Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/MBE

MBE, 16(5): 3561–3594.
DOI: 10.3934/mbe.2019178
Received: 16 January 2019
Accepted: 27 March 2019
Published: 22 April 2019

Research article

Verifiable fully outsourced attribute-based signcryption system for IoT eHealth big data in cloud computing

Kittur Philemon Kibiwott[1], Yanan Zhao[1], Julius Kogo[2] and Fengli Zhang[1,*]

[1] School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu, 610054, China

[2] Department of Mathematics and Computer Science, University of Eldoret P. O. Box 1125-30100 Eldoret, Kenya

* **Correspondence:** Email: fzhang@uestc.edu.cn.

Abstract: The entrance of Internet of Things (IoT) technologies to healthcare industry has impacted the explosion of eHealth big data. Cloud computing is widely considered to be the promising solution to store this data because of the presence of abundant resources at a lower cost. However, the privacy and security of the IoT generated data cannot be ensured as the data is kept far from the owner's physical domain. In order to resolve the underlined issues, a reassuring solution is to adopt attribute-based signcryption (ABSC) due to the desirable cryptographic properties it holds including fine-grained access control, authentication, confidentiality and data owner privacy. Nonetheless, executing expensive computation such as pairing and modular exponential operations in resource-constrained IoT device platform can be too taxing and demanding. To address the challenges stated above, we proposed in this paper, a more efficient scheme where computation power is borrowed from the cloud server to process expensive computations while leaving simple operations to local users. In order to realize this, trusted attribute authority, signcryptor and designcryptor outsources to the cloud expensive tasks for key generation, signcryption and designcryption respectively. Moreover, validity and correctness of outsourced computations can be verified by employing outsourcing verification server. Security analysis, comparisons evaluation and simulation of the proposed scheme is presented. The output demonstrates that it is efficient, secure and therefore suitable for application in resource-constrained IoT devices.

Keywords: Internet of Things (IoT); outsourcing computation; attribute-based signcryption; cloud computing; eHealth big data

1. Introduction

With the rapid increase in the adoption of Internet of Things (IoT) [1–4] technologies and digiti-

zation of medical records in today's modern world of wireless communications, a paradigm shift can now be seen in health care industry. Application of IoT in eHealth aims to integrate diverse constrained [5] self-configuring medical smart devices that interacts with each other through global network architecture [6] as illustrated in Figure 1. Due to this, eHealth big data explosion is currently witnessed. To relieve Health care industries the trouble of high storage and computation costs at the same time enhance immediate sharing, the data generated from diverse IoT devices is moved to the cloud server to take advantage of the dynamic scalable resources at a lower cost. Moreover, cloud computing ensures an easy access of data anywhere anytime irrespective of the location and device. In spite of the merits supplied by the cloud, it is assumed to be insecure hence cannot be trusted with sensitive data such as IoT generated medical data. Therefore, data should be encrypted before outsourcing [7] so that in case the storage devices are compromised, the privacy of data can still be maintained. In addition, data owner may choose to enforce data access control measures permitting only the authorized users that possess certain attributes to access the shared data while on the other hand denying access to unauthorized users [8]. To realize the aforementioned goals, several significant research work have been proposed.

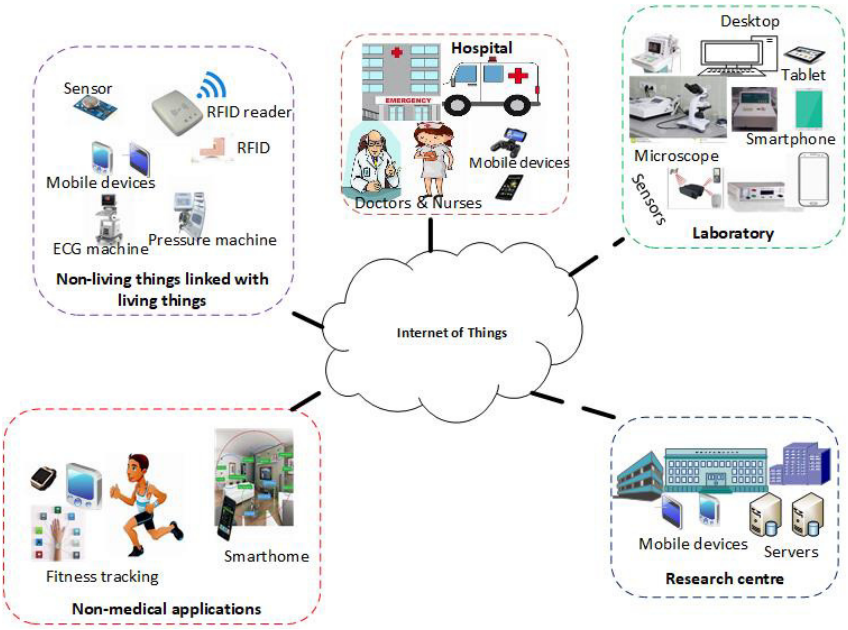


Figure 1. Framework of healthcare Internet of Things.

Attribute-based encryption (ABE) [9] which is an enhancement of identity-based encryption (IBE) [10] has attracted much wide attention recently in areas including cloud services [11, 12], IoT [13, 14], distributed environments [15] and medical systems [16] because of its rich features including fine-grained data access control and data confidentiality. Moreover, the primitive offers an additional appealing feature where data is no longer shared on one-to-one basis instead one-to-many. There are two existing complementary variants of ABE. One is Key-Policy ABE (KP-ABE) [17, 18] and the other is Ciphertext-Policy ABE (CP-ABE) [19, 20]. In KP-ABE, the cipher message is associated (encoded) with attributes set and decryption key is associated (encoded) with access structure while in Ciphertext Policy-ABE (CP-ABE), the cipher message is associated (encoded) with access structure

and decryption key is associated (encoded) with an attributes set. In comparison to KP-ABE, CP-ABE scheme is more flexible since the determination of access policy is done by the encryptor instead of key allocator [21] hence more appealing for practical applications.

Attribute-Based Signature (ABS) [22] has also widely attracted immense research attention for its fascinating feature of signing a message while simultaneously maintaining the privacy of the signer. In ABS, a signer in possession of attributes set issued by authority can sign a message using a predicate that satisfies her attributes. ABS similar to ABE is presented in two variants; Signature-Policy Attribute Based Signature (SP-ABS) [7,23] where signing key is associated (encoded) with attributes set and the plaintext message is signed by making use of the predicate that is satisfied by the set of attributes that belongs to the signing key while in the second variant Key-Policy Attribute Based Signature (KP-ABS) [24] the signing key is associated (encoded) with the predicate that is satisfied by the set of attributes while the message is associated (encoded) with set of attributes.

In real life scenario some of the applications such as health care need encryption combined with signing to achieve concurrently the flavour of confidentiality of medical data and authenticity of the data owner [25] and in most cases the anonymity of users [26] and hiding sensitive information [27,28] that belongs to patients. Medical data uploaded to the cloud and are designed for sharing may include sensitive private information such as patient's names, diseases, prescription etc. To ensure the confidentiality of these data is safeguarded, access mechanism should be provided that allows only authorized parties to access them. Simultaneously, the data users for instance government, research institutions, insurance companies and other hospitals should be convinced that the data sent originated from the actual owner to avoid downloading the corrupted data that may be sent by malicious user. As an illustration take for example a data owner "Doctor A" that outsources hospital records to the public cloud for storage. In this case the cloud has to verify whether the data was outsourced truly by an actual user that has set of attributes represented as "Government Hospital $\wedge$ level $\in \{4,5,6\}$". Whenever a staff of "Insurance B" accesses the eHealth cloud to obtain the stored data, she will be convinced that the owner of data is indeed a doctor that has certain attributes represented as "Government Hospital $\wedge$ level $\in \{4,5,6\}$" while the actual doctor's identity "Doctor A" is kept private. To realize this scenario, the viable solution is to adopt Attribute Based Signcryption (ABSC) technology, that combines flavours of ABE and ABS in single logical step. The cryptographic primitive is efficient compared to classical "encrypt-then-sign" or "sign-then-encrypt" technologies approaches [29]. The ABE part of attribute-based signcryption achieves data confidentiality while ABS part achieves privacy and authentication. However, the rising challenge is how to edge off the expensive computational cost [30] emerging from bilinear pairing and exponentiation which are introduced by primitive while concurrently supporting fine-grained access control for resource-constrained device users. The promising solution is to outsource expensive computations to the cloud. Few existing work in literature have been proposed that discusses this [31–33]. The schemes supports outsourced designcryption functionality where the blinding key is send to the cloud to transform ciphertext to a simple one. The final user performs lighter pairing computation to obtain the plaintext message. In addition to supporting designcryption, the scheme in [33] supports outsourcing of signcryption, where cloud is delegated a task of executing expensive computations and data owner performs simpler local computations.

Verification is another essential feature needful for validating the outsourced computations results to ensure proxy returns correct output. In [31, 33] only the validity of the outsourced transformed ciphertext is checked while the validity of returned outsourced generated key is not verified. Malicious

user may return invalid key without the knowledge of the data owner which might cause inaccurate findings. According to the information we have, there is no proposed work currently in literature that achieves verifiable fully outsourced attribute-based signcryption. Therefore our work proposes this new novel where we incorporated outsourcing service providers [34] and blinding key technique [31, 35] to fulfill the desired goals.

1.1. Our contributions

Taking into account the resource limitations affecting IoT devices in terms of battery, computation power, memory storage space etc, we propose in this paper a new novel verifiable fully outsourced attribute-based signcryption (VFOABSC) scheme for IoT eHealth big data system in cloud computing. In our proposed scheme, the execution of expensive computations are delegated to the powerful cloud server while lightweight computations is handled by the local IoT device user. As far as we know, this is the first proposed ABSC scheme in literature that is fully outsourced. The sequence work flow of our scheme is illustrated in Figure 2. The main contributions of this paper are:

- We introduced for the fist time a new novel VFOABSC that achieves full outsourcing of expensive ABSC computations to accommodate resource-constrained IoT devices. Initially we provide the formal framework behind the design philosophy of VFOABSC with the goal of fulfilling authentication, confidentiality and privacy of data owner simultaneously. The architecture of VFOABSC can be considered as an elaborate combination of attribute-based encryption schemes with outsourcing key generation, verification of outsourcing key generation, outsourcing signcryption and outsourcing designcryption. To realize this, we incorporated seven types of service providers that are responsible for executing expensive tasks.

- This paper proposes, an efficient ABSC scheme, that outsources the key generation tasks (both decryption and signing keys), key verification task, signcryption task and designcryption task simultaneously. Moreover, all the participants in the proposed system can verify the validity and correctness of the outsourcing computations with the support of outsourcing verification cloud service provider.

- The correctness together with security prove of the proposed scheme including its efficiency and complexity are analyzed. In addition, we also compared our scheme with other existing attribute-based signcryption schemes in terms of security requirements, outsourcing functionalities, storage overhead (size of the decryption key, size of the signing key, size of the ciphertext) and the computation overhead incurred while executing signcryption (outsourced and local) and designcryption (outsourced and local).

1.2. Organization

The remainder of this paper is arranged as follows; Section 2 provides a review of existing related work regarding the secure outsourced attribute-based cryptographic primitives (encryption, signature and signcryption). We describe preliminaries which are employed throughout this proposed paper in Section 3, we also define the formal system model and the security model of our VFOABSC scheme in this section. In Section 4 we present a concrete construction of our proposed scheme. Moreover, we

consider security proof and performance in Section 5 and Section 6 respectively. Finally we provide conclusion in Section 7.

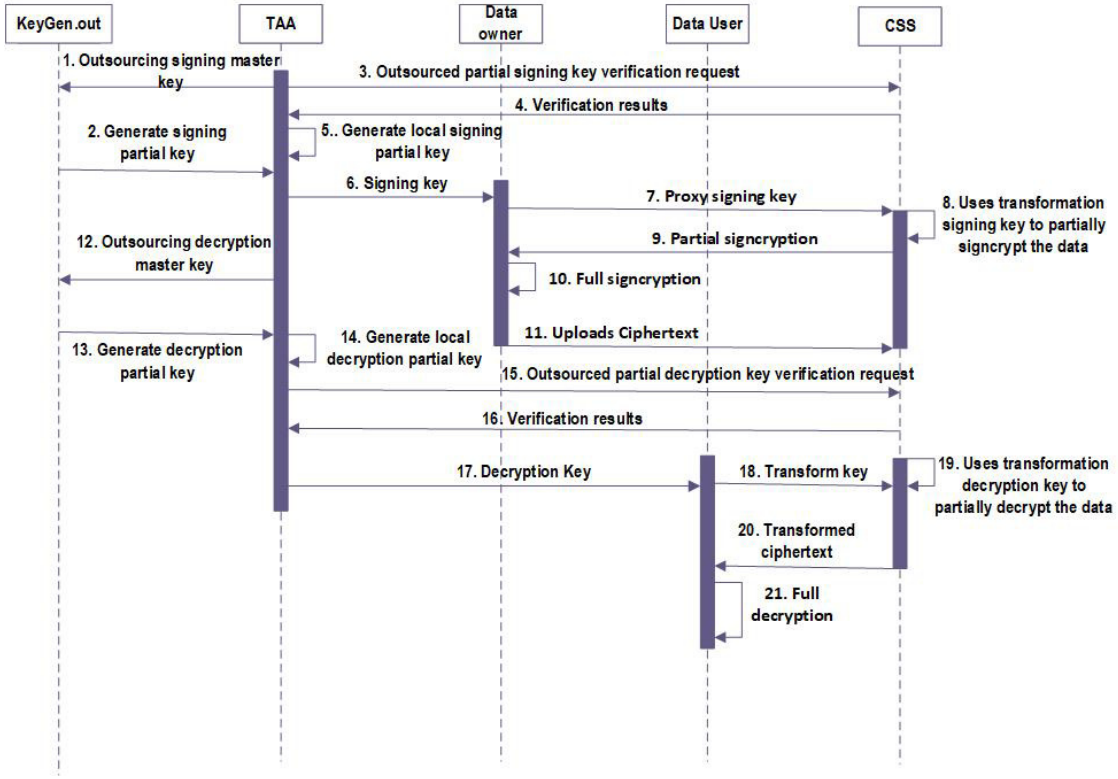


Figure 2. Sequence diagram for fully outsourced ABSC in IoT environment.

2. Related work

2.1. Attribute-Based Encryption (ABE) that securely outsources intensive computations operations.

The functionality advantages derived from ABE access control is so immense. However, it is computationally expensive. This is due to high computation overhead encountered while executing encryption and decryption operations [36–38]. To overcome this hurdle and therefore improve efficiency of lightweight devices, Green *et al.* [35] proposed a primitive that aims to offload intensive computations to the cloud service provider while leaving lighter computations to the end user. Generally the key generating algorithm is designed to return two key pair to the data user as follows:

1. A short El Gamal-form private key known as retrieving key *rk*.
2. Its paired key known as transformation key *tk*, which is send to the server for transformation purposes and its publicly known.

In their scheme, a transformation key *tk* is sent to the cloud server to transform the original ciphertext CT satisfied by end user's set of attributes or access policy into a simpler ciphertext CT'. The user spends the least computation overhead to recover the original message from the transformed ciphertext. While the scheme enhances the computational efficiency, however malicious cloud may substitute

the original ciphertext and provide the user with a transformed ciphertext from an alternate ciphertext which the cloud requires the user to decrypt. To ensure the cloud server executes the outsourced decryption honestly and returns a valid transformed ciphertext, a number of schemes have been proposed [11,39–41]. Lai *et al*. [40] and Mao *et al*. [41] separately proposed novel technique where verifiability of ciphertext is performed on a returned transformed ciphertext. In order to realize this functionality, Lai *et al*. [40] appended an extra verification instance to the existing encryption/decryption algorithm phases of ABE. The presented technique increases the computation and communication costs since encryption algorithm needs the data owner to encrypt the additional random message and also generate a checksum value associated to two equal messages. On the other hand decryption process demands the cloud server to execute the basic decryption algorithm twice and in addition requires data user to perform verification on the outsourced computations regarding encrypted messages. To overcome the stated challenges, Lin *et al*. [11] presented a more efficient ABE primitive with verifiable outsourced decryption by incorporating flavours of a symmetric-key encryption scheme, an attribute based key encapsulation mechanism, and a commitment scheme in generic model. While verifiability provides reassuring solution to outsourced decryption, nonetheless the length of ciphertext and the amount of complex pairing operations grows in proportion with the attribute size. This greatly impedes its application in resource-constrained IoT devices. To solve the underlying problem,schemes [13, 39] were proposed. Qinlong *et al*. [13] proposed a secure scheme that outsources majority of encryption, decryption and signing computations concurrently from IoT device to fog node. To save communication cost, Li *et al*. [39] presented a novel technique verifiable outsourced decryption CP-ABE scheme that has a constant ciphertext length.

Recently, Wang *et al.*[34] and Zhang *et al*. [42] independently presented efficient Ciphertext-Policy ABE (CP-ABE) schemes that can simultaneously outsource expensive key generation, encryption and decryption operations securely to the cloud. The local users incurs less computation costs.

2.2. Attribute-Based Signature (ABS)

ABS is an extension of identity-based signature, a scheme proposed and formalized by Shamir [10] where signer's identity is described using a set of descriptives attributes. In 2008, Piyi *et al*. [43] presented fuzzy identity-based signature scheme a primitive that is closely related to attribute-based signature. The scheme enables data owners to generate signatures with a subset of the attributes they own. The first ABS which achieves user's privacy where a user could sign a document by utilizing a subset of his attributes was formalized in 2008 by Guo *et al*. [44]. In this primitive, the authors claimed it to be unforgeable under adaptive message attack based on intractability strong extended Diffie-Hellman assumption. In 2009, Tan *et al*. [45] proved the scheme to be vulnerable to partial key replacement attack. First ABS scheme supporting a powerful set of predicates involving AND, OR and Threshold gates was formally introduced Maji *et al* [22]. Nonetheless, the security of this scheme is feeble as their design philosophy is solely proved in the generic group model. From that time, many similar works have been proposed [7,23,46–49]. Construction of threshold attribute-based signature applicable to small attribute universe and large attribute universe was formally introduced by Shahandashti *et al*. [50]. In this scheme, a document is signed using attributes subset while the verification of the signature is validated if the set of attributes used in signing is nearly close to the set of attributes that are used to verify.

Table 1. Notations.

ACRONYM	DESCRIPTION
$OKGSP_s$	Outsource Key Generation Service Provider (signing)
$OKGSP_d$	Outsource Key Generation Service Provider (decryption)
$OSSP$	Outsourcing Signcryption Service Provider
$ODSP$	Outsource Designcryption Service Provider
$OVSP_s$	Outsource Verification Service Provider (signing)
$OVSP_d$	Outsource Verification Service Provider (decryption)
TAA	Trusted Attribute Authority
A_s(resp. A_d)	Signing attributes (resp. Encryption attributes)
ICT	Intermediate cipherText
SCT	CipherText
SCT'	partially decrypted ciphertext
MSK	Master Key
MSK_{OKGSP_s}	Outsourced signing master key
MSK_{OKGSP_d}	Outsourced decryption master key
MSK_{TAA_s}	Local signing master key
MSK_{TAA_d}	Local decryption master key
PP	public parameters
SK_{A_s}(resp. SK_{A_d})	Private Signing Key(resp. Private decryption Key)
ISK_s	Intermediate Secret Key(signing)
ISK_d	Intermediate Secret Key(decryption)
TSK_{A_s}	Transformation Secret key (signing)
TSK_{A_d}	Transformation Secret key (decryption)
$RSKA_s$	Local signcryption signing key.
$RSKA_d$	Retrieval key used during local designcryption
$\oplus$	An exclusive-OR(XOR)
H_1, H_2, H_3, H_s	Four hash functions

2.3. Attribute-Based Signcryption

In 2010, Maji *et al.* [51] proposed the first ABSC primitive that presents merits of both ABE and ABS in one logical step to achieve confidentiality of data, fine-grained access control, and unforgeability simultaneously. To realize this scheme, they combined technologies of FIBE [9] and features of new threshold attribute-based signature. The primitive was proved to be secure under selective-predicate attack. The signing access structure of this scheme is static in the setup phase and therefore does not work so well in practice. To eliminate the weakness, ABSC which is dynamic in nature was presented by Emura *et al.* [52], where encryptor's access structures is updated without the need to re-issue the signing private keys to the users. Morever, this primitive presents another essential feature *public verifiability*, where any third party can perform verification on the validity of the ciphertext before forwarding to the intended recipient. Therefore, undesirable cost of designcrypting invalid ciphertexts are eliminated. In 2013, Hu et al. [53] proposed a novel Fuzzy Attribute-Based Signcryption (FABSC) that is applicable to Body Area Network. In this scheme, patients are allowed to specify the attributes set a physician is supposed to have in order to access a particular portion of sensitive data. In order to achieve it, the intersection between the physician's credentials and the essential parts should exceed a predetermined threshold for a physician to access the data. A threshold ABSC primitive that has constant-size ciphertexts and non-monotonic access structure and which make use of inner-product encryption was proposed in 2013 by Han *et al.* [54]. Moreover, the scheme has an

additional functionality *public verification* of ciphertext. Since then, many works on the same theme have been proposed [29,31,33,55–59]. Furthermore, ABSC primitives that outsources expensive computations to resolve the limitations confronting resource-constrained device users has also attracted many researchers [29, 31, 33]. The scheme in [31], offloads expensive computations to the cloud to realize verifiable outsourced designcryption while the final user executes lightweight computations. Compared to scheme [33] that outsources only demanding designcryption computations, scheme [29] outsources simultaneously heavy computations of signcryption and designcryption.

3. Preliminaries

In this section, we present cryptographic preliminaries definitions essential for understanding this paper.

3.1. Notations

The various notations utilized throughout this paper are illustrated in Table 1.

3.2. Access policies formats

Our scheme is constructed under access structures, monotone span program and linear secret sharing scheme policies.

3.2.1. Access structures

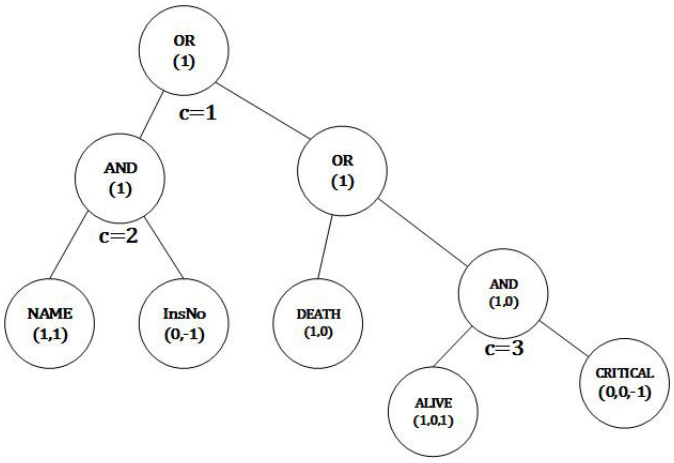
(Access structure [60]). Let $\mathbb{E}$ denote the attribute universe. A collection $\mathbb{X} \subseteq 2^{\mathbb{E}} \setminus \{\emptyset\}$ is monotone if $\forall\, \mathbb{Z}, \mathbb{W}$: if $\mathbb{Z} \in \mathbb{X}$ and $\mathbb{Z} \subseteq \mathbb{W}$,then $\mathbb{W} \in \mathbb{X}$. An access structure is a collection $\mathbb{X}$ of non-empty subsets $\mathbb{X} \subseteq 2^{\mathbb{E}} \setminus \{\emptyset\}$. The sets in $\mathbb{X}$ are called authorized sets, and the sets not in $\mathbb{X}$ are unauthorized sets.

3.2.2. Monotone span program (MSP)

MSP is a algebraic model of computation that is linear in nature [56]. To construct *MSP*, a Boolean formula is first converted into an access tree where *AND* ($\wedge$) and *OR* ($\vee$) form interior nodes while leaf nodes are the attributes [61]. As an example assume data users that can access the data from the cloud have policies such as " *Name* $\wedge$ *InsNo*" for INSURANCE COMPANY or "*DEATH* $\vee$ *ALIVE* $\wedge$ *CRITICAL*" for GOVERNMENT or "*BLOOD* $\wedge$ *SALIVA*" for RESEARCH INSTITUTION where *NAME, InsNo, DEATH, ALIVE, CRITICAL, BLOOD* and *SALIVA* are the attributes while INSURANCE COMPANY, GOVERNMENT, RESEARCH INSTITUTION are data users. The formula shows that any user with attributes set: $(NAME, InsNo)$, $(DEATH, ALIVE, CRITICAL)$, $(BLOOD, SALIVA)$ satisfies the policy. Figure 3 shows an example of eHealth big data storage system access tree for Insurance company and Government with its corresponding matrix below it. Using the method as outlined in [61], the access tree which is an input is converted into its equivalent matrix M. The labeling begins from the root node where it is labeled using vector (1) and the global counter c is assigned with value 1. The labeling is done top down as follows:

i. When the parent node is *OR* ($\vee$) gate denoted using vector υ, then its children is also denoted using υ (and the value of c remains the same).

ii. When the parent node is an *AND* (∧) gate denoted using a vector υ, then we append 0's at the end of υ (if there is a necessity) to make it of length c. Then we mark its left child as $\upsilon|1$ (in this case | denotes concatenation) and the right child is marked using the vector (0,...,0|-1), where (0,...,0) is the zero vector of length c. We should note that this two vectors sum to $\upsilon|0$. The value of c is incremented by 1. value of c is incremented when AND (∧) is encountered. Once the entire tree has been labeled, the vectors denoting the leaf nodes (attributes) compose the rows of linear sharing secret scheme (lsss) matrix. If the vectors differ in length, we have to append 0's at the end of the shorter ones to ensure that all vectors display the same length.



$$\begin{array}{c} NAME \\ InsNo \\ DEATH \\ ALIVE \\ CRITICAL \end{array} \begin{pmatrix} 1 & 1 & 0 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix}$$

Figure 3. Access tree and matrix for insurance and government.

Let Ψ: $\{0,1\}^n \rightarrow \{0,1\}$ represent a monotone function [60]. A monotone span program is represented as $\Theta = (M, \rho)$ over $\mathbb{Z}_p$. It is an $\ell \times k$ matrix $\boldsymbol{M}$ that has entries in $\mathbb{Z}_p$ and the labeling $\Phi : [\ell] \rightarrow [n]$ that links each row of $\boldsymbol{M}$ with an input variable for $(t_1, t_2, ..., t_\ell) \in \{0, 1\}^n$.

A span program allows the input if it meets the following:
$\Psi\ (t_1, t_2, ..., t_\ell) = 1 \Leftrightarrow \exists\ \varphi \in \mathbb{F}^{1\times\ell} : \varphi\mathbf{M} = [1,0,0,....,0]$
and $(\forall\ \mathrm{i}: \mathrm{x}_{\Phi(i)}) = 0 \Rightarrow \varphi_i = 0$
Basically, $\Psi(t_1, t_2, ...t_l) = 1$ provided that the rows of $\boldsymbol{M}$ indexed by $\{\mathrm{i}|\mathrm{t}_\Phi(i) = 1\}$ spans the vector [1,0,0,....,0]. Span programs can be constructed from Boolean functions[56].

3.2.3. Linear secret sharing scheme

Definition 1. *(Linear secret sharing scheme(LSSS) [60]) A secret-sharing scheme $\Pi_{\mathbb{X}}$ for the access structure $\mathbb{X}$ over a given set of attributes ϑ is called linear (in $\mathbb{Z}_p$) if*

1. *The shares derived from secret $\xi \in \mathbb{Z}_p$ for each party forms a vector over $\mathbb{Z}_p$*
2. *For every access structure $\mathbb{X}$ on ϑ, there exist a matrix $\boldsymbol{M}$ that has ℓ row size and k column size referred to as sharing-generating matrix for Π. A function ρ labels every row i of matrix $\boldsymbol{M}$ using the attributes from Π. During the generation of shares we take into consideration the column vector $\vec{\psi} = (\xi, y_2, y_3, .., y_k)$, where ξ is the secret to be shared into ℓ parts, and $y_2, y_3, .., y_k$ are randomly chosen from $\mathbb{Z}_p$. Therefore $\boldsymbol{M}\vec{\psi}$ denotes the vector of ℓ shares of ξ that corresponds to Π and every share in $\boldsymbol{M}\vec{\psi}$ belongs to the party $\rho(i)$.*
   The pair ($\boldsymbol{M}$,ρ) denotes the access policy structure $\mathbb{X}$.

LSSS can be summarized using two randomized algorithms:

Algorithm 1: Distribute

Input : $(\boldsymbol{M}, \rho, \xi)$, ξ is the secret value to be shared
Output: $\{\lambda_{\rho(i)} : i \in [\ell]\}$

1 Choose randomly $y_2, y_3, ...y_k \leftarrow_R \mathbb{Z}_p$
2 Set $\vec{\psi} = (\xi, y_2, y_3, ...y_k) \in \mathbb{Z}_p^k$
3 In every row $i \in [\ell]$, calculate $\lambda_{\rho(i)} = \vec{v}\boldsymbol{M}_i\}_{i\in[\ell]}$, where $\mathbf{M}_i \in \mathbb{Z}_p^k$ is i^{th} row of the matrix $\mathbf{M}$. The share $\lambda_{\rho(i)}$ is given to $\rho(i)$
4 Return: $\{\lambda_{\rho(i)} : i \in [\ell]\}$

Algorithm 2: Reconstruct

Input : (M, ρ, χ), where $\vec{\chi} = \chi_1, \chi_2, ...\chi_m \in \{0, 1\}^m$
Output: $\vec{\Phi} \in \mathbb{Z}_p^k$ or $\perp$ where $\perp$ indicates not valid input

1 Calculate a vector $\vec{\Phi} = (\Phi_1, \Phi_2, ..., \Phi_\ell) \in \mathbb{Z}^\ell$ that satisfy this two characteristics: $\vec{\Phi} \cdot \boldsymbol{M} = (1, 0, . . . , 0)$ and $[\chi_{\rho(i)} = 0 \implies \Phi_i = 0, \forall\ i]$.
2 Return $\vec{\Phi}$.

3.2.4. Bilinear maps

Let map e: $\mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}_T$ denote bilinear pairing where $\mathcal{G}$ and $\mathcal{G}_\tau$ represent two multiplicative cyclic groups that belongs to prime order p. Let g be generator of $\mathcal{G}$. Bilinear map e has the following properties:

- *Bilinearity*: $\forall g \in \mathcal{G}$ and $a, b \in \mathbb{Z}_p^*$ we have $e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a)$.
- *Non-degeneracy*: $e(g, g) \neq 1$.
- *Computability*: There is an efficient algorithm to compute $e(g, g)$ for all $g \in \mathcal{G}$.

3.3. Claim predicates

Claim predicates, are policy formulas applied to the attributes /credential sets. They model access control policies.

Definition 2. *We utilize $\mathcal{U}$ to denote universe of attributes. A predicate (over $\mathcal{U}$) represents a monotone boolean function where inputs are related to the attributes universe $\mathcal{U}$. An attribute set $S \subset \mathcal{U}$ satisfies predicate Υ (or Υ accepts R) if $\Upsilon(\mathcal{U}) = 1$. This means an input is set to be true (i.e set to 1) if its equivalent attribute belong to attribute set S. And the input is set to be false (i.e set to 0) if its equivalent attribute does not belong to attribute set S. We denote using $\Upsilon(S) = 0$ if S does not satisfy Υ.*

Since we have the predicate Υ as monotone, then it means $\Upsilon(R) = true$ and therefore $\Upsilon(W) = true$ for each attribute set $W \supset R$.

Assuming Υ is claim predicate and R_Υ is an attribute set employed in Υ then a labeled matrix $\Theta = (\mathbf{M}_{\ell\times k}, \rho)$ corresponds to MSP for Υ, here $\mathbf{M}$ is $\ell \times k$ matrix while $\rho : [\ell] \to R_\Upsilon$ labels the rows of $\mathbf{M}$ using the attributes from R_Υ. Attributes are taken as variables of MSP.

We represent $A \subset \mathcal{U}$ as set of attributes. We also define $x_1 = \{i \in [\ell] : [\rho(i) = v] \wedge [v \in A]\}$ and $x_0 = \{i \in [\ell] : [\rho(i) = v] \wedge [v \notin A]\}$. Then $x_1 = \{i \in [\ell] : \rho(i) \in A\}$ and $x_0 = \{i \in [\ell] : \rho(i) \notin A\}$. In this case $x_1 \cup x_0 = [\ell]$.

A predicate Υ (consisting $\Theta = (M_{\ell\times k}, \rho)$) accepts an input A that denotes attribute set by following this basis,

$$\Upsilon(A) = 1 \iff [\exists (z_1, ..., z_\ell) \in \mathbb{Z}_p^\ell \text{ such that } \textstyle\sum_{i\in[\ell]} z_i.\vec{M}^i = (1, 0, , , 0) \text{ and } z_i = 0\ \forall i, \text{ where } \rho(i) \notin A].$$

The following result is essential while presenting the security proof of the proposed scheme.

Lemma 1. *Let Υ, $\mathcal{U}$ denote predicate and attribute set respectively. If $\Upsilon(A) = false$ then there exist $\vec{\omega} = (\omega_1, \omega_2, ...\omega_k) \in \mathbb{Z}_p^k$ with $\omega_1 = -1$ such that $\vec{\omega}.\vec{M}^{(i)} = 0\ \forall i$ where $\rho(i) \in A$.*

3.4. Hard Problems

3.4.1. Diffie-Helman Exponent problem

Consider bilinear group $\Omega = (p, e, \mathcal{G}, \mathcal{G}_\tau)$ with distribution tuple $\mathcal{T} = (\Omega, g, g^y, g^{y^2}, ..., g^{y^q}, ..g^{y^{q+2}}, .., g^{y^{2q}})$ such that $g \in \mathcal{G}^{2q}$ and $y \leftarrow \{2, 3, .., p-1\}$.
Given $\mathcal{T}_{y,g} = (g, g^y, g^{y^2}, ..., g^{y^q}, ..g^{y^{q+2}}, .., g^{y^{2q}}) \overset{R}{\leftarrow} \mathcal{T}$, it computes $g^{y^{q+1}}$. The adversary advantage $\mathcal{A}$ in solving q-Diffie-Helman Exponent (abbreviated as $q - DHE$) problem is defined as $Adv_{\mathcal{A}}^{q-DHE} = Pr[g^{y^{q+1}} \leftarrow \mathcal{A}(\mathcal{T}, \mathcal{T}_{y,g})] \leq \epsilon$

Definition 3. *We establish that the q-DHE problem in $(\mathcal{G}, \mathcal{G}_\tau)$ is (τ, ϵ)-hard if the advantage $Adv_{\mathcal{A}}^{q-DHE} \leq \epsilon$ for all PPT adversary $\mathcal{A}$ running in time at most τ.*

3.4.2. q-1 Problem

Consider bilinear group $\Omega = (p, e, \mathcal{G}, \mathcal{G}_\tau)$. Select $g \in \mathcal{G}$ and $2q + 2$ random exponents $\varepsilon, y, k_1, ..., k_q \leftarrow \{2, 3, .., p-1\}$.
Given $\mathcal{T}_{\varepsilon,y,g} = (\Omega, g, g^\theta, \{g^{z^i}, g^{k_j, g^{\theta k_j}}, g^{z^i k_j}, g^{z^i/k_j^2}\}_{(i,j)\in[q,q]}, \{g^{z^i/k_j}\}_{(i,j)\in[2q,q],i\neq q+1}, \{g^{z^i k_j/k_{j'}^2}\}_{(i,j,j')\in[2q,q,q],j\neq j'}, \{g^{\theta z^i k_j/k'_j}, g^{\theta z^i k_j/k_{j'}^2}\}_{(i,j,j')\in[q,q,q],j\neq j'})$, it then computes $X_\upsilon \overset{R}{\leftarrow} \mathcal{G}_\tau$ where $\upsilon = \{0, 1\}$. The $q-1$ problem confirms whether $X = e(g, g)^{\theta.y^{q+1}}$ when $\upsilon = 0$ or X is a random number of $\mathcal{G}_\tau$ otherwise.

Definition 4. *We establish that the q-1 problem in $(\mathcal{G}, \mathcal{G}_\tau)$ is (τ, ϵ)-hard if the advantage $Adv_{\mathcal{A}}^{q-1} = Pr[\upsilon' = \upsilon] - 1/2$ for all PPT adversary $\mathcal{A}$ running in time at most τ.*

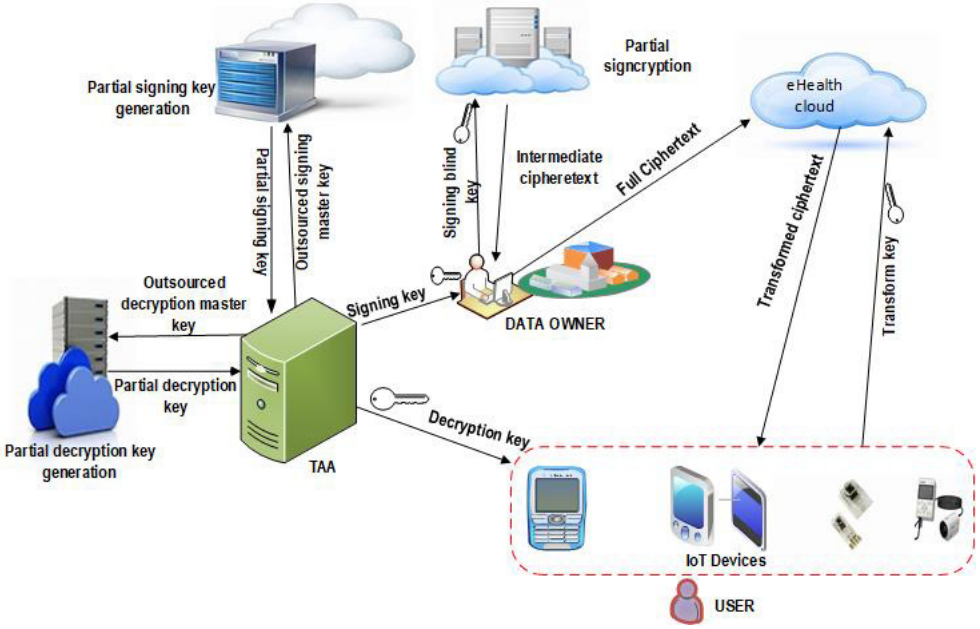


Figure 4. System model.

3.5. System model of VFOABSC

Figure 4 illustrates an architecture based on our proposed VFOABSC scheme. In our proposed scheme expensive ABSC computations are delegated to the cloud server so as to minimize computation overhead encountered by lightweight IoT devices with inefficient computational capability. Trusted Attribute Authority (TAA) sends the outsourcing master key (both signing and decryption) to respective servers to generate partial key (signing and decryption) then TAA locally completes the process by generating full signing and decryption keys. To reduce heavy computations while checking the validity of outsourced generated key (both signing and decryption), TAA requests trusted third party to verify. When data owner receives signing key from TAA, it generates two keys from it. One of the key is sent to the cloud server to be used for executing partial signcryption and its local key is utilized in generating the remainder portion of the ciphertext. Data owner then uploads full ciphertext to the cloud that is considered to be secure. To decrypt the data, the IoT device user sends decryption blinding key to the

server to execute expensive computations if its attributes satisfies access structure policy. If that is the case, it uses the corresponding retrieving key to perform full decryption.

Our proposed system comprise the following participants: trusted attribute authority (TAA), cloud server, data owner and data user. The functionalities of these participants are:

1. *Trusted Attribute Authority (TAA)*: It generates the outsourcing master key (both signing and decryption) then sends to the respective servers for them to generate partial key (signing and decryption). Upon receiving partial keys from the servers, TAA completes the process by generating full signing and decryption keys. It is the only entity entirely trusted by all other participants in the system.

2. *Cloud server:* There are seven cloud servers in our system. The first is for generating partial decryption key, the second for generating partial signing key, the third for verifying outsourced signing partial generated key, the fourth for verifying outsourced decryption partial generated key, fifth performs outsourced signcryption to generate partial ciphertext, sixth performs outsourced designcryption to generate partial plaintext and the seventh stores data for the owner at the same time provides data access services to the users.

3. *Data owner:* Generates two keys from the signing key. Sends one key to the cloud server for generating partial ciphertext while he remains with another key. Once it receives partial ciphertext from the cloud, it computes full ciphertext using his own key. Finally it outsources full ciphertext to the cloud.

4. *Data user:* The user with lightweight IoT device such as medical sensors sends a decryption blinding key to the proxy to transform ciphertext into simple one. When it receives partially decrypted ciphertext it uses retrieving key to complete the decryption. It can only fully decrypt the ciphertext when his attributes satisfies access policy in the ciphertext.

Sixteen algorithms are defined in our system as follows:
Setup $(1^{\kappa}, U) \rightarrow (PP, MSK)$
This algorithm takes the input values κ and U as security parameter and the universe of attributes respectively. It returns as an output the public parameters PP and master key MSK.
SigKeyGen.init$(PP, MSK) \rightarrow (MSK_{OKGSP_s}, MSK_{TAA_s})$.
TAA executes this pre-processing algorithm. It takes the input PP and MSK as public parameters and master key respectively and returns as output outsourcing signing master key MSK_{OKGSP_s} and local signing master key MSK_{TAA_s}
SigKeyGen.out $(PP, MSK_{OKGSP_s}, A_s) \rightarrow (ISK_s)$.
$OKGSP_s$ executes this algorithm. It takes as input public parameters PP, outsourcing signing master key MSK_{OKGSP_s} and attribute set A_s. It gives as an output intermediate private signing key ISK_s.
SigKeyGenout.ver $(PP, MSK_{OKGSP_s}, ISK_s \rightarrow \text{b} \in \{0, 1\})$
Outsourced verification service provider $(OVSP_s)$ executes this algorithm. It takes the input PP, MSK_{KGSP_s} and ISK_s as public parameters, outsourcing signing master key and intermediate private signing key respectively. It yields $b = 1$ if the equations are true, otherwise it yields $b = 0$.
SigKeyGen.local $(PP, MSK_{TAA_s}, A_s, ISK_s) \rightarrow (SK_{A_s})$
TAA executes this algorithm. It takes as an input public parameters PP, signing local master key MSK_{TAA_s}, attribute set A_s and intermediate signing key ISK_s. It yields as an output complete private

signing key SK_{A_s}.
SigKey.blind $(SK_{A_s}) \rightarrow (TSK_{A_s}, RSK_{A_s})$
The input to this algorithm is signing private key SK_{A_s}. It outputs transformation private signing key TSK_{A_s} with its respective retrieving key RSK_{A_s}.
DecKeyGen.init$(PP, MSK) \rightarrow (MSK_{OKGSP_d}, MSK_{TAA_d})$. TAA executes this pre-processing algorithm. It takes public parameters PP and master key MSK as input. It returns as an output outsourcing decryption master key MSK_{OKGSP_d} and local decryption master key MSK_{TAA_d}.
DecKeyGen.out $(PP, MSK_{OKGSP_d}, A_d) \rightarrow (ISK_d)$
$OKGSP_d$ executes this algorithm. It takes as input public parameters PP, master key MSK_{OKGSP_d} for outsourcing decryption key generation and attribute set A_d. It yields intermediate private decryption key ISK_d as an output.
DecKeyGenout.ver$(PP, MSK_{OKGSP_d}, ISK_d \rightarrow \mu \in \{0, 1\})$ Outsourced verification service provider $(OVSP_d)$ executes this algorithm. It takes the input PP, MSK_{OKGSP_d} and ISK_d as public parameters, outsourcing decryption master key and intermediate decryption private key respectively . It outputs $\mu = 1$ if the equations holds, otherwise it outputs $\mu = 0$.
DecKeyGen.local$(PP, MSK_{TAA_d}, A_d, ISK_d) \rightarrow (SK_{A_d})$:
TAA executes this algorithm. It takes as an input public parameters PP, local master key MSK_{TAA_d}, attribute set A_d and intermediate decryption key ISK_d. It outputs complete decryption private key SK_{A_d}.
DecKey.blind $(SK_{A_d}) \rightarrow (TSK_{A_d}, RSK_{A_d})$:
The user takes decryption private key SK_{A_d} as an input to this algorithm and yields transformation private key TSK_{A_d} and its associated retrieval secret key RSK_{A_d} as an output.
Signcrypt.out $(PP, TSK_{A_s}, \Upsilon_s, \Upsilon_e) \rightarrow ICT$:
$OSSP$ executes this algorithm. It takes as input public parameters PP, signing transformation key TSK_{A_s}, signing attributes A_s, signing predicate Υ_s and encryption predicate Υ_e. It produces as an output intermediate ciphertext ICT.
Signcrypt.local $(PP, M, RSK_{A_s}, ICT) \rightarrow SCT_{\Upsilon_e}$:
The algorithm takes public parameters PP, message M, retrieving key RSK_{A_s} and intermediate ciphertext ICT as input. It produces complete ciphertext SCT_{Υ_e} as an output.
Verification $(PP, SCT_{\Upsilon_e}) \rightarrow$*valid* or $\perp$
The algorithm takes as input PP, original ciphertext SCT_{Υ_e} to output either valid or invalid symbol $\perp$
Designcrypt.out $(PP, TSK_{A_d}, A_d, SCT_{\Upsilon_e}) \rightarrow (TCT_{A_d})$:
This algorithm is executed by $ODSP$. It takes public parameters PP, the key for transformation TSK_{A_d}, an attribute set A_d, ciphertext SCT_{Υ_e} as input. It returns transformed ciphertext TCT_{A_d} as output.
Designcrypt.local $(PP, SCT_{\Upsilon_e}, A_d, SK_{A_d}, \Upsilon_e) = M$ or $\perp$:
This algorithm takes as input PP, original ciphertext SCT_{Υ_e}, an attribute set A_d, decryption private key SK_{A_d} and encryption predicate Υ_e. It returns as an output message M or invalid symbol $\perp$.

3.6. Security model of VFOABSC

3.6.1. Confidentiality of the message:

Identical to [25], we employ a security game that describes the confidentiality of the message. We consider a challenger C and an adversary $\mathcal{E}$.
Setup: C executes $Setup$ algorithm to obtain public parameter PP and master key MSK. It sends PP

to $\mathcal{E}$.
Query Phase 1: Challenger C creates an empty table $\mathbf{T}_1$, a integer counter $\varsigma = 0$ and an empty set R. $\mathcal{E}$ then it adaptively issues the following queries:

1. **DecKey Queries:** C on obtaining an attribute set A_d sets a counter $\varsigma = \varsigma + 1$, then executes $SK_{A_d} \leftarrow$**DecKeyGen.local**$(PP, MSK_{TAA_d}, A_d,$ **DecKeyGen.out** $(PP, MSK_{OKGSP_d}, A_d))$ where $\Upsilon_e^*(A_d) = 0$. Then sets $\mathbf{T}_1 = \mathbf{T}_1 \cup \{A_d\}$. It sends SK_{A_d} to $\mathcal{E}$.

2. **DecKey.blind Query:** On obtaining an input attribute set A_d it executes $SK_{A_d} \leftarrow$**DecKeyGen.local**$(PP, MSK_{TAA_d}, A_d,$ **DecKeyGen.out**$(PP, MSK_{OKGSP_d}, A_d))$, $(TSK_{A_d}, RSK_{A_d}) \leftarrow$**DecKey.blind** (SK_{A_d}) . It then stores the entry $(\varsigma, A_d, SK_d, TSK_{A_d}, RSK_{A_d})$ in table $\mathbf{T}_1$ then supply the adversary $\mathcal{E}$ with TSK_{A_d}.

Challenge Phase: $\mathcal{E}$ submits two messages M_0 and M_1 of equal length alongside decryption predicate $\Upsilon_e^*(A_d)$ with the condition that no attribute set in $\mathbf{T}_1$ should satisfy the decryption predicate $\Upsilon_e^*(A_d)$. Next, C picks a bit $\gamma \in \{0, 1\}$ randomly. C then executes signcryption on M_γ using $\Upsilon_e^*(A_d) = 1$.
Query Phase 2: Upon receiving $SCT_{\Upsilon_e}^*$, $\mathcal{E}$ adaptively continues to issue queries in the same way as in **Query Phase 1** with the condition that it cannot **Designcrypt Query**, for an attribute set A_d and A_s in such way that $\Upsilon_e^*(A_d) = 1$ and $\Upsilon_s^*(A_s) = 1$
Guess Phase: $\mathcal{E}$ outputs a guess bit $\gamma' \in \{0, 1\}$ and wins the game if $\gamma' = \gamma$, otherwise outputs $\perp$.

Definition 5. *VFOABSC scheme is CPA-secure if no PPT adversary that have non-negligible advantage wins the security game above.*

3.6.2. Unforgeability of ciphertext:

The following security game of the formal unforgeability definition involves challenger C and adversary $\mathcal{A}$.
Setup: Challenger C executes $Setup$ algorithm to obtain public parameter PP and master key MSK. It sends PP to $\mathcal{A}$.
Query Phase 1: Challenger C in addition to creating an empty table $\mathbf{T}_2$ also creates an integer counter β and empty set R. $\mathcal{A}$ then issues adaptively queries as follows:

1. **DecryptKey Queries** and **DecKey.blind Query** are similar to the described CPA-secure game.

2. **SigncryptKey Queries:** C initially sets $\beta = \beta + 1$, and for an attribute set A_s, it executes $(SK_{A_s}) \leftarrow$**SigKeyGen.local** $(PP, MSK_{TAA_s}, A_s,$**SigKeyGen.out** $(PP, MSK_{OKGSP_s}, A_s))$ where $\Upsilon_e^*(A_s) = 0$ then sets $\mathbf{T}_2 = \mathbf{T}_2 \cup \{A_d\} \cup \{A_s\}$. Finally it sends the signing key SK_{A_s} to $\mathcal{A}$.

3. **SigKey.blind:** On obtaining an input attribute set A_s it executes $(SK_{A_s}) \leftarrow$**SigKeyGen.local** $(PP, MSK_{AA_s}, A_s,$**SigKeyGen.out** $(PP, MSK_{OKGSP_s}, A_s))$, $(TSK_{A_s}, RSK_{A_s}) \leftarrow$ **SigKey.blind**(SK_{A_s}). It stores the entry $(\beta, \diamond, SK_s, TSK_{A_s}, RSK_{A_s})$ in table $\mathbf{T}_2$ then supply the adversary with TSK_{A_s}.

Forgery: $\mathcal{A}$ returns forged ciphertext $SCT^*_{\Upsilon^*_e}$ for the selective signing predicate Υ^*_s.
Output: $\mathcal{A}$ wins the game if it outputs valid ciphertext $SCTY^*_{\Upsilon^*_e}$ where $\mathcal{A}$ has never issued query on signcrypt record $(M^*, \Upsilon^*_s, \Upsilon^*_e)$ and false otherwise.

Definition 6. *The VFOABSC scheme is said to be existential unforgeable secure if no PPT adversary $\mathcal{A}$ that have non-negligible advantage wins the security game above.*

3.6.3. Privacy of Signcryptor:

The formal definition of signcryptor privacy of VFOABSC scheme is based on security game between a challenger C and adversary $\mathcal{A}$.
Setup: C executes $Setup$ algorithm to obtain public parameter PP and master key MSK. It sends PP and MSK to $\mathcal{A}$ where MSK is combination of MSK_{OKGSP_s} and MSK_{TAA_s}.
Challenge: $\mathcal{A}$ presents a message $M \in \mathcal{M}$, signing predicate $\Upsilon_{s(A^1_s)} = \Upsilon_{s(A^2_s)} = 1$, and also an encryption predicate Υ_e. Then, C selects $b \leftarrow \{0,1\}$ to obtain $SK_{A^b_s} \leftarrow$**SigKeyGen.local**$(PP, MSK_{TAA_s}, A_s,$**SigKeyGen.out** $(PP, MSK_{OKGSP_s}, A_s))$ then outputs a challenge ciphertext $SCT_{\Upsilon_e} \leftarrow$ **Signcrypt.local**$(PP, M, \Upsilon_e,$**Signcrypt.out**$(PP, SK_{A^b_s}, \Upsilon_s))$ to $\mathcal{A}$.
Guess: $\mathcal{A}$ outputs guess bit $b' \in \{0,1\}$
Output: C outputs true if $b' = b$, and false otherwise.
It should be noted that adversary $\mathcal{A}$ can itself generate signing keys and ciphertexts in view of the fact that he has knowledge about the master key of the system.

Definition 7. *The VFOABSC scheme is considered perfectly private if no PPT adversary $\mathcal{A}$ that have non-negligible advantage wins the security game above.*

4. The concrete construction

We present in this section the main construction of our proposed Verifiable Fully Outsourced Attribute-Based Signcryption System (VFOABSC) which is based on attribute-based signcryption due to [25] with the extension of outsourced key generation, outsourced key verification, outsourced signcryption and outsourced designcryption computation simultaneously to improve the efficiency of lightweight IoT devices. We first modified the offline/online signcryption phases in Rao's scheme [25] to outsourcing and owner signcryption phases respectively. Then to realize the above stated desirable features, we utilized the cryptographic scheme due to Wang *et al.* [34]. Our scheme supports both boolean function predicates and large attribute universe $U = \{0,1\}^*$. Moreover, it allows public verifiability mechanism for ciphertext. In the construction of our scheme the signing predicate and its counterpart encryption predicate are both represented using Monotone Span Programs (MSPs). We denote signing predicate and encryption predicate using $\Upsilon_s = (M_s, \rho_s)$ and $\Upsilon_e = (M_e, \rho_e)$ respectively. M_s(resp. M_e) represents $\ell_s \times k_s$ (resp. $\ell_e \times k_e$) matrix with $\rho_s : [\ell_s] \to U$ (resp. $[\ell_e] \to U$) denoting row labeling function. The i^{th} row of matrix M_s (resp.M_e) is represented using the notation $\vec{M}^{(i)}_s$ (resp. $\vec{M}^{(i)}_e$). In our scheme, it is assumed that the row labeling function ρ_s to be injective in the signing predicate $\Upsilon_s = (M_s, \rho_s)$. We utilize the bilinear group tuple $\Omega = (q, \mathcal{G}, \mathcal{G}_\tau, e)$. Our new scheme consist of the following sixteen algorithms.

Algorithm 3: Setup - It is executed by TAA

Input : $(1^{\kappa}, U)$, where κ and U are security parameter and the universe of attributes respectively.
Output: (PP, MSK)

1 Two cyclic groups $\mathcal{G}, \mathcal{G}_{\tau}$ of prime order p and generator g of $\mathcal{G}$ is chosen. A bilinear group description is denoted as $\Pi = (p, \mathcal{G}, \mathcal{G}_{\tau}, e)$ where $e : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}_{\tau}$ is a bilinear map. Let $\mathcal{M} = \{0,1\}^{l_m}$
2 The algorithm selects four hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$,$H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$,$H_3 : \{0,1\}^* \rightarrow \{0,1\}^{l_m}$,$H_s : \{0,1\}^* \rightarrow \mathcal{G}$, (Here H_1 and H_2 are two independent cryptographic hash functions)
3 It also chooses randomly $g, h_e, u_e, w_e, u_s, w_s, v, \varrho_1, \varrho_2, \varrho_3 \in_R \mathcal{G}$.
4 It samples $\alpha \in \mathbb{Z}_p^*$ then sets $\Sigma = e(g,g)^{\alpha}$.
5 The system public parameters $PP = (\Pi, \Sigma, g, h_e, u_e, w_e, u_s, w_s, v, \varrho_1, \varrho_2, \varrho_3, \mathcal{M}, U, H_1, H_2, H_3, H_s)$
6 The master key MSK output is $MSK = g^{\alpha}$.
7 Returns (PP, MSK).

Algorithm 4: SigKeyGen.init - It is pre-processing algorithm executed by TAA

Input : (PP, MSK), where PP and MSK denotes public parameters and master key respectively
Output: $(MSK_{OKGSP_s}, MSK_{TAA_s})$

1 Randomly chooses $\alpha_0 \in_R \mathbb{Z}_p$.
2 Computes $MSK_{OKGSP_s} = g^{\alpha_0}$ and $MSK_{TAA_s} = g^{\alpha - \alpha_0}$. Where MSK_{OKGSP_s} is outsourcing master key for generating partial signing key while MSK_{TAA_s} is local master key for generating the remaining portion of the signing key.
3 Returns $(MSK_{OKGSP_s}, MSK_{TAA_s})$.

Algorithm 5: SigKeyGen.out - It is executed by $OKGSP_s$

Input : (PP, MSK_{OKGSP_s}, A_s)
Output: (ISK_s)

1 It takes as input public parameters PP, master key MSK_{OKGSP_s} for outsourcing key generation and attribute set A_s
2 Selects randomly $\delta' \in \mathbb{Z}_p$ then computes $K_{so} = g^{\alpha_0} v^{\delta'}$,$K'_{so} = g^{\delta'}$, $K_{so,a} = H_s(a)^{\delta'}, \forall a \in A_s$.
3 It finally outputs intermediate signing key $ISK_s = (K_{so}, K'_{so}, \{K_{so,a}\} \forall a \in A_s)$
4 Returns (ISK_s).

Algorithm 6: SigKeyGenout.ver - It is executed by $OVSP_s$

Input : $(PP, MSK_{OKGSP_s}, ISK_s)$

Output: $(b \in \{0,1\})$

1 Whenever TAA request for verification of outsourced generated signing key, it executes the following equations:

$$\begin{aligned} e(K_{so}, g) &\stackrel{?}{=} e(g, MSK_{OKGSP_s}).e(K'_{so}, \nu) \\ e(K'_{so}, H_s(a)) &\stackrel{?}{=} e(g, K_{so,a}) \forall a \in A_s \end{aligned} \tag{4.1}$$

It produces $b = 1$ as output if all the above equations are valid, otherwise outputs $b = 0$ to indicate invalid results.

2 Returns (b).

Algorithm 7: SignKeyGen.local - It is executed by TAA

Input : $(PP, MSK_{TAA_s}, A_s, ISK_s)$

Output: (SK_{A_s})

1 It takes as an input public parameters PP, local master key MSK_{TAA_s}, attribute set A_s and intermediate signing key ISK_s.

2 It then computes $K_s = g^{\alpha-\alpha_0}.K_{so}$, $K'_s = K'_{so}$, $\{K_{s,a} = K_{so,a}\} \forall a \in A_s$.

3 It outputs $SK_{A_s} = (PP, A_s, K_s, K'_s, \{K_{s,a}\} \forall a \in A_s)$

4 Returns (SK_{A_s})

Algorithm 8: SigKey.blind - It is executed by data owner

Input : (SK_{A_s})

Output: (TSK_{A_s}, RSK_{A_s})

1 The input to this algorithm is signing private key SK_{A_s}.

2 It randomly picks number $\phi \in_R \mathbb{Z}_p^*$ after which it computes $TK_{s1} := K_s^{1/\phi}$, $TK_{s2} := K_s'^{1/\phi}$, $TK_{s,j} := K_{s,a}^{1/\phi}$, $\forall j \in A_s$.

3 It returns transformation signing key for A_s as $TSK_{A_s} := [A_s, TK_{s1}, TK_{s2}, \{TK_{s,j}\}_{j \in A_s}$ and the retrieving private key as $RSK_{A_s} = \phi$.

4 Transformation signing key TSK_{A_s} is sent to CSP to be utilized for generating partial signcrypted ciphertext.

5 Returns (TSK_{A_s}, RSK_{A_s})

Algorithm 9: DecKeyGen.init - It is pre-processing algorithm executed by TAA

Input : (A_d, MSK)

Output: $(MSK_{OKGSP_d}, MSK_{TAA_d})$

1 Randomly chooses $\alpha_1 \in_R \mathbb{Z}_p$.

2 Computes $MSK_{OKGSP_d} = g^{\alpha_1}$ and $MSK_{TAA_d} = g^{\alpha-\alpha_1}$. Where MSK_{OKGSP_d} is outsourcing master key for generating partial decryption key while MSK_{TAA_d} is local master key for generating the remaining portion of the decryption key.

3 Returns $(MSK_{OKGSP_d}, MSK_{TAA_d})$

Algorithm 10: DecKeyGen.out - It is executed by $OKGSP$

Input : (PP, MSK_{OKGSP_d}, A_d)

Output: (ISK_d)

1 It takes as input public parameters PP, master key MSK_{OKGSP_d} for outsourcing key generation and attribute set A_d.

2 Selects randomly $\delta, \delta_i \in \mathbb{Z}_p$ then computes $K_{d0} = g^{\alpha_1}\nu^{\delta}$, $K_{d1} = g^{\delta}$, $K_{d0,i} = (u_e^i h_e)^{\delta_i} w_e^{-\delta}$, $K_{d1,i} = g^{\delta_i}$.

3 It finally outputs intermediate decryption key $ISK_d = (PP, A_s, K_{d0}, K_{d1}, \{K_{d0,i}, K_{d1,i}\}_{i\in A_d})$.

4 Returns (ISK_d)

Algorithm 11: DecKeyGenout.ver - It is executed by $OVSP_d$

Input : $(PP, MSK_{OKGSP_d}, ISK_d)$

Output: $(\mu \in \{0, 1\})$

1 Whenever TAA request for verification of outsourced generated key, the algorithm performs the following equations:

$$\begin{aligned} e(K_{d0}, g) &\stackrel{?}{=} e(g, MSK_{OKGSP_d}).e(K_{d1}, \nu) \\ e(g, K_{d0,i}) &\stackrel{?}{=} e(K_{d1,i}, (u_e^i h_e)).e(w_e, g^{-\delta}) \forall i \in A_d \end{aligned} \tag{4.2}$$

It gives as output $\mu = 1$ if all the above equations are true, otherwise outputs $\mu = 0$ to indicate incorrect results.

2 Returns (μ)

Algorithm 12: DecKeyGen.local - It is executed by TAA

Input : $(PP, MSK_{TAA_d}, A_d, ISK_d)$

Output: (SK_{A_d})

1 It takes as an input public parameters PP, local master key MSK_{TAA_d}, attribute set A_d and intermediate decryption key ISK_d.

2 It then calculates $K_d = g^{\alpha-\alpha_1}.K_{d0}$, $K_{d1A} = K_{d1}$, $\{K_{d0A,i} = K_{d0,i}, K_{d1A,i} = K_{d1,i}\}_{i\in A_d}$.

3 Finally it returns as an output private decryption key $SK_{A_d} = (A_d, K_d, K_{d1A}, K_{d0A,i}, K_{d1A,i})$

4 Returns (SK_{A_d})

Algorithm 13: DecKey.blind - It is executed by user

Input : (SK_{A_d})

Output: (TSK_{A_d}, RSK_{A_d})

1 The user takes decryption private key SK_{A_d} as an input.

2 It randomly selects number $z \in \mathbb{Z}_p^*$ after it computes the following $TK_{d0} := K_d^{1/z}$, $TK_{d1} := K_{d1A}^{1/z}$, $TK_{d0,i} = K_{d0A,i}^{1/z}$, $TK_{d1,i} = K_{d1A,i}^{1/z}$

3 It returns the transformation decryption private key as $TSK_{A_d} = (TK_{d0}, TK_{d1}, \{TK_{d0,i}, TK_{d1,i}\}$ $\forall i \in A_d)$. and its respective retrieving private key as $RSK_{A_d} = z$.

4 The transformation decryption private key TSK_{A_d} is sent to the cloud for generating partially decrypted ciphertext.

5 Returns (TSK_{A_d}, RSK_{A_d})

Algorithm 14: Signcrypt.out - It is executed by $OSSP$

Input : $(PP, TSK_{A_s}, \Upsilon_s, \Upsilon_e)$

Output: (ICT)

1 It takes as input public parameters PP, signing transformation key TSK_{A_s}, signing attributes A_s signing predicate Υ_s and encryption predicate Υ_e with property that $\Upsilon_s(A_s) = 1$.

2 The algorithm then computes a vector $\vec{c} := (c_1, c_2, .., c_{\ell_s}) \in \mathbb{Z}_p^{\ell_s}$ where $\vec{c}.M_s = \vec{1}_{n_s}$ i.e $\sum_{i\in[\ell_s]} c_i.\vec{M}_s^{(i)} = \vec{1}_{n_s}$ and $c_i = 0$ for all i where $\rho_s \notin A_s$. This is possible because $\Upsilon_s(A_s) = 1$.

3 It randomly selects a vector $(x_1, x_2, .., x_{\ell_s}) \in_R \mathbb{Z}_p^*$ such that $\sum_{i\in[\ell_s]} x_i.\vec{M}_s^{(i)} = \vec{0}_{n_s}$.

4 It also chooses $r' \in_R \mathbb{Z}_p^*$ and uses it to re-randomize transformation signing private key TSK_{A_s} as follows $TSK_{A_s}^R = (A_s, TK_{s1}^R = TK_{s1}.v^{r'/\phi}, TK_{s2}^R = TK_{s2}.g^{r'/\phi}, \{TK_{s,j}^R = TK_{s,j}.H_s(j)^{r'/\phi}\}_{j\in A_s})$ $= (A_s, TK_{s1}^R = g^{\alpha/\phi}.v^{\psi/\phi}, TK_{s2}^R = g^{\psi/\phi}, \{TK_{s,j}^R = H_s(j)^{\psi/\phi}\}_{j\in A_s})$ where $\psi = \delta' + r'$.

5 In addition, it picks randomly $\xi', \zeta \in_R \mathbb{Z}_p^*$.

6 It also chooses, $\lambda_i', \pi_i, t_i \in_R \mathbb{Z}_p$ then sets $\vec{C}_i$=$(v^{\lambda_i'} w_e^{t_i}, (u_e^{\pi_i} h_e)^{t_i}, g^{t_i})_{\forall i\in[\ell_e]}$ $\sigma_0' = (TK_{s1}^R)(\Pi_{i\in[\ell_s]}(TK_{s,j}^R)^{c_i} H_s(\rho_s(i))^{\zeta x_i/\phi}$ $\sigma_i' = (TK_{s2}^R)^{c_i} g^{\zeta x_i/\phi}, \forall i \in [\ell_s]$

7 It returns as an output a partially signcrypted ciphertext, ICT=$[\xi', \{\vec{C}_i, \lambda_i', \pi_i, t_i\}_{i\in[\ell_e]}, \sigma_0', \{\sigma_i'\}_{i\in[\ell_s]}]$.

8 The partial ciphertext ICT is then sent to the data owner to complete signcryption by generating the remaining ciphertext.

9 Returns (ICT)

Algorithm 15: Signcrypt.local - It is executed by the data owner

Input : (PP, M, RSK_{A_s}, ICT)

Output: SCT_{Υ_e}

1 The owner picks $\xi, \vartheta, \eta \in_R \mathbb{Z}_p^*$, then calculates $key = \Sigma^{\xi}$.

2 It also computes shares $\lambda_i = M_e^i \vec{y}$ where $\vec{y} = (\xi, y_2, ..y_{n_e}) \in \mathbb{Z}_p^{n_e}$ and M_e^i is the i^{th} row of the matrix M_e.

3 The algorithm then calculates the following terms: $C_0 := H_3(key, \Upsilon_e, \Upsilon_s) \oplus M$, $\vec{C_1}=(g^{\xi}, (\varrho_1^{\vartheta}\varrho_2^{\eta}\varrho_3)^{\xi})$, $\{C'_{i1} = (\lambda_i - \lambda'_i), C'_{i2} = t_i(\rho_e(i) - \pi_i)\}_{i\in\ell_e}$, $\varsigma = H_1(C_0, \Upsilon_e, \Upsilon_s)$, $C_2 = (u_s^{\varsigma} w_s)^{\xi}$, $\sigma_0 = \sigma_0'^{RSK_{A_s}}.C_2$, $\{\sigma_i = \sigma_i'^{RSK_{A_s}}\}_{i\in[\ell_s]}$.

4 The complete ciphertext is $SCT_{\Upsilon_e} = (C_0, \vec{C_1}, \{C'_{i1}, C'_{i2}\}_{i\in[\ell_e]}, \sigma_0, \{\sigma_i\}_{i\in[\ell_s]})$

5 Returns (SCT_{Υ_e})

Algorithm 16: Verification - It is executed by any third party

Input : (PP, SCT_{Υ_e})

Output: *valid* or $\perp$

1 It takes as input public parameters PP and ciphertext SCT_{Υ_e}.

2 The algorithm computes $\vartheta = H_2(C_0, C_{11}, \{C'_{i1}, C'_{i2}\}_{i\in\ell_e}, \sigma_0, \{\sigma_i\}_{i\in[\ell_s]}, \Upsilon_e, \Upsilon_s)$ and verifies by checking whether

$$e(g, C_{12}) \stackrel{?}{=} (C_{11}, \varrho_1^{\vartheta}\varrho_2^{\eta}\varrho_3) \tag{4.3}$$

3 If the equation is false, $\perp$ is returned, otherwise execute the following; it randomly selects $f, \zeta'_1, .., \zeta'_{k_s} \in_R \mathbb{Z}_p^*$, then computes $(\varpi_1, ..., \varpi_{\ell_s}) = (f, \zeta'_1, .., \zeta'_{k_s}).M_s^T$ and verifies the following by checking whether its true;

$$e(\sigma_0, g^f) \stackrel{?}{=} \Delta^f.e(u_s^{\varsigma} w_s, C_{11}^f).\Pi_{i\in[\ell_s]} e(\nu^{\bar{\omega}_i} H_s(\rho_s(i))^f, \sigma_i) \tag{4.4}$$

here $\varsigma := H_1(C_0, \Upsilon_e, \Upsilon_s)$.

4 The signature is valid only if the equation holds. Main computation proceeds as follows, $\sigma_0 = \sigma_0'^{RSK_{A_s}}.C_2 = \sigma_0'^{\phi}.C_2 = (TK_s^R)^{\phi}(\Pi_{i\in[\ell_s]}(TK_{s,j}^R)^{\phi c_i} H_s(\rho_s(i))^{\zeta x_i.\phi/\phi}.\ (u_s^{\varsigma} w_s)^{\xi}$ $= g^{\alpha}\nu^{\psi}.(u_s^{\varsigma} w_s)^{\xi}.(\Pi_{i\in[\ell_s]} H_s(\rho_s(i))^{\psi.c_i+\zeta x_i} = g^{\alpha}\nu^{\psi}(u_s^{\varsigma} w_s)^{\xi}.(\Pi_{i\in[\ell_s]} H_s(\rho_s(i))^{\psi.c_i+\zeta x_i}$ where $\psi = \delta' + r'$ $\sum_{i\in[\ell_s]}(\psi.c_i + \zeta x_i).\varpi_i = \sum_{i\in[\ell_s]}(\psi.c_i + \zeta x_i).((f, \zeta'_2, .., \zeta'_{k_s})\ .\ \vec{M_s^{(i)}}) = (\psi f, \psi\zeta'_2, ..., \psi\zeta'_{k_s})\ .\ \sum_{i\in[\ell_s]} c_i.\vec{M_s^{(i)}} +$ $(\zeta f, \zeta\zeta'_2, .., \zeta\zeta'_2)\ .\ \sum_{i\in[\ell_s]} x_i.\vec{M_s^{(i)}} = (\psi f, \psi\zeta'_2, ..., \psi\zeta'_{k_s})\ .\ (1, 0, .., 0) + (\zeta f, \zeta\zeta'_2, .., \zeta\zeta'_{k_s})\ .\ (0, 0, .., 0) = \psi f$

5 Now, $e(\sigma_0, g^f)$= $\mathrm{e}(g^{\alpha}\nu^{\psi}(u_s^{\varsigma} w_s)^{\xi}.(\Pi_{i\in[\ell_s]} H_s(\rho_s(i))^{\psi.c_i+\zeta x_i}, g^f)$ $=e(g, g)^{\alpha f}.e(\nu, g)^{\psi f}.e(u_s^{\varsigma} w_s, g)^{\xi f}.e(\Pi_{i\in[\ell_s]} H_s(\rho_s(i))^{\psi.c_i+\zeta x_i}, g^f) = \Sigma^f.e(\nu, g)^{\sum_{i\in[\ell_s]}(\psi.c_i+\zeta x_i).\varpi_i}.e(u_s^{\varsigma} w_s, g^{\xi f}).$ $e(\Pi_{i\in[\ell_s]} H_s(\rho_s(i))^f, g^{\psi.c_i+\zeta x_i}) = \Sigma^f.(\prod_{i\in[\ell_s]} e(\nu_i^{\varpi}, g^{\psi.c_i+\zeta x_i}).e(u_s^{\varsigma} w_s, g^{\xi f}).\ e(\Pi_{i\in[\ell_s]} H_s(\rho_s(i))^f, g^{\psi.c_i+\zeta x_i})$ $=\Sigma^f.e(u_s^{\varsigma} w_s, C_{01}^f).(\Pi_{i\in[\ell_s]} e(\nu^{\varpi_i} H_s(\rho_s(i))^f, g^{\psi.c_i+\zeta x_i}) = \Sigma^f\ .\ e(u_s^{\varsigma} w_s, C_{01}^f)\ .\ (\Pi_{i\in[\ell_s]} e(\nu^{\varpi_i} H_s(\rho_s(i))^f, \sigma_i)$

6 Next it checks if $\Upsilon_e(A_d) = 0$. If that is the case then it returns $\perp$, else it proceeds to execute decryption.

7 Returns (*valid* or $\perp$)

Algorithm 17: Designcrypt.out - It is executed by $ODSP$

Input : $(PP, TSK_{A_d}, SCT_{\Upsilon_e})$

Output: TCT_{A_d}

1 It takes public parameters PP, the key for transformation TSK_{A_d} and ciphertext SCT_{Υ_e} as input. Whenever encryption predicate is satisfied by user's set of attributes, there exist a vector $\vec{c}' = (c'_1, ..., c'_{\ell_e}) \in \mathbb{Z}_p^{\ell_e}$ such that $\vec{c}'.M_e = \vec{1}_{n_e}$ where $\sum_{i\in[\ell_s]} c'.\vec{M}_e^{(i)} = \vec{1}_{n_e}$ and for all i where $\rho_e \notin A_d$, $\vec{c}' = 0$.

2 Next,the cloud server computes the transformed ciphertext TCT_{A_d} as:

$$\frac{e(C_{11}, TK_{d0}).e(\nu^{\sum_{i\in[\ell_e]} C'_{i1}c'_i}.\Pi_{i\in[\ell_e]}C_{i1}^{c'_i}, TK_{d1})^{-1}}{\Pi_{i\in[\ell_e]}(e(C_{i2}u_e^{C'_{i2}}, TK_{d0,i})^{-1}.e(C_{i3}, TK_{d1,i})^{c'_i}} = \frac{e(g,g)^{\alpha\xi/z}e(g^{\delta}/z, \nu^{\xi}).e(\nu^{\xi}, g^{-\delta/z}).\Pi_{i\in[\ell_e]}e(w_e^{c'_i.t_i}, g^{-\delta/z})}{\Pi_{i\in[\ell_e]}.e(g^{c'_i.t_i}, w_e^{-\delta/z})}$$

$= e(g,g)^{\alpha\xi/z} = \Sigma^{\xi/z}$.

3 The calculation process is carried as follows: $\sum_{i\in[\ell_e]} c'_i.(\xi, \zeta_2, .., \zeta_{k_e}).\vec{M}_e^{(i)} = (\xi, \zeta_2, .., \zeta_{k_e})$ $.\sum_{i\in[\ell_e]} c'_i.\vec{M}_e^{(i)} = (\xi, \zeta_2, .., \zeta_{k_e}).(1, 0, .., 0) = \xi$

4 The transformed ciphertext TCT_{A_d} is sent to the end user to complete the decryption.

5 Returns (TCT_{A_d})

Algorithm 18: Designcrypt.local - It is executed by IoT device user

Input : $(PP, TCT_{A_d}, RSK_{A_d})$

Output: M or $\perp$

1 It takes as input PP, partially decrypted ciphertext TCT_{A_d} and decryption retrieval private key RSK_{A_d}.

2 It returns message $M = C_0 \oplus H_3(\Sigma^{\xi.RSK_{A_d}/z}, \Upsilon_e, \Upsilon_s)$ or invalid message $\perp$ symbol.

3 Returns (M or $\perp$)

Table 2. Comparison of security requirements.

Scheme	Security				
	Authentication	Privacy	Confidentiality	Public verifiability	Unforgeability
[25]	√	√	√	√	√
[29]	√	√	√	√	√
[31]	√	√	√	√	√
[51]	√	×	√	×	√
[52]	×	×	√	×	√
[53]	√	×	√	×	√
[56]	×	×	×	√	√
[57]	√	√	√	√	√
Ours	√	√	√	√	√

Abbreviations: √: supports functionality. ×: Do not support functionality.

Table 3. Comparison of the ABSC schemes.

Scheme	**OKG**	**OKV**	**SM**	**OS**	**OD**	**AS**
[25]	No	No	Selective	No	No	Monotone span program
[29]	No	No	Selective	Yes	Yes	Monotone span program
[31]	No	No	Selective	No	Yes	Monotone span program
[51]	No	No	Selective	No	No	Threshold policy
[52]	No	No	Selective	No	No	Threshold structure
[53]	No	No	Selective	No	No	Threshold policy
[56]	No	No	Selective	No	No	Monotone span program
[57]	No	No	Selective	No	No	Monotone span program
Ours	Yes	Yes	Selective	Yes	Yes	Monotone span program

Abbreviations: OKG: Outsourced key generation, OKV: Outsourced key verification, SM: Security Model, OS: Outsourced signcryption, OD: Outsourced designcryption, AS: Access Structure.

Table 4. Comparison of private keys, signing keys, Public parameter size and Ciphertext size for different schemes.

Scheme	Private key size	Signing key size	Public parameter size	Ciphertext size
[25]	$(2\|A_d\|+2)L_{\mathcal{G}}$	$(\|A_s\|+2)L_{\mathcal{G}}$	$10L_{\mathcal{G}}+L_{\mathcal{G}_\tau}$	$(\ell_s+3\ell_e)L_{\mathcal{G}}+(2\ell_e+3)L_{\mathbb{Z}_p}$
[29]	$(\|U\|+3)L_{\mathcal{G}}$	$(\|U\|+3)L_{\mathcal{G}}$	$(\ell_s+5)L_{\mathcal{G}}$	$(4\ell_e+\ell_s+4)L_{\mathcal{G}}+L_{\mathcal{G}_\tau}$
[31]	$(\|A_d\|+2)L_{\mathcal{G}}$	$(\|A_s\|+2)L_{\mathcal{G}}$	$(2n+6)L_{\mathcal{G}}+L_{\mathcal{G}_\tau}$	$2(\ell_e+\ell_s+2)L_{\mathcal{G}}$
[51]	$3\|A_d\|L_{\mathcal{G}}$	$2\|A_s\|L_{\mathcal{G}}$	$(2n+5)L_{\mathcal{G}}+L_{\mathcal{G}_\tau}$	$(\ell_s+\ell_e+n_s+3)L_{\mathcal{G}}+L_M$
[52]	$4(\|A_d\|+1)L_{\mathcal{G}}$	$4(\|A_s\|+1)L_{\mathcal{G}}$	$(\|U\|+\|V\|+4)L_{\mathcal{G}}+L_{\mathcal{G}_\tau}$	$(\|U_d\|+\|V\|+\ell_s)L_{\mathcal{G}}+\|Sig\|+L_M$
[53]	$2(\|A_d\|+1)L_{\mathcal{G}}$	$2(\|A_s\|+1)L_{\mathcal{G}}$	$(2n+3)L_{\mathcal{G}}+L_{\mathcal{G}_\tau}$	$(2\ell_s+\ell_e+1)L_{\mathcal{G}}$
[56]	$(\|A_d\|+2)L_{\mathcal{G}}$	$(\|A_s\|+2)L_{\mathcal{G}}$	$2(n+1)L_{\mathcal{G}}+\|U\|L_{\mathcal{G}}+L_{\mathcal{G}_\tau}$	$2(\ell_e+\ell_s+2)L_{\mathcal{G}}$
[57]	$(\|A_d\|+2)L_{\mathcal{G}}$	$(\|A_s\|+2)L_{\mathcal{G}}$	$(2n+3)L_{\mathcal{G}}+L_{\mathcal{G}_\tau}$	$(\ell_s+\ell_e+4)L_{\mathcal{G}}$
Our scheme	$(\|A_d\|+2)L_{\mathcal{G}}$	$(\|A_s\|+2)L_{\mathcal{G}}$	$10L_{\mathcal{G}}+L_{\mathcal{G}_\tau}$	$(\ell_s+3\ell_e)L_{\mathcal{G}}+(2\ell_e+3)L_{\mathbb{Z}_p}$

Abbreviations: $|A_d|(|A_s|)$: indicates number of decryption(signing attributes),$\ell_e(\ell_s)$: Attribute size in encryption (signing) predicate, $L_{\mathcal{G}}$: denotes the length of an element in $\mathcal{G}$,$L_{\mathcal{G}_\tau}$: denotes the length of an element in $\mathcal{G}\tau$, $L_{\mathbb{Z}_p}$: denotes the length of an element in $\mathbb{Z}_p$, $|U_d|$: universe of encryption attributes, n: the number of elements, L_M: Length of the message, V: verification key size, $|Sig|$: size of message of one-time signature scheme.

Table 5. Comparison of signcryption cost and designcryption cost for different schemes.

Scheme	Outsourced signcryption/OFL cost	Local signcryption/ONL cost	Outsourced designcryption/OFL cost	Local designcryption/ONL cost
[25]	$(5\ell_e+4\ell_s+6)\mathcal{E}_1+\mathcal{E}_2$	—	—	$(3\ell_e+2\ell_s+2)\mathcal{P}+\mathcal{E}_2$
[29]	$(3\ell_e+4\ell_s)\mathcal{E}_1$	$(\ell_s+7)\mathcal{E}_1+\mathcal{E}_2$	$3\ell_e\mathcal{E}_1+\ell_e\mathcal{E}_2+(2\ell_e+1)\mathcal{P}$	$\mathcal{E}_2$
[31]	None	$(2\ell_e+4\ell_s+6)\mathcal{E}_1$	$(\ell_e+\ell_s+2)\mathcal{E}_1+\ell_e\mathcal{E}_2+(3\ell_s+2)\mathcal{P}$	$(\ell_s+2)\mathcal{E}_1+\mathcal{P}$
[51]	None	$(\ell_e+\ell_s+4)\mathcal{E}_1$	None	$\mathcal{E}_2+(2\ell_e+\ell_s+1)\mathcal{P}$
[52]	None	$(2\ell_e+3\ell_s+1)\mathcal{E}_1+\mathcal{E}_2$	None	$(4\ell_e+2\ell_s+3)\mathcal{P}$
[53]	None	$(\ell_e+2\ell_s+2)\mathcal{E}_1+\mathcal{E}_2$	None	$(2\ell_e+2\ell_s+1)\mathcal{P}$
[56]	None	$(2\ell_e+3\ell_s+3)\mathcal{E}_1+\mathcal{P}$	None	$(2\ell_e+3\ell_s+4)\mathcal{P}$
[57]	None	$(2\ell_e+4\ell_s+6)\mathcal{E}_1$	None	$(2\ell_e+\ell_s+3)\mathcal{E}_1+(2\ell_e+\ell_s+5)\mathcal{P}$
Our scheme	$(5\ell_e+4\ell_s+1)\mathcal{E}_1$	$(\ell_s+3)\mathcal{E}_1+\mathcal{E}_2$	$(6\ell_e+4)\mathcal{E}_1+(2\ell_e+3)\mathcal{P}$	$\mathcal{E}_2$

Abbreviations: $\ell_e(\ell_s)$: Attribute size in encryption (signing) predicate, $\mathcal{E}_1,\mathcal{E}_2$: Modular exponential computations in $\mathcal{G}$ and $\mathcal{G}_\tau$ respectively, $\mathcal{P}$: Pairing computation, OFL: Offline, ONL: Online.

Table 6. Signcryption and designcryption execution time.

Scheme	$\ell_e = 10, \ell_s = 10$				$\ell_e = 20, \ell_s = 20$			
	Signcryption execution time (ms)		Designcryption execution time (ms)		Signcryption execution time (ms)		Designcryption execution time (ms)	
	Outsourced	Local	Outsourced	Local	Outsourced	Local	Outsourced	Local
[25]	1191.40	—	—	1286.90	2304.89	—	—	2520.76
[29]	866.05	214.01	926.19	3.68	1732.09	337.73	1827.71	3.68
[31]	—	816.56	1098.66	173.14	—	1558.88	2123.23	296.86
[51]	—	296.93	—	768.67	—	544.37	—	1508.99
[52]	—	634.66	—	1554.66	—	1253.26	—	3035.30
[53]	—	399.59	—	1011.77	—	770.75	—	1998.85
[56]	—	680.40	—	1332.57	—	1299.00	—	2566.43
[57]	—	816.56	—	1271.98	—	1558.88	—	2383.46
Ours	1125.86	164.52	1359.39	3.68	2239.35	288.24	2595.26	3.68

5. Security analysis

Theorem 5.1 (Message Confidentiality:)**.** *Assuming the security in Rao's scheme [25] is assured, then the scheme proposed is also secure.*

Proof. Suppose an adversary $\mathcal{E}$ that has non-negligible advantage has ability to break VFOABSC scheme. In similar way an algorithm $\mathcal{Q}$ can break the scheme [25] with non-negligible advantage.
We take C as a challenger related to algorithm $\mathcal{Q}$ in the selective CPA-secure game due to the scheme proposed by Rao [25]. $\mathcal{Q}$ executes $\mathcal{E}$ as follows.

Setup To obtain public parameters PP', C runs **Setup** algorithm in [25]. The output is
$PP' = (\Sigma, g_T, g, h_e, u_e, w_e, u_s, w_s, \vartheta, \varrho_1, \varrho_2, \varrho_3, M, U, H_1, H_2, H_3, H_s)$
Then it sends PP' to $\mathcal{Q}$. Likewise, **Setup** algorithm is executed by $\mathcal{Q}$ in this paper to obtain the following public parameters
PP=$(\Pi, \Sigma, g, h_e, u_e, w_e, u_s, w_s, \nu, \eta_1, \eta_2, \eta_3, \mathcal{M}, U, H_1, H_2, H_3, H_s)$
It then sends PP to $\mathcal{E}$.

Query Phase 1 $\mathcal{Q}$ initialises both an empty Table $\mathbf{T}_1$ and an empty set R. Next, $\mathcal{E}$ issues queries adaptively as follows;

1. **DecKey Queries:** Similar to [62], we combine the simulation of DecKeyGen algorithms **DecKeyGen.out** and **DecKeyGen.local** as a single key query. When $\mathcal{E}$ makes a decryption query for the attribute set A_d, $\mathcal{Q}$ sends A_d to C to obtain decryption key SK_{A_d}. $\mathcal{Q}$ then sets $R = R \cup \{A_d\}$. Finally it sends SK_{A_d} to $\mathcal{E}$.

2. **DecKey.blind Query:** When $\mathcal{E}$ makes query for transformation key that corresponds to an attribute set A_d, $\mathcal{Q}$ will look for the entry tuple $(\varsigma, A_d, SK_d, TSK_{A_d}, RSK_{A_d})$ in table $\mathbf{T}_1$. If there exists the entry that corresponds to the query, $\mathcal{Q}$ will response with TSK_{A_d}, else an integer $z \in \mathbb{Z}_p^*$ is randomly selected by $\mathcal{Q}$, then computes $TSK_{A_d} = (TK_{d0}, TK_{d1}, \{TK_{d0,i}, TK_{d1,i}\}$ $\forall i \in A_d)$. where $TK_{d0} := g^{\alpha/z}\nu^{\delta/z}$, $TK_{d1} := g^{\delta/z}$, $TK_{d0,i} = (u_e^i h_e)^{\delta_i/z} w^{-\delta/z}$, $TK_{d1,i} = g^{\delta_i/z}$. TSK_{A_d} is taken as another type of SK_d. $\mathcal{Q}$ finally stores the tuple entry $(A_d, \diamond, SK'_{A_d}, \diamond)$ in Table $\mathbf{T}_1$ and return TSK_{A_d} to $\mathcal{E}$.

Challenge Phase: $\mathcal{Q}$ receives the challenge access structure $\Upsilon_e^*(A_d)$ from $\mathcal{E}$. $\mathcal{Q}$ then randomly chooses two messages M_0 and M_1 of equal length which he forwards to C to get a challenge ciphertext $SCT^*_{\Upsilon_e} = (C_0, C_1, \{C'_{i1}, C'_{i2}\} i \in [\ell_e], \sigma_0, \{\sigma_i\} i \in [\ell_s])$ by executing **Signcrypt** algorithm of [25]. $\mathcal{Q}$ finally sends $SCT^*_{\Upsilon_e}$ to $\mathcal{E}$.

Query Phase 2: $\mathcal{E}$ issues another sequence of queries with the condition that the querying key will not satisfy access structure. $\mathcal{Q}$ responds in similar way to the one simulated in **Query Phase 1** and provides responds as in **Query Phase 1**.

Guess Phase: $\mathcal{E}$ provides as an output guess γ likewise $\mathcal{Q}$ gives its output as γ.

In keeping with the earlier discussion, suppose $\mathcal{E}$ can attack our VFOABSC scheme with non-negligible advantage in the selective CPA-secure game. Likewise an algorithm $\mathcal{Q}$ can attack the scheme [25]. □

Theorem 5.2 (Ciphertext Unforgeability:)**.** *The proposed VFOABSC scheme is unforgeable if CDH hardness assumption holds.*

Proof. Assume an adversary $\mathcal{A}$ can attack VFOABSC scheme with non-negligible advantage, then we can build an algorithm simulation $\mathcal{Q}$ that uses $\mathcal{A}$ to solve CDH problem. Provided with a tuple $(g, A = g^a, B = g^b) \in \mathcal{G}^3$ which is a random CDH instance, the role of C is to compute g^{ab} where $a, b \in_R \mathbb{Z}_p^*$

Setup $\mathcal{Q}$ sets parameters as in **Theorem 5.1**. Selects randomly $a, b \in_R \mathbb{Z}_p^*$. Then sets $\Sigma = e(g^a, g^b)$. The aim of $\mathcal{Q}$ is to compute g^{ab}.

Queries C takes an empty Table $\mathbf{T}_1$ with an empty set R. Next, $\mathcal{Q}$ issues sequences of queries adaptively as follows;

1. **DecKey Queries** and **DecKey.blind Query**. These are similar to the above described CPA-game.
2. **SigKey Query**. We combine the SigKey algorithms simulation **SigKey.out** and **SigKey.local** as a single key query. When $\mathcal{A}$ makes a signing key query for the attribute set A_s, $\mathcal{Q}$ sends A_s to C to obtain signing key SK_{A_s}. $\mathcal{Q}$ then sets $R = R \cup \{A_s\}$. Finally it sends SK_{A_s} to $\mathcal{A}$.
3. **SigKey.blind Query** Assuming $\mathcal{A}$ makes a transformation secret signing key query associated with attributes A_s. $\mathcal{A}$ will find out if its stored in tuple $(A_s, SK_{A_d}, SK_{A_s}, TSK_{A_d}, RSK_{A_d})$. If it exists, $\mathcal{Q}$ returns TSK_{A_s}, otherwise it picks at random $\phi \in_R \mathbb{Z}_p^*$ then computes $TSK_{A_s} = [A_s, TK_{s1}, TK_{s2}, \{TK_{s,j}\}_{j \in A_s}$ where $TK_{s1} = g^{\alpha/\phi}\nu^{\delta'/\phi}$, $TK_{s2} = g^{\delta'/\phi}$, $TK_{s,j} = H_s(a)^{\delta'/\phi}$, $\forall j \in A_s$.

TSK_{A_s} is represented as another kind of SK_{A_s}. For that matter, in the data owner's view there is similarity between third party (cloud) and end users. Lastly, the tuple record $(A_s, \diamond, SK'_{A_s}, \diamond)$ is stored in $\mathbf{T}_1$ and TSK_{A_s} is returned to $\mathcal{A}$.

Forgery $\mathcal{A}$ outputs forged ciphertext $SCT^*_{\Upsilon_e}$, constructed under attribute set A_s with the constraint that $A_s \notin \mathbf{T}_1$. C aborts if $A^*_s = A'_s$

Whenever the signcryption ciphertext is valid, $\mathcal{A}$ wins and obtains

$$\frac{e(\sigma_0, g^f)}{e(u_s^{\zeta} w_s, C_{01}^f).(\Pi_{i\in[\ell_s]} e(v^{\varpi_i} H_s(\rho_s(i))^f, g^{\psi.c_i+\zeta x_i})}$$

$$e(g^{ab}, g^f) = \frac{e(\sigma_0, g^f)}{e((u_s^{\zeta} w_s)^{\xi}, g^f).(\Pi_{i\in[\ell_s]} e(v^{\varpi_i} H_s(\rho_s(i))^{\psi.c_i+\zeta x_i}, g^f)}$$

$$g^{ab} = \frac{\sigma_0}{(u_s^{\zeta} w_s)^{\xi}.(\Pi_{i\in[\ell_s]} (v^{\varpi_i} H_s(\rho_s(i))^{\psi.c_i+\zeta x_i}} \text{ as a solution of CDH problem.}$$

□

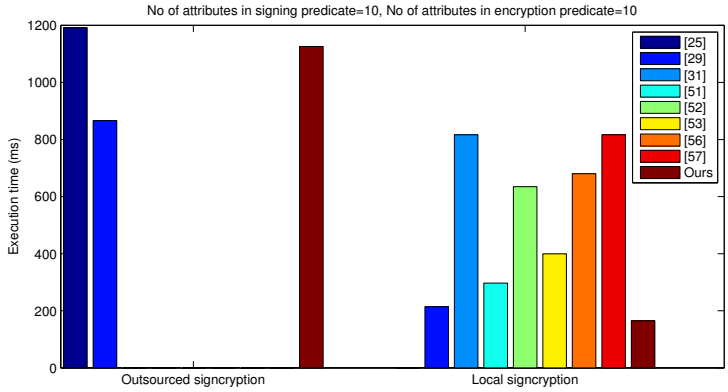


Figure 5. Signcryption 1.

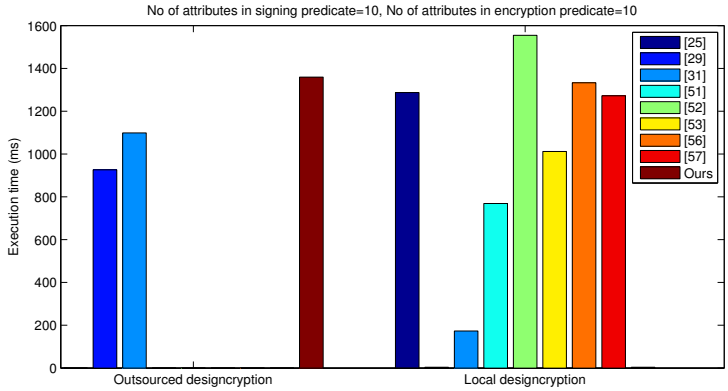


Figure 6. Designcryption 1.

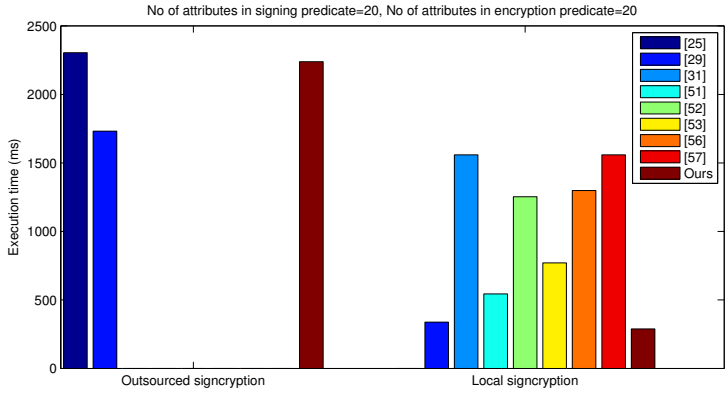


Figure 7. Signcryption 2.

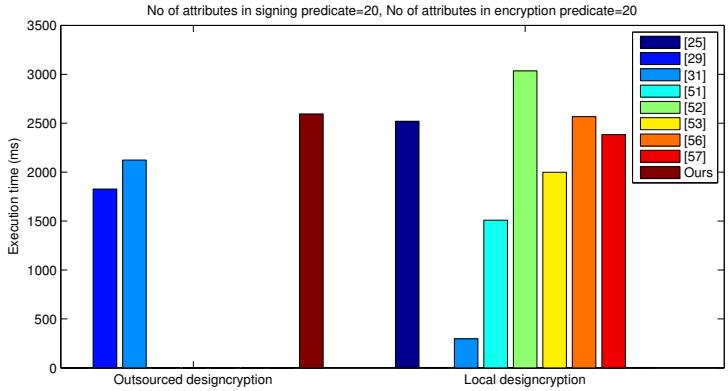


Figure 8. Designcryption 2.

6. Performance

This section describes the performance comparisons of several existing ABSC schemes [25, 29, 31, 51–53, 56, 57] with our scheme. Table 2 provides a summary of the security requirements in terms of authentication, privacy, confidentiality, public verifiability and unforgeability. From the shown results, schemes [25,29,31,57] like our scheme enjoys all the security requirements mentioned above. Scheme [52, 56] do not provide authentication, while [51–53, 56] do not support signcryptor's privacy. On the other hand schemes [51–53] do not support public verifiability. All the schemes supports unforgeability security requirement. In Table 3, we compared the schemes in terms of functionality. Its only our VFOABSC scheme that achieves outsourcing key generation, outsourcing key generation verification, outsourcing signcryption and outsourcing designcryption simultaneously. The results in Table 3 indicates that our proposed scheme is more efficient in comparison with other schemes, since heavy computations are offloaded to the third party whose computation processing power is immense. As far as we know, our proposed VFOABSC scheme is the first ABSC scheme in literature that realizes fully outsourcing of expensive computations and therefore possesses promising and desirable properties applicable to resource-constrained IoT devices. Furthermore, similar to [25, 29, 31, 56, 57] our scheme

adopted monotone span program rich in flexible expressions. Previous existing works [51–53] were constructed under threshold policy which is coarse-grained in nature and supports simple predicates and therefore do not have wider applications. Table 4 describes storage cost comparisons. Similar to schemes [31, 56, 57] our scheme has equal length size of private key and signing key which is slightly smaller in comparison with the sizes of other schemes. Scheme [29] has larger ciphertext size compared to other schemes. Scheme [25] do not have local signcryption computations. Abundant computation overhead is outsourced. Our scheme outsources majority of signcryption computations and therefore suffers less local computations costs compared to [29].

Table 7. Performance evaluation benchmark.

Operation	Notation	Time computation (in *ms*)
Exponentiation in $\mathcal{G}$	$T_{\mathcal{E}_1}$	12.372094
Exponentiation in $\mathcal{G}_\tau$	$T_{\mathcal{E}_2}$	3.680760
Bilinear Pairing	$T_{\mathcal{P}}$	24.677217

Table 5 presents the computation costs comparisons of signcryption (outsourced and local) and designcryption (outsourced and local) algorithms. During signcryption operation process, the data owner in our scheme and scheme [29] borrows computation power from the cloud service provider to generate partial ciphertext that is related to encryption and signing predicates. It will then use the portion generated to compute the remaining part of ciphertext. The cost burden on the data owner side reduces significantly. In designcryption phase, the final user in our scheme like in [29, 31], only suffers single exponentiation cost in $\mathcal{G}_\tau$ compared to other schemes [25, 51–53, 56, 57].

For computation efficiency comparisons, we implemented our experiment using Stanford Pairing-Based Crypto (PBC) library [63] in VC++ 6.0. A laptop equipped with 2.67GHz Intel Core *i*3-M390 CPU and 8GB RAM executing in 64-bit Windows 10 operating system is used in our implementation. The size of $\mathcal{G}$ and $\mathbb{Z}_p$ is set to 64B (i.e 512 bits) whereas the size of $\mathcal{G}_\tau$ is set to 128B (i.e 1024 bits). Moreover, we adopted type A bilinear [63], in this case we employed supersingular curve $y^2 = x^3 + x$ defined when supplying ECC group. From the above stated settings, we obtained the results as shown in Table 7. Notation $T_{\mathcal{E}_1}$ denotes exponentiation computation time in $\mathcal{G}$, $T_{\mathcal{E}_2}$ denotes exponentiation computation time in $\mathcal{G}_\tau$ while $T_{\mathcal{P}}$ denotes pairing computation time. In Table 6 we make comparison of the execution time for both outsourced and local signcryption and outsourced and local designcryption of our scheme against schemes [25, 29, 31, 51–53, 56, 57] by utilizing 10 and 20 number of attributes (in both signing and encryption predicates). Analyzing from the output, our scheme compared to others takes minimal execution time in running both local signcryption (164.52 *ms*) and local designcryption (3.68 *ms*) by employing 10 number of attributes (in both signing and encryption predicates) and (288.24 *ms*) and (3.68 *ms*) respectively for 20 number of attributes (in both signing and encryption predicates). This is because we outsourced expensive computation to the cloud server. Figure 5 and Figure 6 shows the corresponding graphs of signcryption (both outsourced and local) and designcryption (both outsourced and local) respectively for 10 number of attributes (in both signing and encryption predicates) while Figure 7 and Figure 8 presents the corresponding graphs of signcryption (both outsourced and local) and designcryption (both outsourced and local) respectively using size of 20 attributes (in both

encryption and signing predicates). Therefore from the indicated results, our proposed scheme surpass those of the existing schemes in terms of computation overhead while at the same time achieving the desired security goals.

7. Conclusion

This paper proposed ABSC scheme that is fully outsourced which as far as we know is the first work in literature that is fully outsourced. The scheme alleviates burden from IoT devices with limited resources and therefore improves computation efficiency without jeopardizing data owner's privacy. To realize this, we enhanced the scheme due to Rao [25] where we offloaded complex computations such as pairing and exponentiations operations to the cloud to make use of available abundant and scalable resources. Simple remaining computations is executed by respective IoT device user. In addition, our scheme supports confidentiality, fine-grained access control, unforgeability, public privacy, authentication and public verifiability properties. We also presented security proof to demonstrate that our proposed scheme is CPA-secure. Moreover, the simulation output indicates that our proposed scheme is efficient and therefore applicable to resource-constrained IoT devices due to limited storage capacity and lower computation power.

Our future work will include building provably secure fully outsourced ABSC scheme that supports policy updating in IoT devices while at the same time achieving CCA2 security.

Acknowledgments

The authors appreciate the work done by anonymous reviewers and their valuable comments. This work was supported by National Natural Science Foundation of China under Grant No. 61602097, Sichuan Science-Technology Support Plan Program under Grant Nos. 2016GZ0065, 2016ZC2575 and 17ZA0322, Fundamental Research Funds for the Central Universities under Grant No. ZYGX2015J072.

Conflict of interest

All authors declare that there is no conflicts of interest in this paper.

References

1. L. Atzori, A. Iera and G. Morabito, The internet of things: A survey, *Comp. Net.*, **54** (2010), 2787–2805.
2. C. Chen, B. Xiang, Y. Liu, et al., A secure authentication protocol for internet of vehicles, *IEEE Access*, **7** (2019), 12047–12057.
3. K. H. Wang, C. M. Chen, W. Fang, et al., On the security of a new ultra-lightweight authentication protocol in iot environment for rfid tags, *J. Supercomp.*, **74** (2018), 65–70.
4. K. Yeh, A secure transaction scheme with certificateless cryptographic primitives for iot-based mobile payments, *IEEE Sys. J.*, **12** (2018), 2027–2038.

5. Z. Qin, Y. Wang, H. Cheng, et al., Demographic information prediction: A portrait of smartphone application users, *IEEE T. Emer. Top. Comput.*, **6** (2018), 432–444.
6. D. Bandyopadhyay and J. Sen, Internet of things: Applications and challenges in technology and standardization, *Wireless Pers. Commun.*, **58** (2011), 49–69.
7. H. K. Maji, M. Prabhakaran and M. Rosulek, Attribute-based signatures, in *Topics in Cryptology–CT-RSA 2011* (ed. A. Kiayias), Springer Berlin Heidelberg, Berlin, Heidelberg, (2011), 376–392.
8. H. Xiong, H. Zhang and J. Sun, Attribute-based privacy-preserving data sharing for dynamic groups in cloud computing, *IEEE Sys. J.*, 1–22.
9. A. Sahai and B. Waters, Fuzzy identity-based encryption, in *Proc. of the 24th Annual Int. Conf. on Theory and App. of Crypto. Tech.*, EUROCRYPT'05, Springer-Verlag, Berlin, Heidelberg, (2005), 457–473.
10. A. Shamir, Identity-based cryptosystems and signature schemes, in *Advances in Cryptology* (eds. G. R. Blakley and D. Chaum), Springer Berlin Heidelberg, Berlin, Heidelberg, (1985), 47–53.
11. S. Lin, R. Zhang, H. Ma, et al., Revisiting attribute-based encryption with verifiable outsourced decryption, *IEEE T. Inform. Fore. Sec.*, **10** (2015), 2119–2130.
12. J. Sun, Y. Bao, X. Nie, et al., Attribute-hiding predicate encryption with equality test in cloud computing, *IEEE Access*, **6** (2018), 31621–31629.
13. Q. Huang, Y. Yang and L. Wang, Secure data access control with ciphertext update and computation outsourcing in fog computing for internet of things, *IEEE Access*, **5** (2017), 12941–12950.
14. X. Li, R. Lu, X. Liang, et al., Smart community: an internet of things application, *IEEE Com. Mag.*, **49** (2011), 68–75.
15. S. S. Chow, A framework of multi-authority attribute-based encryption with outsourcing and revocation, in *Proc. of the 21st ACM on Symp. on Acc. Cont. Models and Tech.*, SACMAT '16, ACM, New York, NY, USA, (2016), 215–226.
16. M. Li, W. Lou and K. Ren, Data security and privacy in wireless body area networks, *IEEE Wir. Com.*, **17** (2010), 51–58.
17. V. Goyal, O. Pandey, A. Sahai, et al., Attribute-based encryption for fine-grained access control of encrypted data, in *Proc. of the 13th ACM Conf. on Comp. and Com. Sec.*, CCS '06, ACM, New York, NY, USA, (2006), 89–98.
18. A. Lewko and B. Waters, Unbounded hibe and attribute-based encryption, in *Proc. of the 30th Annual Int. Conf. on Theory and Appl. of Crypt. Techn.: Advances in Cryptology*, EUROCRYPT'11, Springer-Verlag, Berlin, Heidelberg, (2011), 547–567.
19. J. Bethencourt, A. Sahai and B. Waters, Ciphertext-policy attribute-based encryption, in *Proc. of the 2007 IEEE Symp. on Sec. and Priv.*, SP '07, IEEE Computer Society, Washington, DC, USA, (2007), 321–334.
20. A. Lewko, T. Okamoto, A. Sahai, et al., Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption, in *Advances in Cryptology – EUROCRYPT 2010* (ed. H. Gilbert), Springer Berlin Heidelberg, Berlin, Heidelberg, (2010), 62–91.
21. C. Fan, V. S. Huang and H. Ruan, Arbitrary-state attribute-based encryption with dynamic membership, *IEEE T. Comp.*, **63** (2014), 1951–1961.

22. H. Maji, M. Prabhakaran and M. Rosulek, Attribute-based signatures: Achieving attribute-privacy and collusion-resistance, (2008).
23. J. Li, M. H. Au, W. Susilo, et al., Attribute-based signature and its applications, in *Proc. of the 5th ACM Symp. on Inf., Comp. and Com. Sec.*, ASIACCS '10, ACM, New York, NY, USA, (2010), 60–69.
24. M. Gagné, S. Narayan and R. Safavi-Naini, Short pairing-efficient threshold-attribute-based signature, in *Proc. of the 5th Int. Conf. on Pairing-Based Cryptography*, Pairing'12, Springer-Verlag, Berlin, Heidelberg, (2013), 295–313.
25. Y. S. Rao, *Int. J. Communication Systems*, **30**, Available from: `https://doi.org/10.1002/dac.3322`.
26. C. T. Li, C. L. Chen, C. C. Lee, et al., A novel three-party password-based authenticated key exchange protocol with user anonymity based on chaotic maps, *Soft Comput.*, **22** (2018), 2495–2506.
27. J. C. W. Lin, Q. Liu, P. Fournier-Viger, et al., Anonymization of multiple and personalized sensitive attributes, in *20th Int. Conf., DaWaK 2018 on Big Data Anal. and Know. Disc.* (eds. C. Ordonez and L. Bellatreche), Springer International Publishing, Cham, (2018), 204–215.
28. J. C. W. Lin, Y. Zhang, P. Fournier-Viger, et al., A metaheuristic algorithm for hiding sensitive itemsets, in *Database and Expert Systems Applications* (eds. S. Hartmann, H. Ma, A. Hameurlain, G. Pernul and R. R. Wagner), Springer International Publishing, Cham, (2018), 492–498.
29. Q. Xu, C. Tan, Z. Fan, et al., Secure data access control for fog computing based on multi-authority attribute-based signcryption with computation outsourcing and attribute revocation, *Sensors*, **18** (2018), 1609.
30. C. M. Chen, K. H. Wang, W. Fang, et al., Reconsidering a lightweight anonymous authentication protocol, *J. Chin. Insti. Eng.*, **42** (2019), 9–14.
31. F. Deng, Y. Wang, L. Peng, et al., Ciphertext-policy attribute-based signcryption with verifiable outsourced designcryption for sharing personal health records, *IEEE Access*, **6** (2018), 39473–39486.
32. S. M. Sedaghat, M. H. Ameri, M. Delavar, et al., An efficient and secure attribute-based signcryption scheme for smart grid applications, Cryptology ePrint Archive, Report 2018/263, (2018).
33. Q. Xu, C. Tan, Z. Fan, et al., Secure multi-authority data access control scheme in cloud storage system based on attribute-based signcryption, *IEEE Access*, **6** (2018), 34051–34074.
34. H. Wang, D. He, J. Shen, et al., Verifiable outsourced ciphertext-policy attribute-based encryption in cloud computing, *Soft Comp.*, **21** (2017), 7325–7335.
35. M. Green, S. Hohenberger and B. Waters, Outsourcing the decryption of abe ciphertexts, in *Proc. of the 20th USENIX Conf. on Security*, SEC'11, USENIX Association, Berkeley, CA, USA, (2011), 34–34.
36. J. Li, W. Yao, Y. Zhang, et al., Flexible and fine-grained attribute-based data storage in cloud computing, *IEEE T. Serv. Comp.*, **10** (2017), 785–796.
37. H. Xiong, K. R. Choo and A. V. Vasilakos, Revocable identity-based access control for big data with verifiable outsourced computing, *IEEE T. B. Data*, 1–1.

38. H. Xiong and J. Sun, Comments on "verifiable and exculpable outsourced attribute-based encryption for access control in cloud computing", *IEEE T. Dep. Sec. Comp.*, **14** (2017), 461–462.
39. L. Jiguo, S. Fengjie, Z. Yichen, et al., Verifiable outsourced decryption of attribute-based encryption with constant ciphertext length, *Sec. Com. Net.*, (2017).
40. J. Lai, R. H. Deng, C. Guan, et al., Attribute-based encryption with verifiable outsourced decryption, *IEEE T. Info. Fore. Sec.*, **8** (2013), 1343–1354.
41. X. Mao, J. Lai, Q. Mei, et al., Generic and efficient constructions of attribute-based encryption with verifiable outsourced decryption, *IEEE T. Dep. Sec. Comp.*, **13** (2016), 533–546.
42. R. Zhang, H. Ma and Y. Lu, Fine-grained access control system based on fully outsourced attribute-based encryption, *J. Sys. Soft.*, **125** (2017), 344–353.
43. P. Yang, Z. Cao and X. Dong, Fuzzy identity based signature, Cryptology ePrint Archive, Report 2008/002, (2008).
44. G. Shanqing and Z. Yingpei, Attribute-based signature scheme, in *2008 International Conference on Information Security and Assurance (isa 2008)*, (2008), 509–511.
45. S. Y. Tan, S. H. Heng and B. M. Goi, On the security of an attribute-based signature scheme, in *U- and E-Service, Science and Technology* (eds. D. Ślęzak, T. H. Kim, J. Ma, W. C. Fang, F. E. Sandnes, B. H. Kang and B. Gu), Springer Berlin Heidelberg, Berlin, Heidelberg, (2009), 161–168.
46. J. Herranz, F. Laguillaumie, B. Libert, et al., Short attribute-based signatures for threshold predicates, in *Topics in Cryptology – CT-RSA 2012* (ed. O. Dunkelman), Springer Berlin Heidelberg, Berlin, Heidelberg, (2012), 51–67.
47. T. Okamoto and K. Takashima, Efficient attribute-based signatures for non-monotone predicates in the standard model, in *Public Key Cryptography–PKC 2011* (eds. D. Catalano, N. Fazio, R. Gennaro and A. Nicolosi), Springer Berlin Heidelberg, Berlin, Heidelberg, (2011), 35–52.
48. T. Okamoto and K. Takashima, Decentralized attribute-based signatures, in *Public-Key Cryptography–PKC 2013* (eds. K. Kurosawa and G. Hanaoka), Springer Berlin Heidelberg, Berlin, Heidelberg, (2013), 125–142.
49. H. Xiong, Q. Mei and Y. Zhao, Efficient and provably secure certificateless parallel key-insulated signature without pairing for iiot environments, *IEEE Sys. J.*, 1–11.
50. S. F. Shahandashti and R. Safavi-Naini, Threshold attribute-based signatures and their application to anonymous credential systems, in *Progress in Cryptology–AFRICACRYPT 2009* (ed. B. Preneel), Springer Berlin Heidelberg, Berlin, Heidelberg, (2009), 198–216.
51. M. Gagné, S. Narayan and R. Safavi-Naini, Threshold attribute-based signcryption, in *Sec. and Crypt. for Net.* (eds. J. A. Garay and R. De Prisco), Springer Berlin Heidelberg, Berlin, Heidelberg, (2010), 154–171.
52. K. Emura, A. Miyaji and M. S. Rahman, Dynamic attribute-based signcryption without random oracles, *Int. J. Appl. Cryptol.*, **2** (2012), 199–211.
53. C. Hu, N. Zhang, H. Li, et al., Body area network security: A fuzzy attribute-based signcryption scheme, *IEEE J. Sel. Areas. Com.*, **31** (2013), 37–46.

54. Y. Han, W. Lu and X. Yang, Attribute-based signcryption scheme with non-monotonic access structure, in *Proc. of the 2013 5th Int. Conf. on Intel. Net. and Collaborative Systems*, INCOS '13, IEEE Computer Society, Washington, DC, USA, (2013), 796–802.
55. H. Hong and Z. Sun, An efficient and secure attribute based signcryption scheme with lsss access structure, *SpringerPlus*, **5** (2016), 644.
56. J. Liu, X. Huang and J. K. Liu, Secure sharing of personal health records in cloud computing: Ciphertext-policy attribute-based signcryption, *Fut. Gen. Comp. Sys.*, **52** (2015), 67–76.
57. Y. S. Rao, A secure and efficient ciphertext-policy attribute-based signcryption for personal health records sharing in cloud computing, *Fut. Gen. Comp. Sys.*, **67** (2017), 133–151.
58. Y. S. Rao and R. Dutta, Expressive bandwidth-efficient attribute based signature and signcryption in standard model, in *Info. Sec. Priv.* (eds. W. Susilo and Y. Mu), Springer International Publishing, Cham, (2014), 209–225.
59. H. Yiliang and L. Wanyi, Attribute based generalized signcryption for online social network, in *2015 34th Chin. Control Conf.*, (2015), 6434–6439.
60. A. Beimel, *Secure schemes for secret sharing and key distribution* Ph.D thesis, Technion-Israel Institute of technology, Faculty of computer science, 1996.
61. A. Lewko and B. Waters, Decentralizing attribute-based encryption, in *Proc. of the 30th Annual Int. Conf. on Theory and App. of Crypt. Tech.: Advances in Cryptology*, EUROCRYPT'11, Springer-Verlag, Berlin, Heidelberg, (2011), 568–588.
62. S. Wang, K. Liang, J. K. Liu, et al., Attribute-based data sharing scheme revisited in cloud computing, *IEEE T. Info. For. Sec.*, **11** (2016), 1661–1673.
63. B. Lynn., The stanford pairing based crypto library. Available from: `https://crypto.stanford.edu/pbc/`.

AIMS Press