted in our system, which mainly consists of the following phases (see Fig. 3).

- **Generate Block**: A leader is responsible for generating a new candidate block. In our system, consensus nodes take turns to generate the block (i.e. each node serves as the leader in order).
- **Pre-prepare (Block Data)**: The leader broadcasts the candidate block to other consensus nodes.
- **Prepare (Block Hash)**: Each node receiving the block will identify its validity and broadcast the prepare message with the hash of the block to every node.
- **Commit (Block Hash)**: After receiving sufficient prepare messages (i.e. the number of messages is more than 2/3 of the number of the

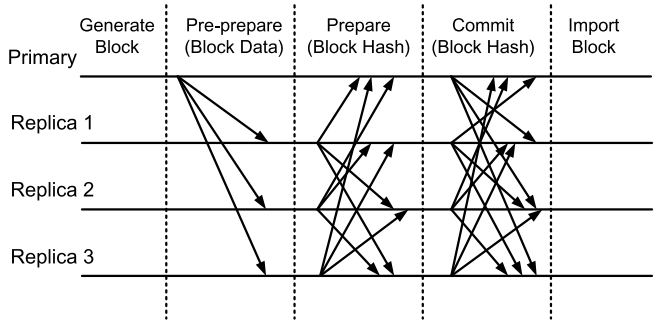

Fig. 3. Consensus procedure of PBFT.

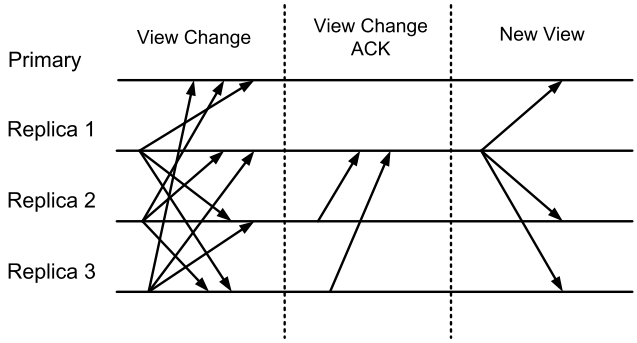

Fig. 4. Consensus procedure of a View-Change protocol.

total nodes), each computes and broadcasts the commit message to everyone.

- **Import Block**: The nodes reach a consensus on the candidate block if the number of received commit messages is not less than 2/3 of the number total nodes. That is, the block can be chained into the permissioned blockchain.

Apart from the above normal case operation, a View-Change protocol is needed when a leader or the consensus fails (i.e. the leader does not broadcast a pre-prepare message as expected or the number of received commit messages is not sufficient). The View-Change protocol is described in Fig. 4, and below.

- **View Change**: Each node receiving the exception message will broadcast a ViewChange message to other consensus nodes.
- **View Change ACK**: After receiving the ViewChange message, each node verifies the validity and replies the ViewChange-ACK message to valid nodes.
- **New View**: The leader of next round broadcasts a NewView message to other consensus nodes to process the next consensus if the number of received ViewChange-ACK messages is more than 2/3 of the total number of nodes.

3.6. Smart contracts

Smart contract, first introduced by Nick Szabo in 1994, is defined as “a computerized transaction protocol that executes the terms of a contract” (Christidis and Devetsikiotis, 2016). When integrated into the blockchain, smart contracts are programmed in a Turing complete language (e.g. Solidity) and stored in the blockchain forever. Each contract can be regarded as a database slot with a unique address that one can publish a transaction to trigger its functions for managing that database. As shown in Fig. 5, a contract creator (also the manager) compiles and deploys the smart contract into the blockchain. Then, clients query the smart contract through submission of the related parameters. Here, smart contract mainly includes the logic of storing

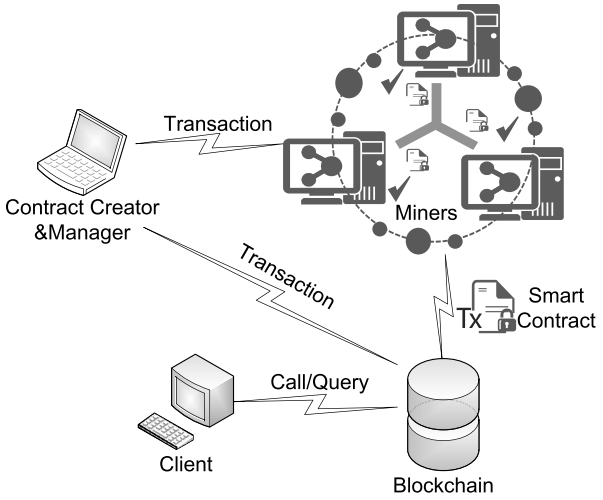

Fig. 5. Overview of the interactions within smart contract among different components.

and deleting the relevant data according to the role set by the manager.

In our proposed BSeIn, we employ smart contracts to manage the Permission Data Hash Table (PDHT) and transactions. That is, terminals can retrieve the related predicate of policy or publish the request (read or control order) through interacting with smart contracts. The corresponding smart contracts are briefly described in Algorithms 1, 2 and 3, 4.

4. Design of BSeIn

In the proposed BSeIn, the terminals can query requests without revealing their real identity and efficiently determine the gateways' authenticity. That is because the one-time public/private key pair and the message authentication code ensure that only the requester can ‘unpack’ the response from the legitimate gateways (i.e. cloud and industrial network). Here, we discuss the design details including **Initialization**, **Request Issuance**, **Chain Transaction**, **State Delivery** and **Permission Update**. Note that all involved transactions below need to interact with the smart contracts (namely, triggering the smart contracts) for invoking goal functions.

We denote AES.Enc, AES.Dec as the encryption and decryption algorithms of the Advanced Encryption Standard (AES) (Daemen and Rijmen, 2002) respectively, and H.MAC the *MAC* generation algorithm (Bellare et al., 2000).

- **Initialization**: In this phase, the factory manager or a related authority initializes system parameters as follows:
  1. **ABS Initialization**: Selects the reasonable width of monotone span programs $c_{\max}$ based on the scenario requirement. Generate the public parameters $PP = (G_1, G_2, \mathscr{H}, g, (h_0,\ldots,h_{c_{\max}}))$ and mask/public key pair (ASK, APK) by invoking the ABS.TSetup and ABS.ASetup algorithms. When each terminal first registers in this system, the authority computes a signing key $SK_{\mathscr{A}}$ based on the terminal's attributes $\mathscr{A}$ using the ABS.AttrGen algorithm.
  2. **CL-MRE Initialization**: Interacts with the involved members (e.g. permissioned nodes, cloud gateway and industrial network gateway) to generate the private/public key pairs of CL-MRE. Assuming that there are t permissioned nodes (including “vdn”s and “bkn”s), a cloud gateway and an industrial network gateway, these nodes obtain the key pairs (sk_i, pk_i) $(i = \{1,\ldots,t\})$, (sk_{cg}, pk_{cg}) and (sk_{ig}, pk_{ig}) respectively, where $sk_i = (d_i, x_i)$, $pk_i = (P_i, T_i)$ and $i = \{1, \ldots, t, cg, ig\}$ after execution of the CL-MRE.Setup, CL-MRE.SVG, CL-MRE.PPKG, CL-MRE.KG algorithms.

Algorithm 1
Part 1 - Smart Contract on PDHT.

Require: Function name, invoked parameters
Ensure: *Setting up functions*:
structure *pUnit*
% Define the structure of components in PDHT.
  hash;
  predicate;
  policy;
function PDHT()
% Constructor, automatically invokes when this smart contract is deployed.
  ownerID = sender.*pk*;
  pUnit t[];
  len = 0;
  return 1;
function updatePDHT(*pk*, *predicate*, *policy*)
% Invoked by the manager to update PDHT.
  if *pk* ≠ *owner ID* **then**
    return 0;
  else
    if Exist(*t[i].hash* = $\mathscr{H}$(*predicate*)) **then**
      if *policy* = *NULL* **then**
        Delete(*t*[*i*]);
        len − −;
      else
        t[*i*]. *policy* = *policy*;
    else
      if *policy* = *NOTNUL* **then**
      len + +;
      t[*len*]. *hash* = $\mathscr{H}$(*predicate*);
      t[*len*]. *policy* = *policy*;
    return 1;

Algorithm 2
Part 2 - Smart Contract on PDHT.

function getPDHT(*policy*)
% Invoked by someone to obtain the predicate to pass through policy.
  if *policy* ∈ *t*[*i*]. *policy* **then**
    return *t*[*i*]. *predicate*;
  else
    return 0;
function checkPDHT(*predicate*, x_p)
% Invoked by someone to decide whether the predicate is permissioned for x_p.
  if Exist(*t*[*i*]. *hash* = $\mathscr{H}$(*predicate*)) and $x_p \in t[i]$. *policy* **then**
    return 1;
  else
    return 0;

Algorithm 3
Part 1 - Smart Contract on TX.

Require: Function name, invoked parameters
Ensure: *Setting up functions*:
function TX()
% Constructor, as mentioned above.
  setMiner [] = {pk_1, pk_2,...,pk_m};
  vTx [];

Algorithm 3 (*continued*)

  vTxLen = 0;
  uvTx [];
  uvTxLen = 0;
  return 1;
function uploadTX(*pk*, *tx*, *k*)
% Invoked by miners to submit the validated transactions (*i.e.* $k = 1$) *or terminate the invalid transactions* (i.e. $k = 0$).
  if $k = 1$ **then**
    if *pk* ∈ *setMiner* **then**
      vTxLen + +;
      vTx[*vTxLen*] = *tx*;
    else
      return 0;
  else
    uvTxLen + +;
    uvTx[*uvTxLen*] = *tx*;
  return 1;

Algorithm 4
Part 2 - Smart Contract on TX.

function getTX(*k*)
% Invoked by someone to obtain the transactions.
  if $k = 1$ **then**
    tx = *vTx*[*vTxLen*];
    vTx[*vTxLen*] = 0;
    vTxLen − −;
else
    tx = *uvTx*[*uvTxLen*];
    uvTx[*uvTxLen*] = 0;
    uvTxLen − −;
  return *tx*

3. **Contract Deployment**: Takes as input the designed smart contracts on PDHT and TX, compile and deploy them as transactions into the blockchain. Once smart contracts pass the validation process, they will be recorded in the blockchain forever by the miners, which own their individual addresses (e.g. *PID* and *TID* respectively).

- **Request Issuance**: Assuming that a terminal, Alice, wishes to publish a request on device, she executes the following steps.

1. Invokes the getPHDT algorithm in the smart contract to obtain the goal predicate γ corresponding to the policy of goal device. If Alice's attributes satisfy the predicate, then she can publish a reasonable request and proceed to the next step.
2. Generates a fresh public/private key pair (pk_u, sk_u) by invoking the key generation algorithm (denoted as PKG) in (Elgamal, 1985). Prepare all components for constructing the transaction based on Alice's requirement (i.e. the contexts involved in a request transaction 1). For example, “to” is padded by *TID*, “data” refers to function uploadTX and parameters $pk = \gamma$, $k = 0$, which shows that this transaction will trigger the function uploadTX in “Smart Contract on TX” and be stored in invalid transactions *uvTx*[].
3. Executes the CL-MRE.Enc algorithm using receivers' public keys $pk_i = (P_i, T_i)$, $i = \{1,...,t, cg, ig\}$ and sign all the data based on the ABS.Sign algorithm using her attribute secret keys $SK_{\mathscr{A}}$.
4. Broadcasts this transaction to the blockchain network and the target smart contract gives the corresponding response. That is, updating the invalid transactions in “Smart Contract on TX”.

- **Chain Transaction**: The phase of chaining transactions contains two steps. That is:

1. For validating each transaction in *uvTx*[], the “vdn” first utilizes

getTX(0) to obtain invalid transactions *uvTx*[] from "Smart Contract on TX". Then, decrypt a *uvTx*[*i*] to obtain *device_info* and *order* by invoking CL-MRE.Dec with his secret key sk_i. If the embedded predicate γ in *uvTx*[*i*] is permissioned for the goal *device_info* with *order*, and attribute-based signature involved is valid under the verification of ABS.Ver algorithm, then the "vdn" updates *uvTx*[*i*] into a validated one using uploadTX(*pk*, *uvTx*[*i*], 1), where *pk* is the current *vdn*'s public information. Otherwise, discard it.

2. The "bkn"s utilize the PBFT consensus mechanism for chaining the transactions. That is, the current leader repeatedly executes getTX (1) to query validated transactions *vTx*[] for a pending block. Then, a consensus is reached on this block if, and only if, at least two-third of total "bkn"s approve the pending block. Finally, the leader chains this block to the blockchain.

- **State Delivery**: Cloud and industrial network gateways monitor the blockchain for new chained transactions. Once the terminal publishes a new valid request, the corresponding transaction generated and chained in the blockchain will be received by the cloud and industrial network. They individually respond to the terminal according to the type of transaction as follows.

1. The cloud gateway is responsible for replying data to the requester when "type" is "read". He/she first performs the decryption using the CL-MRE.Dec algorithm using his/her secret key sk_{cg} and gets *one_time_pk*, *device_info*, where *one_time_pk* is denoted as pk_u. Then, the status information or related message (called *result_data*) about the goal *device_info* is retrieved from the local cloud storage. Next, the Diffie-Hellman key $KEY = pk_u^{d_{cg}}$ is generated which can then be used to encrypt *result_data* by invoking AES.Enc (denoted as *EnData* = AES.Enc (*KEY*, *result_data*) with the corresponding MAC (denoted as *MAC* = H.MAC(*EnData*, *KEY*)) for providing confidentiality and authentication. That is, the final response is (*EnData*, *MAC*).
2. The other type of transactions in the blockchain (i.e. "control") are handled by the industrial network gateway. The first operation is to decrypt transactions, which is the same as that of cloud gateway but using his/her secret key sk_{ig} instead. Upon the decrypted *one_time_pk*, *device_info*, *control*, he/she communicates with the target resource to execute the control order. Then, different sensors configured in the physical resources will execute the commands from the gateway and sends the execution results in return. The industrial network gateway needs to compute the ciphertext and MAC as the above process is performed by the cloud gateway. The difference is that *EnData* and *MAC* in the response (*EnData*, *MAC*) are computed according to the different Diffie-Hellman key $KEY = pk_u^{d_{ig}}$, where d_{ig} is the industrial network gateway's partial secret key.

When the terminal receives (*EnData*, *MAC*), it first computes the Diffie-Hellman key $KEY = P_i^{sk_u}$ (here, P_i is the partial public of the cloud or the industrial network gateway) using his/her one-time private key sk_u. Then, it recomputes MAC_0 = H.MAC(*EnData*, *KEY*). If $MAC_0 = MAC$ holds, then it implies that the response is not from an impersonator. Hence, it invokes AES.Dec(*KEY*, *EnData*) to decrypt *EnData* and obtain the response information about the request.

- **Permission Update**: As mentioned above, access control is mainly based on the PHDT maintained by the manager (i.e. the deployer of "Smart Contract on PHDT"). The manager can adaptively update the PHDT via the updatePDHT algorithm, which is utilized to add, delete, modify the predicates as well as corresponding policies. Only one's attributes satisfying the predicates that appear in PHDT can he/she query a permissioned request based on the matched policies.

5. Security analysis

The proposed BSeIn can satisfy all the security requirements mentioned in Section 2.2, based on the deployed cryptographic schemes (i.e. ABS, MRE, AES, and MAC).

1. **Single registration**: In our proposal, the manager generates signing keys for the users (i.e. terminals), based on their attributes at the registration phase. After that, terminals can publish requests without any further registration even when other terminals join in or update. Namely, the BSeIn provides a one-off registration.
2. **Mutual authentication**: Both cloud and industrial network gateways can authenticate a terminal based on the validation of its published transactions (i.e. whether the attribute signature is valid). In addition, the terminal can identify a legitimate cloud gateway and industrial network gateway by recomputing the MAC. Any probabilistic polynomial time adversary cannot successfully forge a valid MAC for a target message because solving CDH problem is computationally hard. Clearly, BSeIn achieves mutual authentication between terminals and gateways.
3. **User anonymity**: The anonymity of ABS utilized in BSeIn can protect the terminals' real identities from being revealed. Only attributes other than related identity information can be obtained when an attribute signature is verified. Moreover, a fresh one-time public/private key pair generated during each transaction can efficiently resist replay attacks or protect terminals from having their networking traffic monitored or collected. That is, the system can guarantee user privacy without being compromised.
4. **Fine-grained access control**: The ABS deployed in our system not only provides user anonymity, but also realizes fine-grained access control. The manager can preset the predicate corresponding to a policy, which ensures only the terminal whose attributes satisfy the predicate can execute the policy. Furthermore, the manager can revoke the permission of predicates by updating PDHT when malicious behavior is detected.
5. **Session key agreement**: The fresh one-time public/private key pair generated during each transaction is used for securing the session, and generating the MAC (i.e. serving as a session key). This can authenticate gateways as well as ensuring data confidentiality in future communications.
6. **Perfect forward secrecy**: In BSeIn, even after an attacker has successfully compromised both public/private key pairs of a terminal and cloud gateway or industrial network gateway, he/she can only reveal the ciphertext belonging to the current session. Since the one-time public/private key pair is different in each session, perfect forward secrecy is guaranteed.
7. **No verifier table**: As described above, mutual authentication only requires the terminal and gateways to generate public/private key pairs. Then, these keys are used to generate ciphertext and MAC for authentication. Clearly, there is no verifier table maintained by the registration center.
8. **No online registration center**: From the authentication process, the terminals and gateways can authenticate with each other without a registration center.
9. **Relay of current timestamp**: The proposed BSeIn utilizes hard-coded DNS servers to relay current timestamp, which can avoid the timestamp from being padded without detection/authorization.
10. **Birthday collision resilience**: In our design, we utilize the PBFT consensus mechanism in the permissioned blockchain to chain a block (i.e. permissioned participate nodes are responsible for recording). Hence, there is no 'fork' situation, which effectively avoids blocks' birthday collisions.
11. **Resilience to interception and modifications**: The published transactions are signed using the terminals' attribute-based signing keys. Once an attacker modify the signature, it will invalidate the transaction. And a MAC is binded to the response from gateways, which can also prevent any unauthorized modification on the response. Hence, the proposed design is resilience to interception and modifications.
12. **Resilience to hijacking attacks**: All transactions signed by ABS

Table 3
Statistical operations during the design and implementation process of BSeIn.

Phase	Invocation of Cryptographic Algorithms	Publication of Smart Contract	Trigger of Smart Contract
Initialization	ABS.TSetup + ABS.ASetup + ABS.AttrGen + CL-MRE.Setup + CL-MRE.KG	SC-PDHT.*deploy* + SC-TX.*deploy*	*NULL*
Request Issuance	PKG.DH + CL-MRE.Enc + ABS.Sign	*NULL*	getPDHT + uploadTX
Chain Transaction	CL-MRE.Dec + ABS.Ver + uploadTX	*NULL*	2*getTX
State Delivery	CL-MRE.Dec + AES.Enc + AES.Dec + PKG.DH + H.MAC	*NULL*	*NULL*
Permission Update	CL-MRE.Enc + ABS.Sign	*NULL*	updatePDHT

can also resist hijacking attacks, since any probabilistic polynomial time adversary is unable to tamper the context of transactions without invalidating the signatures.

13. **Resilience to other attacks**: Our proposal can resist the following attacks.
    (a) **User impersonation attacks**: Only legitimate terminals can generate a valid attribute signature for a request in BSeIn. Any impersonator disguising as terminals or gateways will be identified since the mutual authentication will fail. The manager can then revoke the permission of predicates when misbehavior is detected.
    (b) **DDoS attacks**: BSeIn inherits Bitcoin's resilience to DDoS attacks, for example by limiting the block size and checking the maximum number of attribute signatures for transaction input. Moreover, the fine-grained access control ensures that a terminal can only communicate with some target resources if the predicate has been granted in PDHT. The operation of revoking can be executed in real-time, as required.
    (c) **Modification attacks**: Assuming that an attacker modifies the broadcast transaction or response message, it will be discovered and discarded because of the attribute signature and the MAC.
    (d) **Replay attacks**: A fresh one-time public/private key pair generated by terminals in each request is used to encrypt the requested resulting message and to compute the MAC. This can facilitate the terminals and gateways in detecting any replay attack due to the freshness of public/private key pairs.
    (e) **Man-in-the-middle attacks**: From the above, it is clear that BSeIn provides secure mutual authentication. Thus, it is also man-in-the-middle-attack resilience.

6. Implementation and performance evaluation

In this section, we implement our BSeIn and evaluate its performance. The main processes of BSeIn are implemented on JUICE,[2] which is an open service platform. This platform not only supports *Solidity* (designed for writing contracts such as Ethereum), but provides a user friendly web/client management and monitoring tools based on *Java* and *Javascript*. Moreover, there is a rich set of cryptographic API calls (e.g., homomorphic encryption, group signature, and zero-knowledge proof) for privacy-preserving applications. Since JUICE does not provide the API of ABS and CL-MRE at the time of this research, we will quantify the time cost of cryptographic materials and smart contracts operations separately.

First, we count the operations (i.e., invoking cryptographic algorithms, publishing and triggering smart contract) during the design and implementation process of BSeIn. In addition to the aforementioned denotations, we also denote SC-PDHT.*deploy* as the deployment of a smart contract on PDHT, and SC-TX.*deploy* the deployment of the smart contract on TX, and PKG.DH the Diffie-Hellman key generation algorithm. *NULL* denotes that the phase does not need the current operation. Table 3 summarizes the operations in the proposed system.

Then, we investigate the time cost of cryptographic algorithms (i.e., ABS, CL-MRE, ABS and MAC). Here, we leverage the pairing-based library (version 0.5.12)[3] and the GNU multiple precision arithmetic library (version 6.0.0a)[4] for our simulation, among which Type A pairings are constructed on the curve $y^2 = x^3 + x$ over the field $\mathbb{F}_q$ for some prime $q = 3 \bmod 4$. Table 4 presents information about the simulation platform. Measurements are given as the average of 1000 test runs, and Table 5 presents the findings. Note that we employ a simple predicate "(A and B) or C" and $r = 3$, $c = 2$ in the simulation of ABS, and three receivers $t = 3$CL-MRE.

We also develop a software prototype based on JUICE to evaluate BSeIn as shown in the following steps:

1. We utilize JUICE (*client version*) to build a private test chain comprising only permissioned nodes in three servers. As shown in Table 6, the server configuration mainly consists of *nginx-1.11.3* (a high performance HTTP and reverse proxy server), *truffle-2.1.1* (a development framework for the Ethereum) and *JUICE-client* (a client version of JUICE), where nginx-1.11.3 only needs to be deployed in

Table 4
Simulation platform.

Operating System	Ubuntu 16.04
CPU	Intel (R) Core (TM) i7-6700 CPU @ 3.40 GHZ
Memory	3 GB RAM
Program language	C

Table 5
Time cost (in s) of cryptographic algorithms.

ABS				
Algorithm	ABS.TSetup & ASetup	ABS.AttrGen	ABS.Sign	ABS.Ver
Max Time	0.042295	0.023485	0.057023	0.040100
Min Time	0.035256	0.014279	0.039682	0.028587
Average Time	0.037656	0.01708	0.044016	0.03386
CL-MRE				
Algorithm	CL-MRE.Setup	CL-MRE.KG	CL-MRE.ENC	CL-MRE.Dec
Max Time	0.012775	0.032023	0.035571	0.016423
Min Time	0.006246	0.019391	0.020425	0.008314
Average Time	0.007586	0.024755	0.024778	0.012466
DH-PKG & AES & MAC				
Algorithm	PKG.DH	AES.Enc	AES.Dec	H.MAC
Max Time	0.004931	0.005116	0.000093	0.000121
Min Time	0.000719	0.000355	0.000010	0.000003
Average Time	0.000937	0.000445	0.000012	0.000006

[2] https://www.juzhen.io/.

[3] http://crypto.stanford.edu/pbc/

[4] https://gmplib.org/.

Table 6
Information of permissioned nodes in the private test chain.

No.	Operating System	CPU	Memory	LAN IP	Server Configuration
1	Ubuntu 16.04	Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70 GHz	8 GB RAM	192.168.1.237	1 *nginx-1.11.3* 2 *truffle-2.1.1* 3 *JUICE-client*
2	Ubuntu 16.04	Intel(R) Xeon(R) CPU X3320 @ 2.50 GHz	4 GB RAM	192.168.1.238	1 *truffle-2.1.1* 2 *JUICE-client*
3	Ubuntu 16.04	Intel(R) Xeon(R) CPU E5-2603 v3 @ 1.60 GHz	32 GB RAM	192.168.1.239	1 *truffle-2.1.1* 2 *JUICE-client*

Table 7
Time cost (in s) of cryptographic algorithms.

Operation	PHDT	TX	uploadTX	getTX	updatePDHT	getPDHT
Max Time	6.751	6.127	2.55	3.137	3.231	3.147
Min Time	6.049	5.388	1.660	1.521	1.678	1.455
Average Time	6.405	5.631	2.550	2.191	2.491	2.239

Table 8
Time cost (in s) of phases in BSeIn.

Phase	Time
Initialization	12.123
Request Issuance	4.810
Chain Transaction	6.978
State Delivery	0.013
Permission Update	2.559

the main node (i.e. No.1), whereas *nginx-1.11.3* and *JUICE-client* are deployed in all the servers. In addition, a JUICE-Client Software (Microsoft Windows Version) should be installed for managing and monitoring transactions or contracts in the chain. Due to page length restrictions, we will omit details of the deployment process.

2. We deploy the smart contracts in BSeIn using solidity (i.e. smart contracts on PDHT and TX). Note that smart contracts are published in the local computer (see Table 4 for system information) via a permissioned node (No.1 is chosen in our simulation). With *Web3j* (a lightweight library for Java applications), the contracts can be easily placed on the test chain without the need to write additional integration codes for Java and Solidity.
3. Since Solidity only provides accuracy up to one second[5] we obtain the time cost using the *shell* script and *javascript* instead. For consistency, we also run 1000 test times to measure the approximate time cost of operations relating to smart contracts (i.e. SC-PDHT.*deploy*, SC-TX.*deploy*, uploadTX, getTX, updatePDHT, getPDHT). The findings are presented in Table 7.

Table 8 summarizes the time cost of BSeIn in each phase according to the summarization in Table 3 and the corresponding operation time costs in Tables 5 and 7 We can find that the main time cost is due to the operations of smart contracts, especially the deployment of contracts. As a result, the time cost of all the phases (but State Delivery) in BSeIn are in terms of seconds. The delay may not be tolerated by a human user, however in JUICE, each function call is equivalent to publishing a transaction. In other words, the calling results are returned after they are chained into the blockchain. Moreover, the test time cost includes the transaction generation time, that is, our simulation is padding the data in the defined transaction format of JUICE. Therefore, the time cost of all phases, with the exception of Initialization, should be in terms of milliseconds if a specialized platform is designed for BSeIn is used.

[5] http://solidity.readthedocs.io/en/develop/units-and-global-variables.html (last accessed February 5th, 2018).

7. Conclusion

Industry 4.0 is not a far fetched concept, and the complexity of such Industry 4.0 systems will require a more robust security solution.

To establish a secure remote user authentication with fine-grained access control for Industry 4.0 applications, we proposed a novel blockchain-based framework (BSeIn) in this paper. The proposed framework leverages the underpinning characteristics of blockchain as well as several cryptographic materials to realize a decentralized, privacy-preserving and auditable solution. Specifically, we utilized ABS (to anonymously authenticate terminals), MAC (to efficiently authenticate gateways, and CL-MRE (to provide confidentiality of the requested messages). We then demonstrated the security of BSeIn and evaluated the performance of the prototype.

Future research will include exploring to optimize the performance of the proposed system using hardware implementation, and collaborating with a smart factory operator to implement and evaluate our prototype (most probably, a hybrid hardware and software based prototype) in a real-world setting.

Acknowledgements

The work was supported by the National Key Research and Development Program of China (No. 2017YFB0802500), the National Natural Science Foundation of China (Nos. 61572379, 61501333), the Guangxi Key Laboratory of Cryptography and Information Security (No. GCIS201608) and the Natural Science Foundation of Hubei Province of China (No. 2015CFB257).

References

Aitzhan, N.Z., Svetinovic, D., 2016. Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams. IEEE Trans. Dependable Secure Comput. 99 1–1.

Azouvi, S., Al-Bassam, M., Meiklejohn, S., 2017. Who am i? secure identity registration on distributed ledgers. In: García-Alfaro, J., Navarro-Arribas, G., Hartenstein, H., Herrera-Joancomartí, J. (Eds.), Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2017 International Workshops, DPM 2017 and CBT 2017, Oslo, Norway, September 14-15, 2017, Proceedings, Vol. 10436 of Lecture Notes in Computer Science. Springer, pp. 373–389.

Bellare, M., Kilian, J., Rogaway, P., 2000. The security of the cipher block chaining message authentication code. J. Comput. Syst. Sci. 61 (3), 362–399.

Bellare, M., Boldyreva, A., Staddon, J., 2003. Randomness re-use in multi-recipient

encryption schemeas. In: Desmedt, Y. (Ed.), Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings, Vol. 2567 of Lecture Notes in Computer Science. Springer, pp. 85–99.
Boneh, D., Gentry, C., Waters, B., 2005. IACR Cryptology ePrint Archive. Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys, vol. 2005. pp. 18.
Chen, Y., Tang, C., 2013. IACR Cryptology ePrint Archive. Construction of Multiplicative Monotone Span Program, vol. 2013. pp. 777.
Chen, X., Li, J., Weng, J., Ma, J., Lou, W., 2016. Verifiable computation over large database with incremental updates. IEEE Trans. Comput. 65 (10), 3184–3195.
Christidis, K., Devetsikiotis, M., 2016. Blockchains and smart contracts for the internet of things. IEEE Access 4, 2292–2303.
Daemen, J., Rijmen, V., 2002. The Design of Rijndael: AES - the Advanced Encryption Standard. Information Security and Cryptography. Springer.
Elgamal, T., 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theor. 31 (4), 469–472.
Fiat, A., Naor, M., 1993. Broadcast encryption. In: Stinson, D.R. (Ed.), Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings, Vol. 773 of Lecture Notes in Computer Science. Springer, pp. 480–491.
Groth, J., Sahai, A., 2008. Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (Ed.), Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings, Vol. 4965 of Lecture Notes in Computer Science. Springer, pp. 415–432.
Haller, N.M., 1995. The s/key one-time password system. In: Proceedings of the Internet Society Symposium on Network and Distributed Systems, pp. 151–157.
He, D., Zeadally, S., Kumar, N., Wu, W., 2016. Efficient and anonymous mobile user authentication protocol using self-certified public key cryptography for multi-server architectures. IEEE Trans. Inf. Forensics Secur. 11 (9), 2052–2064.
Huang, Z., Liu, S., Mao, X., Chen, K., Li, J., 2017. Insight of the protection for data security under selective opening attacks. Inf. Sci. 412, 223–241.
Islam, S.H., Khan, M.K., Al-Khouri, A.M., 2015. Anonymous and provably secure certificateless multireceiver encryption without bilinear pairing. Secur. Commun. Network. 8 (13), 2214–2231.
Kopp, H., Mödinger, D., Hauck, F., Kargl, F., Bösch, C., 2017. Design of a privacy-preserving decentralized file storage with financial incentives. In: 2017 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2017, Paris, France, April 26-28, 2017. IEEE, pp. 14–22.
Kurosawa, K., 2002. Multi-recipient public-key encryption with shortened ciphertext. In: Naccache, D., Paillier, P. (Eds.), Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings, Vol. 2274 of Lecture Notes in Computer Science. Springer, pp. 48–63.
Li, C.-T., Hwang, M.-S., 2010. An efficient biometrics-based remote user authentication scheme using smart cards. J. Netw. Comput. Appl. 33 (1), 1–5.
J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, H. Ye, Significant permission identification for machine learning based android malware detection, In IEEE Transactions on Industrial Informatics http://dx.doi.org/10.1109/TII.2017.2789219.
Li, J., Man, H.A., Susilo, W., Xie, D., Ren, K., 2010. Attribute-based Signature and its Applications, in: 5th International Symposium on Acm Symposium on Information,computer and Communications Security. pp. 60–69.
Li, X., Niu, J., Khan, M.K., Liao, J., 2013. An enhanced smart card based remote user password authentication scheme. J. Netw. Comput. Appl. 36 (5), 1365–1371.
Li, J., Liu, Z., Chen, X., Xhafa, F., Tan, X., Wong, D.S., 2015. L-encdb: a lightweight framework for privacy-preserving data queries in cloud computing. Knowl.-Based Syst. 79, 18–26.
Li, M., Weng, J., Yang, A., Lu, W., 2017. IACR Cryptology ePrint Archive. Crowdbc: A Blockchain-based Decentralized Framework for Crowdsourcing, vol. 2017. pp. 444.
Liang, X., Shetty, S., Tosh, D.K., Kamhoua, C.A., Kwiat, K.A., Njilla, L., 2017. Provchain: a blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017, Madrid, Spain, May 14-17, 2017. IEEE Computer Society/ACM, pp. 468–477.
Liu, J., Zhou, A., Gao, M., 2008. A new mutual authentication scheme based on nonce and smart cards. Comput. Commun. 31 (10), 2205–2209.
Maji, H.K., Prabhakaran, M., Rosulek, M., 2010. IACR Cryptology ePrint Archive. Attribute-based Signatures, vol. 2010. pp. 595.
M'Raïhi, D., Bellare, M., Hoornaert, F., Naccache, D., Ranen, O., 2005. HOTP: an hmac-based one-time password algorithm. RFC 4226, 1–37.
Qiu, M., Gao, W., Chen, M., Niu, J., Zhang, L., 2011. Energy efficient security algorithm for power grid wide area monitoring system. IEEE Trans. Smart Grid 2 (4), 715–723.
Qiu, M., Su, H., Chen, M., Ming, Z., Yang, L.T., 2012. Balance of security strength and energy for a PMU monitoring system in smart grid. IEEE Commun. Mag. 50 (5), 142–149.
Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E., 1996. Role-based access control models. IEEE Comput. 29 (2), 38–47.
Tsai, C., Lee, C., Hwang, M., 2006. Password authentication schemes: current status and key issues. I. J. Netw. Secur. 3 (2), 101–115.
Vaidya, B., Park, J.H., Yeo, S., Rodrigues, J.J.P.C., 2011. Robust one-time password authentication scheme using smart card for home network environment. Comput. Commun. 34 (3), 326–336.
Wang, S., Wan, J., Li, D., Zhang, C., 2016. Implementing smart factory of industrie 4.0: an outlook. IJDSN 12 (1), 1–10.
Zyskind, G., Nathan, O., Pentland, A., 2015. Decentralizing privacy: using blockchain to protect personal data. In: 2015 IEEE Symposium on Security and Privacy Workshops, SPW 2015, San Jose, CA, USA, May 21-22, 2015. IEEE Computer Society, pp. 180–184.

Chao Lin received his Bachelor and Master degrees from the School of Mathematics and Computer Science, Fujian Normal University in 2013 and 2017, respectively. Currently, he is pursuing his Ph.D. degree in School of Cyber Science and Engineering, Wuhan University. His research interests mainly include authentication of graph data and blockchain security.

Debiao He received his Ph.D. degree in applied mathematics from School of Mathematics and Statistics, Wuhan University in 2009. He is currently a Professor of the School of Cyber Science and Engineering, Wuhan University. His main research interests include cryptography and information security, in particular, cryptographic protocols. His work has been cited more than 3400 times at Google Scholar (H-Index: 32). He is also Associate Editors of IET Wireless Sensor Systems, Computers & Electrical Engineering, Journal of Medical Systems and Journal of Information Security and Applications.

Xinyi Huang received the Ph.D. degree from the School of Computer Science and Software Engineering, University of Wollongong, Australia. He is currently a Professor with the School of Mathematics and Computer Science, Fujian Normal University, China, and the Co-Director of Fujian Provincial Key Laboratory of Network Security and Cryptology. He has authored over 100 research papers in refereed international conferences and journals. His research interests include applied cryptography and network security. His work has been cited more than 4300 times at Google Scholar (H-Index: 38). He is also an Associate Editor of the IEEE Transactions on Dependable and Secure Computing.

Kim-Kwang Raymond Choo (SM′15) received his Ph.D. in Information Security in 2006 from Queensland University of Technology, Australia. He currently holds the Cloud Technology Endowed Professorship at The University of Texas at San Antonio, and is an Adjunct Associate Professor at University of South Australia. He is the recipient of various awards including ESORICS 2015 Best Paper Award, Winning Team of the Germany's University of Erlangen-Nuremberg (FAU) Digital Forensics Research Challenge 2015, 2014 Highly Commended Award by the Australia New Zealand Policing Advisory Agency, Fulbright Scholarship in 2009, 2008 Australia Day Achievement Medallion, and British Computer Society's Wilkes Award in 2008. He is also a Fellow of the Australian Computer Society.

Athanasios V. Vasilakos is currently a Professor with the Lulea University of Technology, Sweden. He has authored or coauthored over 200 technical papers in major international journals and conferences, five books, and 20 book chapters. He received the Web of Science 2017 Highly Cited Researcher award. He served as a General Chair, a Technical Program Committee Chair for many international conferences. He served or is serving as an Editor of the IEEE renowned transactions and, also a General Chair of the Council of Computing of the European Alliances for Innovation.