Cryptographic Attribute-Based Access Control (ABAC) for Secure Decision Making of Dynamic Policy With Multiauthority Attribute Tokens

Yan Zhu, Ruyun Yu, Di Ma, and William Cheng-Chung Chu, *Senior Member, IEEE*

***Abstract*—This article aims to establish a cryptographic solution to improve security and reliability of the National Institute of Standards and Technology's attribute-based access control (ABAC) model. By breaking down the existing structure of attribute-based encryption, we propose a new cryptographic ABAC (C-ABAC) framework with dynamic policy authorization and real-time attribute credentials. Moreover, a practical C-ABAC construction is proposed to support provable policy decision making and verifiable attribute Tokens among multiple distributed authorities. In this construction, we develop a concrete approach of generating a cryptographic policy from access control markup language. We also prove that attribute Token has existential unforgeability under chosen-attribute and chosen-nonce attacks, and the cryptographic policy is existentially unforgeable under chosen-object attack. In addition, our C-ABAC construction provides semantic security against chosen-plaintext attack with Token and policy queries under the extended general Diffie–Hellman exponent assumption. Finally, we evaluate the performance of the C-ABAC system according to complexity analysis and experimental results. The results show that the C-ABAC system is reliable and easy to implement.**

***Index Terms*—Access control, attribute Tokens, cryptographic policy, cryptography, reliability, secure decision making.**

I. Introduction

ATTRIBUTE-BASED access control (ABAC) [1], [2], considered as the latest milestone in the evolution of access control models, has gained widespread acceptance among government, industry, and academic circles. The reason is that it can provide rich semantics, high flexibility, fine granularity, and other beneficial features like customizable policy and context-aware authentication. Moreover, it ensures that corporate policies are flexibly enforced on many different levels within the infrastructure: within individual applications, in web portals or gateways, within an enterprise service bus, etc. These peculiarities are more advantageous as organizations or institutions move to greater collaboration and resource sharing outside and across the enterprise, or many hot fields, e.g., Big data, Blockchain, Internet-of-Things (IoT), and Cloud computing [3].

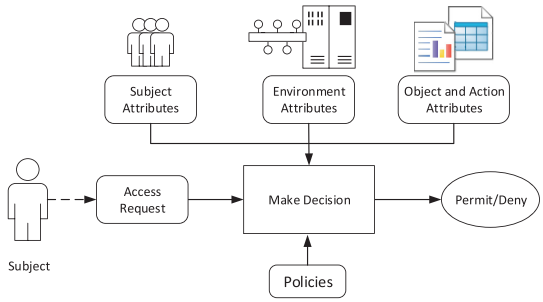

Fig. 1. General model of ABAC.

In Fig. 1, we show the general ABAC model, which consists of four kinds of entity attributes: object attributes describe the resource being accessed, e.g., author or owner, region, classification or sensitivity, created date, and time-to-live; subject attributes describe the user attempting the access, e.g., name, department, role, and job title; action attributes describe the operation being attempted, e.g., browse, modify, delete, and append; and environment attributes deal with location, time, or dynamic aspects of the access scenario. By using them, ABAC can combine multiple attributes to make a context-aware decision based on fine-grained attributes evaluated at runtime [4]. In light of this framework, ABAC is thought to promote higher flexibility, scalability, granularity, and manageability for dynamic authorizations. More exactly, it provides two important features.

1) *Real-time attribute assignments* is an effective mechanism, by which real-time attribute values can be acquired from different entities associated with subject, object, action, or environment when the access requests occur.
2) *Dynamic policy-based authorization* is an approach, in which access rights are granted dynamically by evaluating instance-specific policies with attribute credentials.

Manuscript received January 2, 2019; revised February 10, 2019; accepted March 21, 2019. Date of publication November 21, 2019; date of current version November 26, 2019. This work was supported in part by the National Key Technologies R&D Programs of China under Grant 2018YFB1402702, in part by the National Natural Science Foundation of China under Grant 61972032, and in part by the China Scholarship Council. Associate Editor: C.-Y. Huang. *(Corresponding author: William Cheng-Chung Chu.)*

Y. Zhu and R. Yu are with the School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China (e-mail: zhuyan@ustb.edu.cn; yuruyun@xs.ustb.edu.cn).

D. Ma is with the Department of Computer and Information Science, University of Michigan–Dearborn, Dearborn, MI 48128 USA (e-mail: dmadma@umich.edu).

W. C.-C. Chu is with the Department of Computer Science, Tunghai University, Taichung 40704, Taiwan (e-mail: cchu@thu.edu.tw).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org provided by the authors.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2019.2948713

To implement and enforce ABAC, several general policy specifications or languages, e.g., next-generation access control [5] and extensible access control markup language (XACML) [6], have been proposed to allow administrators to capture and author corporate policies in an exact and standardized way. Besides policy language and request/response process, XACML also provides another important mechanism, called "target," which can be used to find a policy that can apply to a certain request from a large collection of stored policies (called "PolicySet"). These specifications and mechanisms will be especially conducive to solving a policy management problem.

Although ABAC has many enticing benefits, this model is facing with various security threats and attacks, such as attribute forgery and permission counterfeiting. The system reliability is also incrementally decreased along with the expansion of resource sharing space and enterprise scale. This is especially remarkable for open environments, e.g., Cloud [7], Blockchain, and IoT. A solution to this problem is using cryptographic access control, e.g., *attribute-based encryption (ABE)* [8], [9], proposed over a decade ago. In ABE, the data provider will provide an attribute-based access policy $can\_access(X)$ for some designated attributes X, and each requester will be assigned a private key associated with attribute credentials x. This requester will decrypt a ciphertext associated with $can\_access(X)$ if and only if $can\_access(x) = 1$. ABE can provide a flexible and effective authorization based on strong expressive capability of access policies.

Although ABE supports data confidentiality, existing ABE schemes [10] still have significant limitations for supporting dynamicity and real-time ability, as described above. The reason is that ABE is constructed over the traditional encryption framework with quadruples, i.e., (Setup, KeyGen, Encrypt, Decrypt), rather than over the ABAC framework with four service nodes, i.e., *policy enforcement point (PEP), policy decision point (PDP), policy administration point (PAP), and policy information point (PIP)* (see Section II-B). Limited to the static and invariant property of the encryption framework, ABE has the following shortcomings.

1) *Not for real-time attribute assignments:* Considering that the user's private key will remain unchanged after it has been generated by $KeyGen$, the attribute credentials attached to the key would not be changed dynamically in real time. Therefore, ABE cannot support dynamic attributes, such as environment attributes.
2) *Not for dynamic policy-based authorization:* Since the access policy should be bound to the ciphertext of protected object after Encrypt, the policies could not be changed automatically with evolution of the system so that ABE might not meet the requirements for dynamic application scenarios.

Due to these two shortcomings, it is infeasible for ABE to provide an effective attribute expression for environment conditions. Hence, the existing ABE framework cannot meet the requirements of practical applications with dynamic attributes, which may change along with time, location, traffic volume, or bandwidth. This also means a lack of dynamicity and real-time ability in ABE. Therefore, existing ABE schemes do not support the ABAC model very well.

To remedy this situation, the purpose of our article is to develop a novel cryptographic solution for the ABAC model, which offers overall support for dynamic authorization and real-time attribute credentials. To do this, we need to break down existing structures of ABE and design a novel cryptographic framework in full consistency with the distributed ABAC model. This new kind of framework would provide a higher level of reliability and security without additional modifications to the fundamental structure of ABAC.

According to the above goal, the ABAC model should be innovated by the following four new security mechanisms.

1) A new secure representation of policy, called *cryptographic policy*, could be generated and bound dynamically to the ciphertext of the protected object against counterfeiting, tampering, and replaying attacks.
2) Instead of the user's private key, *one-time attribute Tokens* should be produced dynamically to implement attribute credentials in real time.
3) A cryptographically verifiable method must be utilized to guarantee the validity of policy decision making, according to attribute Tokens and cryptographic policy, which is called *secure policy decision-making*.
4) *Multiple attribute authorities* should be deployed to provide decentralized authorization, such that the system reliability could be improved even though several authorities are broken down or attacked.

Our contribution: The objective of this article is to develop a new secure ABAC framework for implementing dynamic authorization over diverse attributes in real time in accordance with the standard National Institute of Standards and Technology (NIST)'s distributed ABAC model. This framework is able to support provable decision making of access policies that can be dynamically bound to protected objects, as well as verifiable attribute credentials derived from multiple authorities. Based on this, we present a new *cryptographic attribute-based access control (C-ABAC)* model and conduct research on the following aspects.

1) The workflow and the framework of the C-ABAC model are specified on strictly abiding by the distributed ABAC model. Meanwhile, two new mechanisms, *attribute Tokens* and *cryptographic policy*, are presented for assuring secure decision making and credential exchanging among multiple attribute authorities.
2) We present a C-ABAC construction to completely support multisource attribute Tokens and secure policy decision making in real time. Especially, an approach of generating cryptographic policy built upon the XACML policy specification and linear secret-sharing scheme (LSSS) is designed to implement cryptographic representation of dynamic policies.
3) Based on strong Diffie–Hellman (SDH) and computational Diffie–Hellman (CDH) assumptions, we prove that attribute Tokens are existentially unforgeable against chosen-nonce attacks (CNAs) and chosen-attribute attacks (CAAs), simultaneously. The cryptographic policy

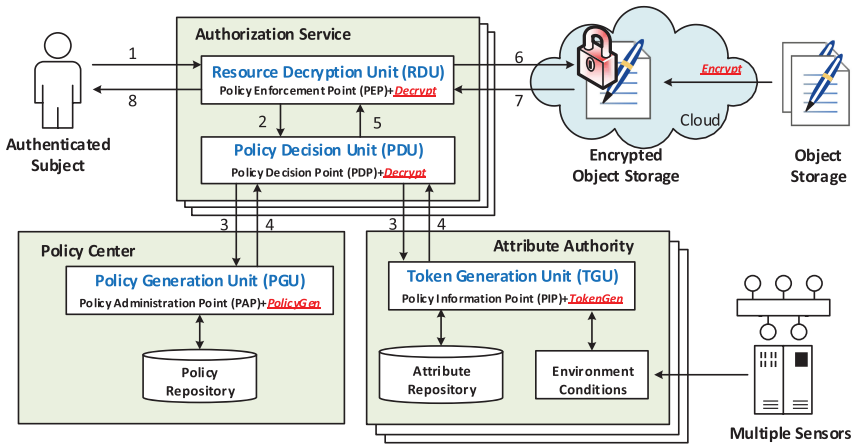

Fig. 2. Framework of the C-ABAC model.

is proved to be existentially unforgeable against chosen-object attack (COA) under the extended decisional bilinear Diffie–Hellman (eBDH) assumption. Moreover, the C-ABAC construction is proved to be semantically secure against chosen-plaintext attack with Token and policy queries under the extended general Diffie–Hellman exponent (eGDHE) assumption.

We give the complexity analysis of C-ABAC construction in theory and make an experimental verification for the performance of the C-ABAC prototype system. The results from experiments indicate that our C-ABAC system is reliable and easy to implement.

Organization: In Section II, we propose a new system architecture and the C-ABAC model. In Sections III and IV, we present the construction of the C-ABAC scheme and the cryptographic policy, respectively. Next, security analysis and performance evaluation are discussed in Sections V and VI, respectively. Finally, Section VII concludes this article.

II. Architecture and Mechanism

A. Scope and Purpose

This article aims to describe a new cryptographic method on the standard ABAC model to manage the encrypted resources in open environments, e.g., Cloud, Blockchain, and IoT. This method could provide an available solution to verify the validity of policy decision making and support diversification and dynamicity of attributes. The main challenge of meeting this goal is to construct an efficient and provably secure ABAC framework, which should be sufficient to provide the following security functions.

1) *Escrowed object encryption* ensures that the decryption of an object is determined by the access policy, without the original owner's intervention.
2) *Secure decision making of dynamic policy* is used to verify whether or not the subject's attributes match the current policies or rules for accessing a certain object in a nondeceptive and verifiable way.
3) *Real-time attribute Tokens,* as an encapsulation of authorized attributes, are used to evaluate the cryptographic policy across multiple components and could be authorized real timely for preventing attributes from forgery and alteration.

The advantage gained by introducing cryptography into ABAC is that it permits resource management in an open environment, e.g., public Cloud and Blockchain. Our new framework, based on ABAC, can be deployed on various practical applications, including multitenant private-preserving outsourced storage in Cloud, fast health interoperability resources for storage and retrieval of medical data, secure big data distributed processing systems, etc.

B. Standard ABAC Model

The reference architecture of NIST's distributed ABAC model [11] will lay a foundation for our subsequent research. Fig. 2 illustrates the relationship of the fundamental components required for ABAC authorization services (ASs). Or, to be more exact, the reference architecture of the standard ABAC model is depicted in Fig. 2. The ABAC model is composed of the following four main components.

1) *PEP* receives request from an authenticated subject, interprets and sends the request to PDP, and fulfills the request according to the decision of PDP.
2) *PDP* evaluates the access policy retrieved from PAP in terms of the attribute values obtained by querying PIP to determine whether access request should be permitted or not.
3) *PIP* provides attribute values upon receiving request from the PDP context, where the values are extracted from typically LDAP services, environment sensors, or other repositories.
4) *PAP* is used to manage (e.g., authoring, deployment, and maintenance) and retrieve policies the PDP later evaluates, and to verify policies before they are added into system.

In addition, PEP and PDP, that may be centralized or distributed, make up the *AS* to manage access workflow, decision, and enforcement. The above architecture is also composed of three repository modules and one environment perception module. These three repository modules store and manage policies,

entity attributes, and objects, respectively. The environment perception can acquire environment information from multiple sensors at runtime.

C. C-ABAC Model

In the open environment such as Cloud, the encryption technique becomes an inevitable option for protecting the valuable resources. In the light of the present situation that the existing ABAC model does not support encryption, we develop a new *C-ABAC* model by extending the NIST's ABAC model [11]. This is done through maintaining the original workflow of ABAC to ensure that the presented model is entirely consistent with the standard ABAC model.

The C-ABAC model, shown in Fig. 2, contains an encryption scheme with five cryptographic algorithms (Setup, Encrypt, PolicyGen, TokenGen, Decrypt). First of all, by using Encrypt, the object is encrypted by the data owner (or service provider), and the encrypted object is stored in the Cloud. Next, several new cryptographic mechanisms (marked out with underline in Fig. 2) are appended into the ABAC's functional components, as described above. These new mechanisms are described as follows.

1) *Policy generation unit (PGU)* relies on the PAP and PolicyGen for acquiring policy relevant to the current request, then producing cryptographic representation of this police, i.e., cryptographic policy.
2) *Token generation unit (TGU)* relies on the PIP and TokenGen for collecting attribute information and generating their corresponding verifiable attribute Tokens.
3) *Policy decision unit (PDU)* relies on the PDP to make decision over cryptographic policy from the PGU according to attribute Tokens from the TGU.
4) *Resource decryption unit (RDU)* relies on the PEP to interpret and redirect access requests to the PDU and, then, decrypt the object and implement the request in terms of the permit of the PDU.

With the help of the above four cryptographic units, the authorization of the encrypted object will be managed by the access policy. That is, the encryption can be performed by the object's owner just before the object is transmitted into the system, but the decryption is implemented by the AS (including RDU and PDU) according to the decision result of the cryptographic policy.

From a managerial perspective, the components of C-ABAC, apart from the AS, can be divided into two categories.

1) One is called "*policy center*," which consists of PGU and policy repository.
2) The other one is *attribute authority*, which consists of TGU, environment perception, and attribute repository.

Obviously, the former is responsible for policy management and the latter is for attribute management.

D. Application Instance for C-ABAC

To make better understanding of our model, we provide an example of Cloud-based outsourcing service as a direct link between the C-ABAC model and practical applications. In this scenario, the RDU is located in the client side (e.g., PDA, smartphone, and desktop PC) as a middleware, which completes certain data operations according to the permit of PDU. Each access request from RDU can activate a PDU-based session in Cloud, in which a new virtual machine is loaded from a Snapshot, and the PDU scripts automatically run in this virtual machine. Then, the PDU scripts, as the core of access decision making, will execute the following steps:

1) issue queries for cryptographic policy and attribute credentials from PGU and TGUs, respectively;
2) make a policy decision according to the acquired cryptographic policy and attribute credentials;
3) return the decisional result to the RDU, which realizes the actual resource decryption and access authorization if the decisional result is permit.

The virtual machine that accomplishes these processes forms a *security sandbox*, which provides a tightly control of resources for RDU to run in and can be deleted automatically after the session ends. In this example, the resource's provider does not need to do anything but write an effective policy. This means that our C-ABAC model can enforce a direct control over outsourcing resources in a simple and efficient way.

E. Workflow of the C-ABAC Model

In our C-ABAC model, there is no concept of user key, and the decryption key of the encrypted object is replaced by attribute Tokens. Since these Tokens are generated by different entities, the authorization will be dispersed into all attribute entities in the system. For any access request from an authenticated subject to an encrypted object, the workflow of the model is shown in Fig. 2 and described as follows.

1) After the subject asks permission to access a protected object, the RDU translates and shifts this request to PDU and waits for the decision result (steps 1 and 2).
2) On receiving the request, the PDU picks up a nonce τ and then sends it to the PGU for the cryptographic representation of policy and the TGU for the attribute Tokens over τ, respectively (step 3).
3) The PGU chooses the policy Π from policy repository and produces the corresponding cryptographic representation C_{Π}. Meanwhile, the TGU generates the attribute Token $T_{a,\tau}$ for a specified entity attribute a and the nonce τ. Finally, C_{Π} and all related Tokens $\{T_{a,\tau}\}$ are sent to the PDU (step 4).
4) The PDU makes decision on the received C_{Π} and $\{T_{a,\tau}\}$. If the entity attributes satisfy the policy Π, the PDU generates the authorized information of decryption and sends it to RDU (step 5).
5) On receiving the authorized information, the RDU uses it to decrypt the encrypted object and fulfills the operation requested by the subject (steps 6–8).

Two important mechanisms, *cryptographic policy* and *attribute Token*, are involved in the above workflow. In our system, an attribute Token is a one-time security proof for the authorized attribute issued by the TGU over a nonce τ specified by PDU. We consider the Tokens as publicly verifiable "tickets" to prove that the attribute is what they claimed to be. This ticket usually

contains the information of the issuer, attributes, issue time, as well as a tag, where the tag is a simple signature that verifies the authenticity of all other information by using the issuer's public key. In addition, we describe the cryptographic policy in Section IV.

In the standard ABAC model, the input (i.e., policy and attribute) and the output of the decision process are represented in the form of plaintext without any verification mechanism. Hence, both the decision process and the decision result are fully dependent on the trustiness of PDP. However, the C-ABAC model makes security enhancement by introducing the cryptographic policy and attribute Token. Meanwhile, the policy decision making is a cryptographic process, and the decision result is verifiable. By making such improvements, the decision process and its result are unrelated to the trustiness of PDU. The behavior of RDU is also dependent on the decision result of PDU as follows.

1) If the decision result is "permit," the output of PDU is used to decrypt the accessed object, such that the RDU can fulfill the access request on it.
2) If the decision result is "deny," the access will be denied even if RDU is untrusted or corrupted.

F. Security Threats

We assume that the protected objects in our C-ABAC model are stored into a public untrusted platform; the policy center and the attribute authorities are managed by trusted institutions, but the AS can be deployed to the client. Moreover, we assume that a malicious attacker can counterfeit and eavesdrop the communication between any two components, but he cannot disturb the operation of any functional components. In addition, the only thing that the subject have to do is to authenticate his identity when he logs-in the system. However, the user does not need to store any key for resource decryption, so it will significantly reduce the user's burdens.

As described above, one-time attribute Tokens take the place of the traditional private key for decrypting resource in our C-ABAC model. In this situation, suppose that a malicious attacker can eavesdrop a large number of ineffective or expired Tokens. The poor design of Token could damage the system security if the attacker learns the secret hidden in these Tokens and thereby counterfeits new Tokens. Thus, the unforgeability of attribute Tokens is extremely important in our model. Exactly, this kind of unforgeability contains two types.

1) *CNA* is to forge a new Token with a specified nonce after learning a lot of prior Tokens.
2) *CAA* is to forge a new Token of a specified attribute after learning a lot of prior Tokens for other attributes.

Furthermore, policy counterfeiting and tamper must be taken into consideration. Aiming at this problem, we replace the plaintext policy with cryptographic representation of policy (called cryptographic policy) in C-ABAC. We require that the cryptographic policy generated by the policy center is existentially unforgeable for any other entities. Considering that the cryptographic policy is bound with the ciphertext of object, we mainly concern about the unforgeability of the cryptographic policy against COA, i.e., to forge a valid cryptographic policy for a challenged object after learning a lot of cryptographic policies corresponding to other objects.

Another important problem is to realize the verifiability of policy decision making. It means that the decision result will be unalterable and immutable. Otherwise, for a specified policy, suppose an attacker can forge the result (permit or deny) of decision making; we call it "permission counterfeiting." Furthermore, since an effective decision making equals successful decryption in our model, the permission counterfeiting can be explicated as ciphertext's semantic security [12], which implies the indistinguishability of the decision result (permit or deny) from prior information. The attacker's prior information includes two aspects: learning expired Tokens with specified attributes and learning ciphertexts with the specified policy. As a result, we require that our C-ABAC model is secure against this attack.

III. Construction of the C-ABAC System

A. Definition of the C-ABAC System

Without loss of generality, we employ $S, O, E,$ and A to denote the set of all *subject, object, environment*, and *action*, respectively. The symbol P is also used to denote the policy center, as described in Section II-C. Therefore, the set $\mathcal{S} = \{P, S, O, E, A\}$ will be used to represent all entities in our model. By using them, the encryption scheme related to the C-ABAC model is defined formally as follows.

Definition 3.1: Given a set of entities $\mathcal{S} = \{P, S, O, E, A\}$, a C-ABAC system (C-ABAC system) consists of five algorithms.

1) *Setup*$(\mathbb{S}) \rightarrow (pk_s, sk_s)$*:* It takes as input a system parameter $\mathbb{S}$ and outputs a public/private key pair (pk_s, sk_s) for entity s in $\mathcal{S}$.
2) *Encrypt*$(pk_P, ek) \rightarrow C_{ek}$*:* It takes as inputs the public key pk_P of policy center P and a session key ek and produces the ciphertext C_{ek} of ek.
3) *PolicyGen*$(sk_P, C_{ek}, \Pi, \tau) \rightarrow C_\Pi$*:* It takes as inputs the private key sk_P of policy center P, the ciphertext C_{ek}, the access policy Π, and the nonce τ and outputs the cryptographic representation C_Π of policy Π.
4) *TokenGen*$(sk_s, a, \tau) \rightarrow T_{a,\tau}$*:* It takes as inputs the private key sk_s, an attribute a issued by the entity s and the nonce τ, and generates the attribute Token $T_{a,\tau}$ for attribute a and nonce τ.
5) *Decrypt*$(pk_{\mathcal{S}}, \{T_{a,\tau}\}, C_{ek}, C_\Pi) \rightarrow ek$*:* It takes as inputs the public keys $pk_{\mathcal{S}}$ of all entities in $\mathcal{S}$, a set of Tokens $\{T_{a,\tau}\}$, the ciphertext C_{ek}, and the cryptographic representation C_Π of policy Π, and retrieves the session key ek hidden in C_{ek}.

Let $\{pk_s, sk_s\}_{s\in\mathcal{S}}$ be the public–private key pairs for each entity $s \in \mathcal{S}$, where $(pk_s, sk_s) \leftarrow \text{Setup}(\mathbb{S})$. Given any ciphertext $C_{ek} \leftarrow \text{Encrypt}(pk_P, ek)$ of the session key ek and the corresponding ciphertext $C_\Pi \leftarrow \text{PolicyGen}(sk_P, C_{ek}, \Pi, \tau)$ for policy Π and nonce τ, we say that a C-ABAC system is correct if for all issued Tokens $\{T_{a,\tau} \leftarrow \text{TokenGen}(sk_s, a, \tau)\}_{\forall a\in W}$ over a set W of authorized attributes, the session key ek can be

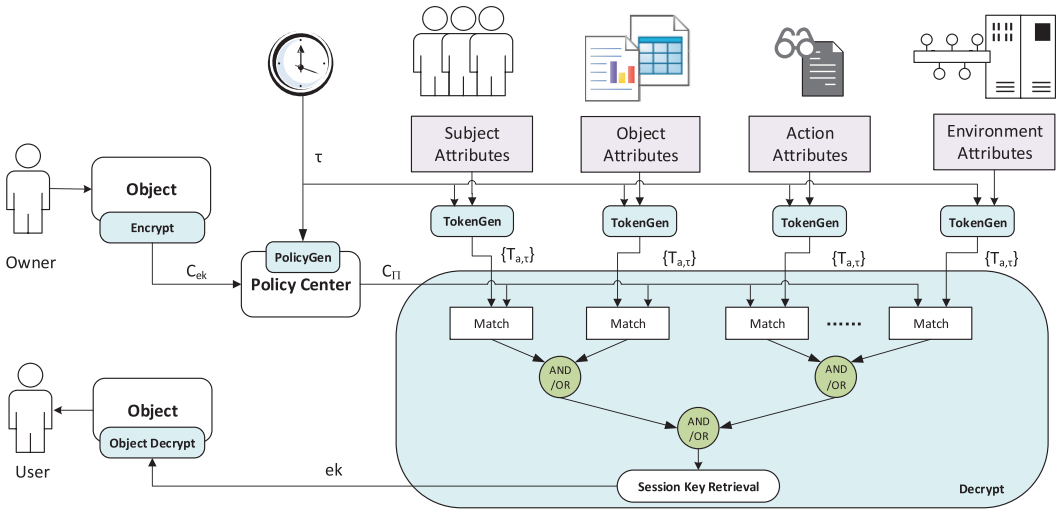

Fig. 3. Design diagram of the C-ABAC system.

retrieved with successful probability 1 according to

$$\Pr\left[\begin{array}{c}\text{Decrypt}(pk_{\mathcal{S}}, \{T_{a,\tau}\}_{a\in W}, C_{ek}, C_{\Pi}) = ek : \\ \Pi(W) = \text{True}\end{array}\right] = 1$$

where $\Pi(W)$ denotes the decision-making process for a policy Π in accordance with W, and the symbol W denotes a collection of all attributes gathered from TGU.

Compared with the existing ABE [13], our C-ABAC system has gone away from the traditional encryption framework, including the following.

1) The TokenGen algorithm is used to produce one-time attribute Tokens, instead of the user's decryption key, for dynamic authorization of resource access.
2) The PolicyGen algorithm is used to produce the cryptographic representation of dynamic policy, which is bound to the ciphertext of object.

Thus, C-ABAC is considered as a new kind of encryption framework supporting diversified attributes and dynamic policies, simultaneously.

B. Construction of C-ABAC

Let us turn our attention to practical C-ABAC construction, which is built on the general bilinear map [14] group system $\mathbb{S} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e(\cdot,\cdot))$ of prime order p. In addition, we introduce two operations into our system.

1) Cryptographic Hash function $H: \{0,1\}^n \to \mathbb{G}_2$, which maps an arbitrary string to a random element in $\mathbb{G}_2$ in order to deal with the attributes in cryptography.
2) The symbol $\oplus$ denotes XOR operation between two binary strings, in which the cycle filling method is used to pad the high bits of the short string when the string lengths are not same.

In Fig. 3, we describe the interactions among four attribute entities and the policy center, as well as the information transfers among the algorithms, as follows.

1) For a given object o, the object's owner uses the public key pk_P of the policy center P to produce the ciphertext C_{ek} for a random session key ek by invoking the Encrypt algorithm, and then, the key ek is used for practical object encryption on o.
2) For an access request to the object o, the PolicyGen algorithm is executed to generate C_{Π} by encapsulating policy Π into C_{ek} according to the private key sk_P of P, the policy Π extracted from PAP, the nonce τ, and the ciphertext C_{ek}.
3) For each attribute a involved in the policy Π, the corresponding entity executes the TokenGen algorithm to acquire the attribute Token $T_{a,\tau}$ of attribute a according to his secret key and the same nonce τ.
4) After all Tokens $\{T_{a,\tau}\}$ have acquired, the Decrypt algorithm makes attribute matching and logical decision between C_{Π} and $\{T_{a,\tau}\}$. The session key ek will be retrieved if the final decision is "permit," and then, the encrypted object can be decrypted.

In the above procedure, the nonce τ, chosen by the PDU, is used to achieve synchronization, that is, while the PDU sends the chosen τ to the PGU and all TGUs, only the cryptographic policy and attribute Tokens over this specified nonce are effective and can pass the verification. This nonce is generated by a clock in Fig. 3, considering that it could be calculated from the current time and its hash value.

In our C-ABAC construction, the five algorithms are described in detail as follows.

1) System Initialization: To build the system, the managers choose a bilinear map group system $\mathbb{S}$ of prime order p and two random generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$. These parameters will be published in the entire system. Then, every entity invokes the Setup algorithm (described in Algorithm 1) to produce his own public–private key pairs.

As shown in Algorithm 1, each entity is allowed to choose a random integer as the private key by himself. Specifically, we have the following.

Algorithm 1: Setup Algorithm (Setup).

Input: The bilinear map group system $\mathbb{S}$ with generators (g, h);

Output: The public–private key pairs pk_s/sk_s of entity s;

1: Choose a random $\delta \in_R \mathbb{Z}_p^*$ as the private key $sk_s = (\delta)$;
2: Compute g^δ and publish the public key $pk_s = (g, h, g^\delta)$;
3: **return** pk_s, sk_s;

Algorithm 2: Object Encryption Algorithm (Encrypt).

Input: Public key pk_P of the policy center and a session key ek;

Output: The ciphertext C_{ek} of the session key ek;

1: Choose a random $w \in_R \mathbb{Z}_p^*$;
2: Compute the ciphertext of ek as

$$C_{ek} = (c_1 = g^w, c_2 = ek \oplus e(g^\alpha, h)^w);$$

3: **return** C_{ek};

1) The policy center P chooses a random $\alpha \in_R \mathbb{Z}_p^*$ as the private key $sk_P = (\alpha)$ and publishes the corresponding public key $pk_P = (g, h, g^\alpha)$ in the form of public key certificate, e.g., X.509, PGP, and SKIP. This can also be found online.[1]
2) Each entity s in $\mathcal{S} \setminus \{P\} = \{S, O, A, E\}$ also picks a random $\beta_s \in_R \mathbb{Z}_p^*$ as $sk_s = (\beta_s)$ and publishes the public key $pk_s = (g, h, g^{\beta_s})$ as a certificate. For example, the object attribute's public key is $pk_O = (g, h, g^{\beta_O})$.

An important feature of this phase is that every entity is allowed to generate his own private key by himself. The advantage of this feature is to keep away from "key escrow," in which the private key will be generated and managed by a third party. This means that the system must have complete faith in the escrowed party. In addition, considering that the private key can be used for identity authentication, this feature also means that any entity apart from himself could not forge his identity in our system.

2) Object Encryption: The Encrypt algorithm can be executed by the object's owner. By using it, any legal user, called data holder, can migrate the data resources into the system in encrypted form to prevent unauthorized access. The above process is described in Algorithm 2.

As shown in Algorithm 2, the data holder only needs to encrypt the session key ek by using the public key $pk_P = (g, h, g^\alpha)$ of the policy center P. The subcipher c_2 can be produced by XOR operation after ek and $e(g^\alpha, h)^w$ are converted into binary form. Note that the session key ek is used to encrypt the object o by using a generic encryption algorithm. If we take the case of $AES = \{Enc, Dec\}$, in which the object o is encrypted by $C_o = Enc_{ek}(o)$, the final ciphertext is represented as $C = (C_{ek}, C_o)$.

In addition, the data holder also needs to define the object attributes of the migrated data resources into the object entity.

[1][Online]. Available: https://www.blackhat.com/presentations/bh-usa-99/EdGerck/certover.pdf

Algorithm 3: Policy Generation Algorithm (PolicyGen).

Input: The private key sk_P of policy center P, the access policy Π, the nonce τ, and the ciphertext C_{ek} of ek;

Output: The cryptographic policy C_Π of policy Π;

1: Choose $t \in_R \mathbb{Z}_p^*$ and calculate $p_0 = c_1^{1/t} = (g^w)^{1/t}$;
2: Convert the policy Π into (M, π) (see Section IV);
3: Generate a random vector $v = (t, r_2, \ldots, r_n)^T \in_R \mathbb{Z}_p^n$;
4: Set l as the number of literals in Π;
5: **for** $k = 1$ to l **do**
6: Calculate $\lambda_k = M_k \cdot v$, where M_k is the kth row of M;
7: Pick a random $\gamma_k \in_R \mathbb{Z}_p^*$ and compute

$$p_k = \begin{cases} p_{k1} = (g^{\beta_s} \cdot g^\tau)^{\gamma_k} \\ p_{k2} = (h^\alpha)^{\lambda_k} \oplus e(g^{\beta_s}, H(a_{\pi(k)})^{\gamma_k}). \end{cases}$$

8: for the kth attribute $a_{\pi(k)}$ belongs to the entity s;
9: **end for**
10: Generate the cryptographic policy for Π as

$$C_\Pi = (\Pi, (M, \pi), p_0, \{p_k\}_{k=1}^l).$$

11: **return** C_Π;

For example, "$Biology$" $\leftarrow Department(O)$ and "Tom" $\leftarrow Owner(O)$. Typically, these object attributes will be used for decision making of the cryptographic policy. This, however, is not absolutely necessary.

Obviously, the Encrypt algorithm is considered as an authorization shift process from the owner to the policy center because the session key ek is encrypted by using pk_P, such that only the policy center is able to decrypt the above ciphertext by using his private key $sk_P = (\alpha)$, that is, $ek = c_2 \oplus e(g^w, h)^\alpha$ and $o = Dec_{ek}(C_o)$.

3) Cryptographic Policy Generation: When a subject requests access to a specified object, the AS, including RDU and PDU, asks the PGU for the corresponding access policy. In response to this request, the PGU generates an access policy $\Pi := can\_access(S, O, A, E)$ by extracting the access rules from policy repository. This process can be performed on the policy repository in terms of the ABAC model. For an acquired policy Π, the PolicyGen algorithm (described in Algorithm 3) is executed by the PGU.

The PolicyGen algorithm is able to convert the policy Π into a cryptographic representation C_Π, called cryptographic policy, in order to improve the capability to resist the policy attack. It is easy to find that the subcipher $c_1 = g^w$ from C_{ek} is used to create the value p_0 by encapsulating with a random encryption exponent $t \in_R \mathbb{Z}_p^*$. The reason for this is that the created cryptographic policy C_Π will be only applicable to the current ciphertext C_{ek} and is invalid for other ciphertext.

Next, we make use of the LSSS (see Definition 4.1) to share the encryption exponent t into l values $\{\lambda_k = M_k \cdot v\}_{k=1}^l$. These values will then be concealed into $\{p_k\}_{k=1}^l$, where the kth item $p_k = (p_{k1}, p_{k2})$ corresponds to the kth literal in policy Π. Note that, for the attribute $a_{\pi(k)}$ of the kth literal belongs to the entity s, the pair (p_{k1}, p_{k2}) can encapsulate the information of

Algorithm 4: Token Generation Algorithm (TokenGen).

Input: The private key sk_s of entity s, the attribute value a, and the same nonce τ in Algorithm 3;
Output: The attribute Token $T_{a,\tau}$ of attribute a;
1: Compute the attribute Token $T_{a,\tau}$ from $sk_s = (\beta_s)$ as

$$T_{a,\tau} = H(a)^{\frac{\beta_s}{\beta_s+\tau}}$$

return $T_{a,\tau}$;

the attribute $a_{\pi(k)}$, the nonce τ, and the public key g^{β_s} of entity s. In addition, the reason of using the policy center's private key α in p_{k2} is to ensure that the PGU is only builder for the cryptographic policy.

4) Real-Time Attribute Token Generation: We use the TokenGen algorithm to describe the generation process of attribute Tokens. This kind of Token is an encapsulation of the attribute a with the nonce τ by using the secret key $sk_s = (\beta_s)$ and a cryptographic hash function H. The result of encapsulation, called attribute Token, is considered as a specific signature of attribute a under the aforementioned τ. This process is described in Algorithm 4.

The validity of Token $T_{a,\tau}$ can be verified with the equation $e(g^{\beta_s} \cdot g^{\tau}, T_{a,\tau}) = e(g^{\beta_s}, H(a))$ for anyone according to the issuer's public key pk_s. Moreover, the nonce τ, as the time-varying parameter, should be frequently modified, for example, it may be changed in an expiration period of about 5 s or shorter. The Token is disposable and immediate so that there is no need to manage and store them. In addition, the Token is just one element in group $\mathbb{G}_2$ such that the overheads of computation and communication are rather small.

Attribute Token is dynamically generated by individual authority according to the request of the PDU. For an attribute assignment query, including nonce τ, the authority first checks whether or not τ is in the validity period of current time and then acquires the attribute assignment from PIP by means of the attribute repository or the sensors in runtime environment. Furthermore, the TokenGen algorithm is invoked to generate the attribute Token, which is a simple signature for presenting the authenticity of attribute assignment. After receiving the Tokens, the PDU uses them to make cryptographic policy decision and discards them later on.

5) Object Decryption: In terms of the standard ABAC model, the final phase will be implemented by the ASs, the content of which includes the evaluation and enforcement of policy decision making. Thus, the Decrypt algorithm also consists of two stages: policy decision making and session key retrieval. More exactly, such two stages will be implemented by the PDU and the RDU, respectively. Algorithm 5 depicts the execution process in detail.

In the stage of policy decision making, the PDU checks whether the nonce τ is valid. If the check passes, it takes as inputs the cryptographic policy $C_\Pi = (\Pi, (M, \pi), p_0, \{p_k\}_{k=1}^{l})$ from the PGU and all possible Tokes $\{T_{a,\tau}\}$ from TGU and produces an authorized set $U \subseteq \{1, 2, \ldots, l\}$ and the index set $I = \{i : \pi(i) \in U\}$. Then, it evaluates the policy literals $p_k = (p_{k1}, p_{k2})$ associated with the policy Π with the authorized

Algorithm 5: Object Decryption Algorithm (Decrypt).

Input: The ciphertext C_{ek}, the ciphertext C_Π of policy Π, and all attribute Tokens $\{T_{a,\tau}\}$ related in the policy Π;
Output: The session key ek;
1: Set U as an authorized set according to $\{T_{a,\tau}\}$;
2: Set $I = \{i : \pi(i) \in U\}$;
3: **for** each k in I **do**
4: Compute $m_k = p_{k2} \oplus e(p_{k1}, T_{a_{\pi(k)},\tau})$;
5: **end for**
6: Extract (M, π) from C_Π and compute $\{\omega_i \in \mathbb{Z}_p^*\}_{i\in I}$ such that $\sum_{i\in I} \omega_i M_i = (1, 0, \ldots, 0)$ (see Section IV);
7: Compute $m = \prod_{i\in I} m_i^{\omega_i}$;
8: Retrieve the session key ek by computing

$$ek = c_2 \oplus e(p_0, m);$$

9: **return** ek;

attributes Tokes $T_{a_{\pi(k)},\tau}$ for all $k \in I$. By using it, the secret m_k of literals p_k can be recovered by

$$\begin{aligned} m_k &= p_{k2} \oplus e(p_{k1}, T_{a_{\pi(k)},\tau}) \\ &= p_{k2} \oplus e\left((g^{\beta_s} \cdot g^{\tau})^{\gamma_k}, H(a_{\pi(k)})^{\frac{\beta_s}{\beta_s+\tau}}\right) \\ &= p_{k2} \oplus e(g^{\gamma_k}, H(a_{\pi(k)})^{\beta_s}) \\ &= (h^{\alpha})^{\lambda_k} \oplus e(g^{\beta_s}, H(a_{\pi(k)}))^{\gamma_k} \\ &\quad \oplus e(g^{\gamma_k}, H(a_{\pi(k)})^{\beta_s}) \\ &= (h^{\alpha})^{\lambda_k}. \end{aligned} \tag{1}$$

In terms of M, the PDU computes a set of constants $\{w_i \in \mathbb{Z}_p\}_{i\in I}$ according to $\sum_{i\in I} \omega_i M_i = (1, 0, \ldots, 0)$ (see Section IV). With all values $\{m_i\}_{i\in I}$, the main secret m can be obtained as follows:

$$m = \prod_{i\in I} m_i^{\omega_i} = (h^{\alpha})^{\sum_{i\in I} \omega_i \lambda_i} = h^{\alpha t} \tag{2}$$

where m could be considered as an authorized information of policy decision making. Then, the value m is sent to the RDU in a secure way.[2]

In the stage of session key retrieval, the RDU takes as inputs the ciphertext $C_{ek} = (c_1, c_2)$, the returned value m from the PDU, as well as the value p_0 outputted by the PGU, and computes the session key ek as follows:

$$\begin{aligned} ek &= c_2 \oplus e(p_0, m) = c_2 \oplus e((g^{w})^{1/t}, h^{\alpha t}) \\ &= ek \oplus e(g^{\alpha}, h)^{w} \oplus e(g^{w}, h^{\alpha}). \end{aligned} \tag{3}$$

Finally, the object $o = Dec_{ek}(C_o)$ can be decrypted by the RDU for a valid ek. If so, the decrypted object o can be allowed to enforce the operations requested by the subject.

[2]The elliptic curve ElGamal system can be implemented for encrypting m. Let $pk = h^{\gamma}$ and $sk = \gamma \in \mathbb{Z}_p^*$ be the public and private keys of the RDU, respectively. The PDU picks up $\delta \in \mathbb{Z}_p$ at random, and the ciphertext ($e_1 = h^{\delta}$ and $e_2 = m \oplus (h^{\gamma})^{\delta}$) is returned to the RDU. At least, the RDU retrieves m according to $e_2 \oplus (e_1)^{\gamma} = m \oplus (h^{\gamma})^{\delta} \oplus (h^{\delta})^{\gamma} = m$.

IV. Cryptographic Access Policies

The most attractive area of the ABAC model is the design and implementation of access policies. The C-ABAC model also has the similar requirement of implementing the representation and evaluation of policies by means of cryptographic techniques. Hence, the translation from XACML-type language to the cryptographic policy is an important issue that must be solved. In this section, we will provide an effective solution for the aforementioned translation, as well as an evaluation method for the translated cryptographic policies.

In our solution, the XACML-type policies are first translated to the Boolean statement represented by a formula involving "AND ($\wedge$)" and "OR ($\vee$)" operators over a set of attribute rules (also called literal). Each rule involves one entity's attribute name and a constant, such as "*Role(S) = Doctor*" (grants access if the subject S is a doctor) or "*Date(E) = Weekday*" (grants access if the current date captured from environment E is Weekday), and yields a true or false result. Second, the access policy Π will be translated to a share-generating matrix M and a permutation function π in C-ABAC. This procedure is divided into two phases.

1) Translate the access policy Π to a policy tree.
2) Translate the policy tree to (M, π).

After that, the policy center can make use of (M, π) as input to integrate the policy Π into the cryptographic representation C_Π by using the PolicyGen algorithm. In order to explain (M, π) clearly, we introduce the LSSS [15], whose definition is given as follows.

Definition 4.1 (LSSS): Given a set of participants $\mathcal{P} = \{P_1, \ldots, P_l\}$, the scheme is called linear secret sharing for a policy Π in $\mathbb{Z}_p$, if it satisfies the following.

1) π is a function that defines the attribute labeling row i as $\pi(i)$.
2) M is an $l \times n$ share-generating matrix for a policy Π.
3) t denotes a random secret shared over $\mathbb{Z}_p$.
4) $share$ is a function that uses $\lambda = Mv$ to generate the vector λ, including l shares of the secret t, corresponding to l participants $P_1, \ldots, P_l$ in terms of Π, where $v = (t, r_2, \ldots, r_n)$ is a column vector and $r_2, \ldots, r_n$ are selected at random from $\mathbb{Z}_p$.
5) $reconstruction$ is a function that retrieves the secret t from any authorized set U by computing $\sum_{i \in I} \omega_i \lambda_i = t$ for some known constants $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$, where $I = \{i : \pi(i) \in U\}$ is an index set and $\lambda_i = (Mv)_i$ denotes the ith element of the vector λ.

The function π of the above definition is used to map each row of M to an attribute, i.e., the share $\lambda_i = (Mv)_i$ belongs to party $P_{\pi(i)}$ for all $i = 1, \ldots, l$. We next show the reconstruction property of the LSSS as follows: Define $I \subseteq \{1, 2, \ldots, l\}$ as $I = \{i : \pi(i) \in U\}$ for any authorized set U over attributes. If the shares $\{\lambda_i = (Mv)_i = M_i \cdot v : i \in I\}$ are held by the user, then the user can compute the reconstruction constants $\{\omega_i \in \mathbb{Z}_p : \forall i \in I\}$; thus, we have $\sum_{i \in I} \omega_i M_i = (1, 0, \ldots, 0)$. Based on it, the secret t is computed by using the equation

$$\sum_{i \in I} \omega_i \lambda_i = \sum_{i \in I} \omega_i (M_i \cdot v) = \sum_{i \in I} (\omega_i M_i) \cdot v = t$$

```
<Policy ID="Policy1" rule-method="Permit-overrides">
    <Target combine-method="one-and-only">
        Subject : Role          : Doctor
        Subject : Role          : Nurse
        Object  : ObjectName    : Ward Records
        Action  : ActionID      : Read
    </Target>
    <Rule ID="rule1" Effect"Permit">
        Subject : Role          : Doctor
    </Rule>
    <Rule ID="rule2" Effect="Permit">
        <Condition function=and>
            Subject      : Role   : Nurse
            Environment  : Time   : Weekday
        </Condition>
    </Rule>
</Policy>
```

Fig. 4. Example of access policy for Ward Records.

where all constants ω_i are public and can be found within a polynomial time over the size of M. Finally, the above process will be used to decrypt the ciphertexts in the Decrypt algorithm.

A. C-ABAC's Policies and Rules

Using the XACML as reference, we propose a reference architecture of the C-ABAC policy repository based on a lightweight policy specification language. Following mentioned symbols in Section III, the set of *subject, object, environment*, and *action* is denoted as the symbols S, O, E, and A, respectively. We also specify the attribute instance as a key–value pair, where the key denotes an attribute name that expresses the property or characteristic related to a certain entity in $\{S, O, E, A\}$. As an example of medical field, $ResourceID(O) = MedicalRecords$ expresses the attribute $ResourceID$ related to the object O is $MedicalRecords$. The other examples like this include $Role(S) = Nurse$, $Time(E) = 8{:}35$, and so on.

As shown in Fig. 4, the policy in our architecture consists of one set of rules and one target. The latter is used for policy discovery through a collection of conditions, which must be satisfied for a given request. These conditions are represented as the form of "attribute-category : attribute-name : attribute-value," which denotes an attribute instance as defined above, e.g., in Fig. 4 "Action : ActionID : Read" is the same as $ActionID(A) = Read$. For multiple conditions in one target, we employ a logical combination approach to reconcile these individual conditions, e.g., "one-and-only" is used to confine that each attribute of access request must meet at least one condition in the target. The "Target" in Fig. 4 expresses that "either a Doctor or a Nurse is allowed to Read the Ward Records," which can be translated into a logical expression

$$target(S, O, A) := ActionID(A) = Read$$
$$\text{AND } ObjectName(O) = WardRecords$$
$$\text{AND } Role(S) \in \{Nurse, Doctor\}.$$

Note that $:=$ is the definition symbol.

Besides the target, the rule also expresses a logical expression on conditions and produces an authorized effect, the value of which allows either "Permit" or "Deny." For example, in Fig. 4, the first rule denotes $rule1(S) := Role(S) = Doctor$, and the second rule is $rule2(S, E) := Role(S)$

$= Nurse$ AND $Time(E) = Weekday$. At last, several general combining methods are applied for integrating multiple rules together. These methods include the following.[3]

1) Deny overrides: a deny or indeterminate rule can override all the permit rules, and the decision is permit only when all conditions evaluate to permit.
2) Permit overrides: the final decision result is given as permit if one of the conditions evaluates to permit.
3) Supermajority rule: the decision result is permit if more than two-third conditions evaluate to permit.

The above example employs the "permit overrides" for combining rule1 and rule2 together, i.e.,

$$can\_access(S, O, E, A) := target(S, O, A)$$
$$\text{AND } (rule1(S) \text{ OR } rule2(S, E)).$$

This indicates that the access request is allowed if one of two rules, $rule1$ and $rule2$, returns true.

B. Cryptographic Representation of Policies

A structured approach is used to realize the translation from the policy represented by the Boolean formula into (M, π). The bridge of this translation is the policy tree. By using (M, π), the cryptographic policy can be produced dynamically by the PolicyGen algorithm.

For clarity, we give a concrete example for the aforementioned procedure in C-ABAC. Suppose our scenario works in the medical diagnostic recording system (MDRS). In this system, there are much sensitive data, such as patient logs, pharmacy materials, ward records, etc. To enforce authorized access, the policies are specified in the MDRS, e.g., a simple policy (described above) is defined as follows:

$$can\_access(S, O, A, E) := (Role(S) = Doctor \text{ OR}$$
$$(Role(S) = Nurse \text{ AND } Time(E) = Weekday)) \text{ AND}$$
$$ObjectName(O) = Ward\ Records \text{ AND}$$
$$ActionID(A) = Read.$$

As mentioned above, we make use of attribute name to define the *attribute Extractor* for a given entity in $\{S, O, E, A\}$, e.g., $Role(S)$ may be *Nurse* or *Doctor* for a subject S. In addition, we use the symbol P_i to name each predicate in policy from left to right, e.g., P_1 denotes "*Role(S) = Doctor*," P_2 denotes "*Role(S) = Nurse*," and so on.

At first, the access policy can be translated into a policy tree, which is shown in Fig. 5. In the policy tree, the leaf node must be a simple rule over attributes, and the inner node represents a Boolean (AND or OR) operator. The content written in each leaf node is the corresponding name (i.e., $P_1, P_2, \ldots, P_5$).

Next, a random value will be assigned to each (leaf and inner) node of the policy tree. The secret t, chosen by the policy center (see PolicyGen), is assigned to the root node. Then, the shares

[3]There exist different combination algorithms, e.g., first-applicable and only-one-applicable, for various requirements. We only take common combination algorithms as example to describe the translation from policy language to the cryptographic policy.

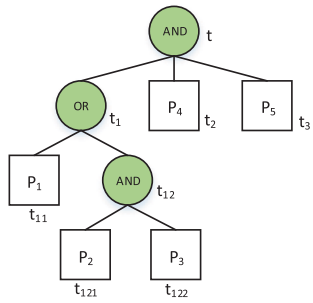

Fig. 5. Policy tree of the given access policy.

of parent are specified to its child nodes, e.g., let t_1, t_2, and t_3 be three shares of t, and they are assigned to three children of the root node, respectively. We repeat this process until all nodes are assigned with shares, e.g., the shares t_{121} and t_{122} of t_{12} are assigned to the leaf nodes P_2 and P_3, respectively. In Fig. 5, these values are marked outside each node.

Then, we pick five random elements s_1, s_2, s_3, s_4, and s_5 to produce the share-generating matrix M for the policy, as described above. Considering an (n, n)-threshold access structure is the same as AND gates, the shares of t are generated by the Shamir's threshold secret sharing scheme as follows: let $f(x) = t + r_2x + r_3x^2$ denote a random polynomial of degree 2, so that we have $t_1 = f(s_1)$, $t_2 = f(s_4)$ and $t_3 = f(s_5)$. Next, for an OR gate, we simple define $t_{11} = t_{12} = t_1$. At last, we build a polynomial $g(x) = t_{12} + r_4x$ to yield the shares of t_{12}, i.e., $t_{121} = g(s_2) = f(s_1) + r_4s_2 = t + r_2s_1 + r_3s_1^2 + r_4s_2$ and $t_{122} = g(s_3) = f(s_1) + r_4s_3 = t + r_2s_1 + r_3s_1^2 + r_4s_3$. According to the values s_1, s_2, s_3, s_4, s_5 and the above equations, the share-generation matrix M can be generated as follows:

$$M = \begin{pmatrix} 1 & s_1 & s_1^2 & 0 \\ 1 & s_1 & s_1^2 & s_2 \\ 1 & s_1 & s_1^2 & s_3 \\ 1 & s_4 & s_4^2 & 0 \\ 1 & s_5 & s_5^2 & 0 \end{pmatrix}. \tag{4}$$

We define the vector $v = (t, r_2, r_3, r_4)$, where $r_2, r_3,$ and r_4 are randomly chosen, as mentioned above. Let $\lambda_k = M_k \cdot v$ be the share of attribute $P_{\pi(k)}$, where M_k indicates the kth row of matrix M. It is easy to see $(\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5) = (t_{11}, t_{121}, t_{122}, t_2, t_3)$. Hence, we have

$$M \cdot v = \begin{pmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \end{pmatrix} \cdot (t, r_2, r_3, r_4)^T = (\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5)^T \tag{5}$$

and the mapping function π is

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}. \tag{6}$$

At last, the result (M, π) will be used in the PolicyGen algorithm to generate the cryptographic representation C_Π of policy Π.

C. Cryptographic Evaluation of Policies

In C-ABAC, policy evaluation is a cryptographic decision-making process for the cryptographic policy based on (M, π). To illustrate this process, we will continue the above example for the following request: *the subject "John" who is a doctor has an "Read" request for the file named "Ward Records" on Monday*. This means that three attribute assignments hold, i.e., *Role(S) = Doctor, Date(E) = Weekday, and Name(O) = "Ward Records."* These assignments ensure that the decision-making results of P_1, P_4, and P_5 are true. Hence, the authorized set is defined as $U = \{P_1, P_4, P_5\}$ in our example.

We now focus on the evaluation process in the Decrypt algorithm. According to π, we set $I = \{i : P_{\pi(i)} \in U\} = \{1, 4, 5\}$ and choose three rows M_1, M_4, and M_5 of the matrix M corresponding to P_1, P_4, and P_5 as the reconstruction matrix. Then, we see whether there exist ω_1, ω_4, and ω_5 such that

$$\omega_1 \cdot M_1 + \omega_4 \cdot M_4 + \omega_5 \cdot M_5 = (1, 0, 0, 0). \tag{7}$$

In doing so, we plug the corresponding values of M_1, M_4, and M_5 into the above equation, so that we acquire

$$\begin{aligned}(\omega_1, \omega_4, \omega_5) \cdot \begin{pmatrix} M_1 \\ M_4 \\ M_5 \end{pmatrix} &= (\omega_1, \omega_4, \omega_5) \cdot \begin{pmatrix} 1 & s_1 & s_1^2 & 0 \\ 1 & s_4 & s_4^2 & 0 \\ 1 & s_5 & s_5^2 & 0 \end{pmatrix} \\ &= (1, 0, 0, 0). \end{aligned} \tag{8}$$

As a result, the values ω_1, ω_4, and ω_5 could be produced efficiently in a polynomial time by using the Inverse of Vandermonde's Matrix, and the result is shown as follows:

$$\begin{cases} \omega_1 = \frac{s_4 s_5}{(s_1 - s_4)(s_1 - s_5)} \\ \omega_4 = \frac{s_1 s_5}{(s_4 - s_1)(s_4 - s_5)} \\ \omega_5 = \frac{s_1 s_4}{(s_5 - s_1)(s_5 - s_4)} \end{cases}. \tag{9}$$

Hence, for the shares λ_1, λ_4, and λ_5 corresponding to P_1, P_4, and P_5, the secret t can be computed according to (5) and (8) as follows:

$$\begin{aligned} \sum_{i \in I} \omega_i \lambda_i &= \omega_1 \lambda_1 + \omega_4 \lambda_4 + \omega_5 \lambda_5 \\ &= (\omega_1, \omega_4, \omega_5) \cdot (\lambda_1, \lambda_4, \lambda_5)^T \\ &= (\omega_1, \omega_4, \omega_5) \cdot \begin{pmatrix} M_1 \\ M_4 \\ M_5 \end{pmatrix} \cdot (t, r_2, r_3, r_4)^T \\ &= (1, 0, 0, 0) \cdot (t, r_2, r_3, r_4)^T \\ &= t. \end{aligned}$$

This result can be used to compute the value $h^{\alpha t}$ by using $\prod_{i \in I} m_i^{\omega_i} = (h^\alpha)^{\sum_{i \in I} \omega_i \lambda_i} = h^{\alpha t}$ for given values $\{m_i = (h^\alpha)^{\lambda_i}\}_{i \in I}$. Furthermore, the session key ek will be retrieved from $h^{\alpha t}$, and the encrypted file "Ward Records" is decrypted by using ek. At last, the doctor "John" is able to read this file.

V. Security Analysis

A. Security Assumptions

Four common complexity assumptions, including CDH, SDH, eBDH, and eGDHE, are used to ensure the security of the C-ABAC scheme. First of all, considering a cyclic group $\mathbb{G}$ of order p and a randomly chosen generator G in it, we define the CDH assumption [16] as follows.

Definition 5.1 ((ϵ, t)-CDH assumption): We say that the (ϵ, t)-CDH assumption holds if no t-time algorithm $\mathcal{A}$ has advantage at least ϵ in solving the CDH problem in $\mathbb{G}$, i.e.,

$$Adv_{\mathrm{CDH}}(\mathcal{A}) = \Pr[\mathcal{A}(G, G^x, G^y) = G^{xy}] < \epsilon$$

where the probability is over random choices of x, y in $\mathbb{Z}_p^*$.

In our C-ABAC scheme, the unforgeability of attribute Token will be ensured under the SDH assumption [17], which is defined as follows.

Definition 5.2 ((ϵ, t, q)-SDH assumption): We say that the (ϵ, t, q)-SDH assumption holds if no t-time algorithm $\mathcal{A}$ has advantage at least ϵ in solving the q-SDH problem in $\mathbb{G}$, i.e.,

$$Adv_{\mathrm{q-SDH}}(\mathcal{A}) = \Pr\left[\mathcal{A}(G, \{G^{(x^i)}\}_{i=1}^q) = \langle a, G^{\frac{1}{x+a}} \rangle\right] < \epsilon$$

where the probability is over the random choice of x in $\mathbb{Z}_p^*$.

The existential policy unforgeability against COA will be proved by using the following eBDH assumption.

Definition 5.3 ((ϵ, t)-eBDH assumption): We say that the (ϵ, t)-eBDH assumption holds in $\mathbb{S}$, if no t-time algorithm $\mathcal{A}$ has advantage at least ϵ in solving the decisional eBDH problem in $\mathbb{S}$, i.e.,

$$Adv_{\mathrm{eBDH}}(\mathcal{A}) = \left| \begin{matrix} \Pr\left[\mathcal{A}(\mathcal{J}, T) = 1 : T \in_R \mathbb{G}_T\right] - \\ \Pr\left[\mathcal{A}(\mathcal{J}, T) = 1 : T = e(G, H)^{ab}\right] \end{matrix} \right| < \epsilon$$

where $\mathcal{J} = (G, H, G^a, G^b, G^{1/c}, H^{ac})$, and $a, b, c \in \mathbb{Z}_p^*$.

Finally, a new complexity problem, called the eGDHE problem, is proposed to prove the semantic security of the C-ABAC scheme on the bilinear map group system $\mathbb{S}$. This new problem is constructed and defined by extending the general Diffie–Hellman exponent problem [18], as follows.

Definition 5.4 ((ϵ, t, n)-eGDHE assumption): Let $\beta, \gamma \in \mathbb{Z}_p^*$ be two secret random variables. The polynomial $f(X) = \prod_{i=1}^n (X + \tau_i)$ is constructed on the known set $\{\tau_i\}_{i=1}^n$, and $\tau \notin \{\tau_i\}_{i=1}^n$ is a known constant (i.e., $(X + \tau) \nmid f(X)$). We say that the (ϵ, t, n)-eGDHE assumption holds if no t-time algorithm $\mathcal{A}$ has advantage at least ϵ in solving the eGDHE problem in $\mathbb{S} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e(\cdot, \cdot))$, i.e.,

$$\begin{aligned} &Adv_{\mathrm{eGDHE}}(\mathcal{A}) \\ &= \left| \begin{matrix} \Pr\left[\begin{matrix} \mathcal{A}(\mathcal{L}, T) = 1 : \\ T \in_R \mathbb{G}_T \end{matrix}\right] - \\ \Pr\left[\begin{matrix} \mathcal{A}(\mathcal{L}, T) = 1 : \\ T = e(G, H)^{\beta\gamma f(\beta)} \end{matrix}\right] \end{matrix} \right| < \epsilon \end{aligned}$$

where $\mathcal{L} = (G, H, G^\beta, \{H^{\beta^i}\}_{i=1}^n, (\tau, G^{(\beta+\tau)\gamma}))$; G and H are the generators of groups $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.

By using the following theorem, we can prove that the eGDHE problem is secure for our scheme on a cyclic group of order p for a large prime p.

Theorem 5.5: For any algorithm $\mathcal{A}$ that makes a total of at most q queries to the oracles computing the group operation and the bilinear pairing, the advantage of $\mathcal{A}$ breaking a given eGDHE instant is $Adv_{\mathrm{eGDHE}}(\mathcal{A}) \leq \frac{(q+2s+2)^2 \cdot d}{2p}$, where $s = n + 4$ and $d = 2n + 2$.

B. Security Analysis

Multiple authorities, as far as federal autonomy and decentralized administration are concerned, will be an important mechanism for the identification and authentication of users in their own area. This is beneficial for the promotion of interregional cooperation in the distributed ABAC system. In such a setting, each user should be authenticated before entering the system, and the authentication methods are various and chosen on demand, e.g., smart card, PIN code, and credential biometric (fingerprint, iris, etc.). This kind of authentication is unrelated to the C-ABAC system, and it is unnecessary for the user's private key.

As far as the general cryptosystem is concerned, the semantic security [19] is a standard notion to explain that the ciphertext cannot reveal any information to the adversary, and the security of decryption key will not be taken into account. In contrast, our C-ABAC scheme is utterly different from the general cryptosystem because the decryption key has been replaced by a temporary attribute Token. In this setting, it might bring a new risk as a result that the adversary can get a great deal of expired Tokens, so as to learn the secret hidden in these Tokens and thereby counterfeit new valid Tokens. Therefore, the security of attribute Tokens must be guaranteed and proved for our C-ABAC system at first. Beyond that, we also require that the scheme achieves semantic security for a specified policy.

The unforgeability is a typical requirement for C-ABAC system in order to ensure the security of Tokens. More precisely, this requirement can be expressed as: for a certain entity X, it is computationally infeasible for a different entity X^* to forge an attribute Token $T_{a,\tau}$ for nonce τ and attribute a that belongs to the entity X. Moreover, it is still unable to make a forgery of new Tokens even if the adversary learned a large number of existential Tokens. In terms of the different ways of forgery, the unforgeability of attribute Tokens can be divided into two categories, CAAs and CNAs, in the C-ABAC scheme. We will then analyze each of these attacks separately.

1) Unforgeability Against Chosen-Attribute Attacks: It is said that the attribute Tokens have existential unforgeability against chosen-attribute attacks (EUF-CAA) when it can assure that the attacker cannot generate new Tokens under a designated attribute even if he has learned a lot of Tokens issued previously. In other words, this means that an attacher, who picks up many Tokens for different attributes by his choice, could not forge a Token for a new attribute. The EUF-CAA property can be expressed as an interaction process between an adversary and a challenger for a certain nonce τ, where the challenger may be any entity s in $\{S, O, E, A\}$. This process is described as follows.

1) *Setup:* The challenger runs setup. It gives the adversary the public key pk_s, but keeps the private key sk_s by itself.
2) *Token queries:* The adversary issues Token queries for $\{a_i\}_{i=1}^{q}$. For each query a_i, the challenger responds by running TokenGen to generate a Token $T_{a_i,\tau}$ of a_i and the given nonce τ and sending $T_{a_i,\tau}$ to the adversary.
3) *Output:* The adversary outputs a triple $(a^*, \tau, T_{a^*,\tau})$. The adversary wins if $T_{a^*,\tau}$ is a valid Token of (a^*, τ).

The attribute Token is said to be existential (ϵ, q, t)-unforgeable if no t-time adversary $\mathcal{A}$ making at most q queries has advantage at least ϵ in the above interaction process, i.e., $Adv^{\mathrm{EUF-CAA}}_{\mathrm{C-ABAC}}(\mathcal{A}) < \epsilon$. Our scheme satisfies the following theorem.

Theorem 5.6 (EUF-CAA security): In our C-ABAC scheme, the attribute Token is existentially (ϵ, q, t)-unforgeable against CAA under the (ϵ, t')-CDH assumption, where $t' \leq t + q \cdot t_Q$ and t_Q is the time of one Token query.

2) Unforgeability Against Chosen-Nonce Attacks: It is said that the attribute Tokens have existential unforgeability against chosen-nonce attacks (EUF-CNA) when it can assure that the attacker cannot generate new Tokens under a designated nonce even if he has learned a lot of Tokens issued previously. In other words, this means that an attacher, who picks up many Tokens for a set of nonces by his choice, could not forge a Token for new nonce. The EUF-CNA property can be expressed as an interaction process between an adversary and a challenger, where the challenger may be any entity s in $\{S, O, E, A\}$. This process is described as follows.

1) *Setup:* The challenger runs setup for a nonce set $\mathcal{T} = \{\tau_i\}_{i=1}^{q}$ chosen by the adversary. It gives the adversary the resulting public key pk_s, but keeps the private key sk_s by itself.
2) *Token queries:* The adversary issues Token queries for $\{a_i, \tau_i\}_{i=1}^{q}$ and $\tau_i \in \mathcal{T}$. For each query (a_i, τ_i), the challenger responds by running TokenGen to generate a Token T_{a_i,τ_i} of a_i and τ_i and sending T_{a_i,τ_i} to the adversary.
3) *Output:* The adversary outputs a triple $(a^*, \tau^*, T_{a^*,\tau^*})$. The adversary wins if T_{a^*,τ^*} is a valid Token of (a^*, τ^*) and $\tau^* \notin \mathcal{T}$.

The attribute Token is said to be existential (ϵ, q, t)-unforgeable if no t-time adversary $\mathcal{A}$ making at most q queries has advantage at least ϵ in the above interaction process, i.e., $Adv^{\mathrm{EUF-CNA}}_{\mathrm{C-ABAC}}(\mathcal{A}) < \epsilon$. Our scheme satisfies the following theorem.

Theorem 5.7 (EUF-CNA security): In our C-ABAC scheme, the attribute Token is existentially (ϵ, q, t)-unforgeable against CNA under the $(\epsilon, q+1, t')$-SDH assumption, where $t' \leq t + q \cdot t_Q$ and t_Q is the time of one Token query.

3) Policy Unforgeability Against Chosen-Object Attacks: It is said that the cryptographic policies have existential unforgeability against chosen-object attacks with policy queries (EUF-PQ-COA) when the attacker cannot generate new cryptographic policies under a designated object even if he has learned many cryptographic policies related to other objects. This kind of unforgeability can be expressed as the following interaction process.

1) *Setup:* The challenger runs setup to generate the public–private keys of each entity in $\mathcal{S}$. It gives the adversary the resulting public keys $pk_{\mathcal{S}}$.
2) *Learning:* The adversary makes at most n times policy queries for any object $o \in \{o_i\}_{i=1}^n$ and $\tau \in \{\tau_j\}_{j=1}^n$. For each policy query (Π, τ), the challenger runs the Encrypt and PolicyGen algorithms to generate C_Π and sends it to the adversary.
3) *Challenge:* The adversary declares the object o^* he wants to attack, where $o^* \notin \{o_i\}_{i=1}^n$. The challenger runs Encrypt to generate the ciphertext C_{ek^*} of session key ek^* corresponding to o^* and sends C_{ek^*} to the adversary.
4) *Output:* The adversary outputs a triple $(\Pi^*, \tau^*, C_{\Pi^*})$, where $\Pi^* \notin \{\Pi_i\}_{i=1}^n$ and $\tau^* \notin \{\tau_j\}_{j=1}^n$. The adversary wins if C_{Π^*} is valid for $(\Pi^*, \tau^*, C_{ek^*})$.

The cryptographic policy is said to be existentially (ϵ, n, t)-unforgeable if no t-time adversary $\mathcal{A}$ making at most n times policy queries has advantage at least ϵ in the above interaction process, i.e., $Adv_{\text{C-ABAC}}^{\text{EUF-PQ-COA}}(\mathcal{A}) < \epsilon$. The cryptographic policy in our scheme satisfies the following theorem.

Theorem 5.8 (EUF-PQ-COA): In our C-ABAC scheme, the cryptographic policy is existentially (ϵ, n, t)-unforgeable with at most n times policy queries against COA under the (ϵ, t)-extended bilinear Diffie–Hellman assumption.

4) Semantic Security for Designated Policies: In this subsection, the semantic security of ciphertext will be analyzed for a designated policy. The reason for this is to state clearly that the ciphertext cannot reveal the hidden information to the adversary if he does not have the ability to obtain the permit from the designated policy. In this setting, the C-ABAC system has two significant discrepancies in comparison with the normal semantic security: one is the adversary's ability of learning many attribute Tokens, and the other is of learning the ciphertext with the policy designated by the adversary.

Considering the well-known equivalence relation between indistinguishability and semantic security, we present a new notion, called the indistinguishability with Token queries against chosen-plaintext attacks, to improve the normal semantic security [12]. The purpose for proposing this new notion is that it can better meet the requirement of two above discrepancies for our C-ABAC system. Furthermore, the detailed definition of this notion can be described as follows.

1) *Setup:* The adversary declares the challenged policy Π^* and the nonce τ. For a certain entity s in $\mathcal{S}$ and policy center P, the challenger runs Setup and gives the adversary the public keys pk_s, pk_P.
2) *Learning:* The adversary makes at most n times Token queries to the TokenGen algorithm for any attribute and nonce τ_i, as well as cryptographic policy queries to the PolicyGen algorithm for any policy and nonce τ_i, where $i \in [1, n]$ and $\tau_i \neq \tau$.
3) *Challenge:* The adversary sends two chosen messages m_0 and m_1 to the challenger. The challenger randomly picks $\sigma \in \{0, 1\}$ and runs the Encrypt to produce the ciphertext C_σ of the message m_σ. Then, it invokes the PolicyGen algorithm to acquire the cryptographic representation C_{Π^*} of the policy Π^* and τ. At last, the challenger sends $C_\sigma^* = (C_\sigma, C_{\Pi^*})$ to the adversary.
4) *Response:* The adversary outputs $\sigma^* \in \{0, 1\}$ as a guess. The adversary wins the interaction process if and only if $\sigma^* = \sigma$.

We say that a C-ABAC scheme is semantically (ϵ, t, n)-secure if no t-time adversary $\mathcal{A}$ making at most n queries has advantage at least ϵ in the above game, i.e., $Adv_{\text{C-ABAC}}^{\text{IND-TQ-CPA}}(\mathcal{A}) = |\Pr[\sigma^* = \sigma] - 1/2| < \epsilon$. Our scheme satisfies the following theorem.

Theorem 5.9 (Semantic security): For a specified policy under a certain entity, our C-ABAC scheme is semantically (ϵ, n, t)-secure with n times Token queries against chosen-plaintext attack under the (ϵ, n, t')-eGDHE assumption, where $t' \leq t + n \cdot t_Q$.

The proof of theorems, including Theorem 5.5–5.9, can be found in the supplemental material.

VI. Performance Evaluation

We provide a detailed performance evaluation and analysis for our scheme based on the experimental results from our prototype system in this section. Based on these research studies, we will bring insight into the effectiveness and practicability of our C-ABAC model.

A. Complexity Analysis

There is no doubt that the complexity of C-ABAC will be increased due to the introduction of new cryptographic mechanisms, including the following.

1) The plaintext policy is substituted with the cryptographic policy in order to prevent policy counterfeiting.
2) The plaintext-based attribute assignments are replaced by attribute Tokens for avoiding attribute forgery.
3) The decision-making process of the policy is performed in a cryptographic way, which improves the trustiness of decision result.
4) The access operations would be enforced after receiving a specified authorized information and valid resource decryption.

The above security enhancements are maintained at the expense of computation and storage overheads. Exactly, the overheads of PGU and TGU increase the new overheads of PolicyGen and TokenGen algorithms besides the original overheads of PAP and PIP, respectively. Moreover, the overheads of RDU and PDU are increased considering that the Decrypt algorithm is introduced to PEP and PDP. For the sake of clarity, the performance evaluation of these algorithms is presented as follows.

The computational complexity of the C-ABAC scheme is presented in Table I. For clarity, we give several notations to express the running time of different operations in our prototype system. Let $E(\mathbb{G}_1)$, $E(\mathbb{G}_2)$, and $E(\mathbb{G}_T)$ express the exponentiation operation in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, respectively. $M(\mathbb{G}_1)$ denotes the multiplication in $\mathbb{G}_1$. B denotes the bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. It is easy to find that the computational overheads of Setup and Encrypt are constant, while the computational overheads of PolicyGen, TokenGen, and Decrypt

TABLE I
COMPUTATIONAL COMPLEXITY OF THE C-ABAC SCHEME

	Computational complexity
Setup	$E(\mathbb{G}_1)$ (each entity)
Encrypt	$E(\mathbb{G}_1) + B + E(\mathbb{G}_T)$
PolicyGen	$E(\mathbb{G}_1) + l \cdot (3E(\mathbb{G}_1) + M(\mathbb{G}_1) + 3E(\mathbb{G}_2) + B)$
TokenGen	$E(\mathbb{G}_2)$ (each attribute)
Decrypt	$\|I\| \cdot (B + E(\mathbb{G}_2)) + B$

TABLE II
STORAGE COMPLEXITY OF THE C-ABAC SCHEME

	Storage/communication complexity	Bytes ($\kappa = 80$ bits)
pk_s	$l_{\mathbb{G}_1}$ (each entity)	40
sk_s	$l_{\mathbb{Z}_p^*}$ (each entity)	20
C_{ek}	$l_{\mathbb{G}_T} + l_{\mathbb{G}_1}$	120
C_Π	$l_{\mathbb{G}_1} + l \cdot (l_{\mathbb{G}_1} + l_{\mathbb{G}_T})$	40+120 l
$T_{a,\tau}$	$l_{\mathbb{G}_2}$ (each attribute)	40
ek	$l_{\mathbb{G}_T}$	80

are directly proportional to the number l of attributes (or literals) in the policy or the number $|I|$ of elements in the set I (see Sections III-B5 and IV-C).

The storage complexity of the C-ABAC scheme is presented in Table II. In this table, the sizes of elements in $\mathbb{Z}_p^*$, $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are denoted as $l_{\mathbb{Z}_p^*}$, $l_{\mathbb{G}_1}$, $l_{\mathbb{G}_2}$, and $l_{\mathbb{G}_T}$, respectively. We neglect the operations in $\mathbb{Z}_p^*$, XOR operation, as well as matrix multiplication operation, since they are much more efficient than exponentiation and pairing operations. As shown in Table II, the storage overheads of each entity's public–private keys and ciphertext of session key are constant and short. Moreover, the communication overheads of PolicyGen and TokenGen are directly proportional to the number l of attributes (or literals) in the policy.

To evaluate the storage/communication overheads, we take $\kappa = 80$ bits as the security parameter. In this setting, the elliptic curve domain parameters over $\mathbb{F}_q$ is defined as $|q| = 2\kappa = 160$ bits and $l_{\mathbb{Z}_p^*} = 2\kappa = 160$ bits [20], [21]. We construct the bilinear map group system from the supersingular curve $Y^2 = X^3 + X$ in $\mathbb{F}_q$, such that we have $l_{\mathbb{G}_1} = l_{\mathbb{G}_2} = 4\kappa = 320$ bits and $l_{\mathbb{G}_T} = 8\kappa = 640$ bits when the embedding degree is 2. According to these parameters, we give the exact storage overheads for the cryptographic parameters in Table II. It is easy to see that the storage complexity is rather small. In particular, the size of C_Π is merely 12 kB for $l = 100$.

B. Experimental Results

We have presented an implementation of our C-ABAC system by constructing an experimental prototype system. Our development of this system consists of two subsystems: ABAC subsystem and cryptographic subsystem over the former. Moreover, we fully utilize some existing systems in order to speed up our development.

For implementing the ABAC subsystem, we used a spring security framework[4] and its expression language spring expression language (SpEL). SpEL is a modeling language [22], like secureUML and XACML, that can be used to express ABAC access model based on rules and policies. For storing policies, we have used in-memory and static JSON file, in which each rule consists of name, description, target, and condition. At present, we have established more than 100 rules.

The implementation of the cryptographic subsystem was constructed in Java totaling almost 2000 lines of code. The cryptographic algorithms worked on the java-pairing-based cryptography library. By using this library and one Intel CPU, the pairings can be computed in about 18.5 ms based on a 224-bit MNT elliptic curve ($\kappa = 112$ bits).

We have designed an experimental policy set to evaluate our system. In this set, each policy, represented as a Boolean logic, is built on a random combination of rules over rule library. Moreover, the performance experiments were performed using a special test routine more than 500 lines of Java code. The number of attributes is defined as N in an experimental policy. In each experiment, the message was encrypted using an experimental policy that was formed by some AND and OR gates with N distinct attribute conditions, where N was changed from 1 to 20. Then, the corresponding ciphertext is decrypted successfully by providing a group of valid attribute assignments.

The results of experiments for five algorithms in the C-ABAC scheme are shown from subgraphs (a)–(e) in Fig. 6. In each subgraph, we illustrate the actual time consumption (y-axis) under the different number of attributes (from 1 to 20) involved in experimental policies (x-axis). From the experimental results, it is easy to find that the time consumptions of Setup and Encrypt are constant-size with little fluctuation, in subgraph (a) and (b), respectively. The time consumptions of PolicyGen, TokenGen, and Decrypt depend linearly on the number of attributes in the experimental policy, as shown in subgraphs (c)–(e), respectively. We finally present the total time consumption of the whole C-ABAC system (including the time consumption of ABAC subsystem) undergoing the same experiments in subgraph (f).

In our C-ABAC system, attribute Tokens are dynamically generated by multiple authorities; therefore, for a given number of Tokens, this kind of multiauthority would be beneficial for load reduction when the number of authorities increases. We have done the experiments on the whole system to test the time consumption of the TokenGen algorithm under multiple authorities (two, four, six, and eight authorities, respectively). And the experimental results, composed of four colored curves, are shown in Fig. 7. From this figure, it is easy to see that the average time of the TokenGen algorithm is decreased with the increase of the number of authorities from two to eight. Exactly, the time consumption of TokenGen is just 0.66 s for 100 attributes under eight authorities, while it is 1.47 s under two authorities. To further improve performance, a parallel or fast algorithm over an elliptic curve could be used as a better solution. To sum up, our experimental results show that the C-ABAC system is lightweight and has low complexity.

C. Comparisons With Related Work

In Table III, we compared our C-ABAC scheme with KP-ABE, CP-ABE, and MA-ABE from nine aspects. First, both

[4]https://github.com/mostafa8eltaher/AbacSpringSecurity

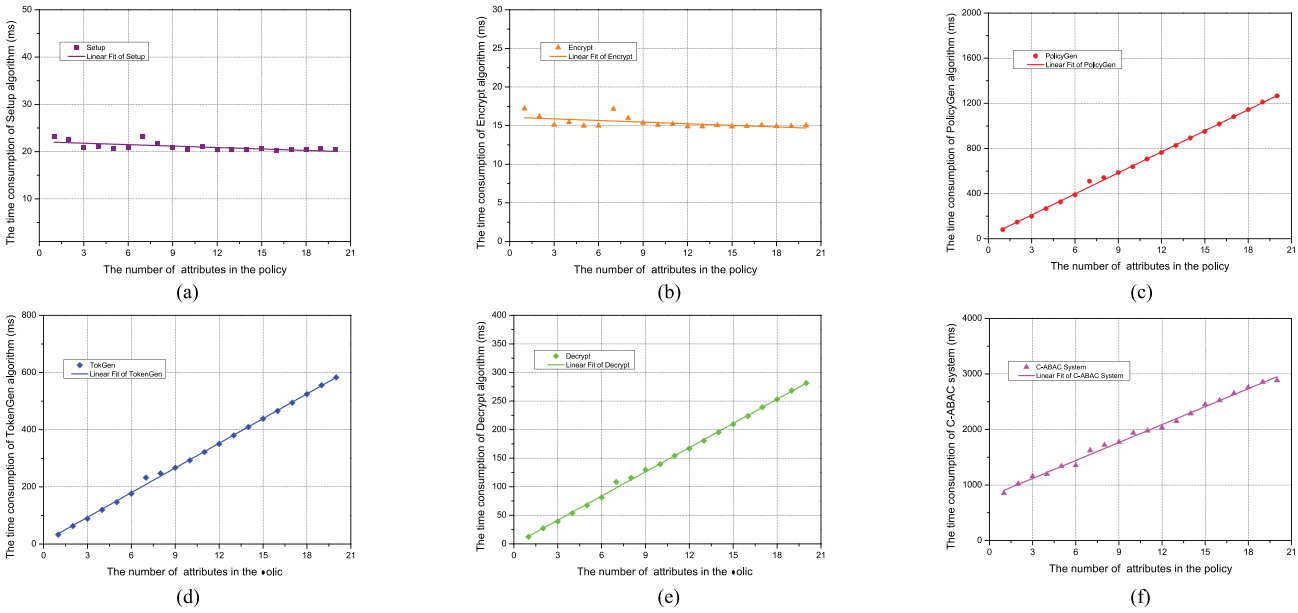

Fig. 6. Time consumption of the C-ABAC system under different number of attributes. (a) Setup algorithm. (b) Encrypt algorithm. (c) PolicyGen algorithm. (d) TokenGen algorithm. (e) Decrypt algorithm. (f) C-ABAC system.

TABLE III
COMPARISON BETWEEN EXISTING ABE SCHEMES AND C-ABAC

	CP-ABE [23]	KP-ABE [10]	MA-ABE [24]	**C-ABAC**
ABAC model	Centralized	Centralized	Distributed	**Distributed**
Attribute type	Subject, object	Subject, object	Subject, object	**Subject, object, Action, environment**
Presentation of attribute	Private key	Ciphertext	Private key	**Attribute token**
Authorization of attribute value	KDC	Data owner	Multiple authorities	**Multiple attribute entities**
Policy provider	Data owner	User manager	Data owner	**Policy center**
Generation of policy ciphertext	Encryption	Private key generation	Encryption	**After access request**
Presentation of policy	Ciphertext	Private key	Ciphertext	**Object ciphertext**
Ciphertext size	$O(n)$	$O(\|\mathbb{A}\|)$	$O(n)$	$\mathbf{O(n)}$
Private key size	$O(\|\mathbb{A}\|)$	$O(n)$	$O(\|\mathbb{A}\|)$	-

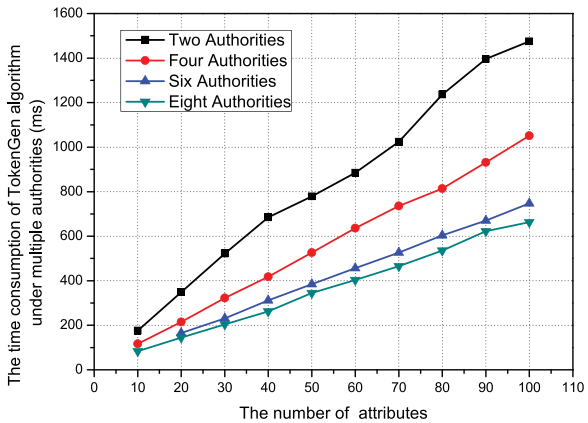

Fig. 7. Time consumption of TokenGen under different authorities.

CP-ABE and KP-ABE can only support a centralized ABAC model, while MA-ABE and our C-ABAC scheme can support the distributed ABAC model. Second, in terms of difference of supported attributes, our scheme supports four kinds of attributes, including subject, object, action, and environment attributes, while other three schemes can only support subject and object attributes. Next, in both CP-ABE and MA-ABE, the attribute credentials are bound with the user's private key, but they are bound to the ciphertext in KP-ABE. However, in our scheme, these credentials are bound into attribute Tokens, which are dynamically changed along with time. Finally, the authorization institute of attribute credentials is the key distribution center (KDC) in CP-ABE, is the data owner in KP-ABE, is multiple authorities in MA-ABE, and is multiple attribute authorities in our scheme.

In the aspect of the access policy, the policy is derived from the data owner for both CP-ABE and MA-ABE before encryption and is bound with the ciphertext. Also, the policy is designated by the user manager for KP-ABE before private key generation and is bound with the user's private key. However, the policy will be extracted from the policy center for every access request in our C-ABAC scheme and is bound with object's ciphertext at runtime. Moreover, as shown in Table III, the storage complexities of ciphertext and private key are also described for each scheme, where n is the number of literals in policy, $\mathbb{A}$ is the attribute set, and $|\mathbb{A}|$ is the number of attributes in $\mathbb{A}$. It is easy to see that the ciphertext size in CP-ABE, MA-ABE, and our C-ABAC scheme is $O(n)$, while it is $O(|\mathbb{A}|)$ in KP-ABE. The size of the private key is $O(|\mathbb{A}|)$ for both CP-ABE and MA-ABE and is $O(n)$ for KP-ABE. Especially, there does not exist the user's private key in our C-ABAC scheme; the attribute credential is associated with attribute Token, which is generated dynamically by multiple attribute authorities after the access request.

VII. Conclusion

Aiming at the security threats of existing ABAC systems in the distributed open environment, we designed a new secure ABAC model, called C-ABAC, to support provable decision making of access policies and verifiable attribute credentials. Moreover, this model can support dynamic policy extension and real-time attribute Token in comparison with existing ABEs. In addition, we developed a practical C-ABAC scheme that was proved to be secure with unforgeability of policy and Token. Finally, we give the performance evaluation of the C-ABAC scheme according to complexity analysis and experimental results.

Acknowledgment

The authors would like to thank the reviewers for their valuable comments. The authors would also like to thank Z. Zhou for his hard work on the subsequent experiment on the C-ABAC system.

References

[1] R. Sandhu, "Attribute-based access control models and beyond," in *Proc. 10th ACM Symp. Inf., Comput. Commun. Secur.*, 2015, p. 677.
[2] N. K. Sharma and A. Joshi, "Representing attribute based access control policies in owl," in *Proc. 10th IEEE Int. Conf. Semantic Comput.*, 2016, pp. 333–336.
[3] Z. Yan, G. Guohua, G. Ruiqi, and H. Dijiang, "PHE: An efficient traitor tracing and revocation for encrypted file syncing-and-sharing in cloud," *IEEE Trans. Cloud Comput.*, vol. 6, no. 4, pp. 1110–1124, Oct./Dec. 2018.
[4] K. Riad and Z. Yan, "Multi-factor synthesis decision making for trust-based access control on cloud," *Int. J. Cooperative Inform. Syst.*, vol. 26, no. 4, 2017, Art. no. 1750003.
[5] D. Ferraiolo, R. Chandramouli, R. Kuhn, and V. Hu, "Extensible access control markup language (XACML) and next generation access control (NGAC)," in *Proc. ACM Int. Workshop Attribute Based Access Control*, 2016, pp. 13–24.
[6] R. Nasim and S. Buchegger, "XACML-based access control for decentralized online social networks," in *Proc. IEEE/ACM 7th Int. Conf. Utility Cloud Comput.*, 2014, pp. 671–676.
[7] Y. Zhu, G.-J. Ahn, H. Hu, S. S. Yau, H. G. An, and C.-J. Hu, "Dynamic audit services for outsourced storages in clouds," *IEEE Trans. Services Comput.*, vol. 6, no. 2, pp. 227–238, Apr./Jun. 2011.
[8] S. Gorbunov, V. Vaikuntanathan, and H. Wee, "Attribute-based encryption for circuits," *J. ACM*, vol. 62, no. 6, 2015, Art. no. 45.
[9] K. Yang, Z. Liu, X. Jia, and X. S. Shen, "Time-domain attribute-based access control for cloud-based video content sharing: A cryptographic approach," *IEEE Trans. Multimedia*, vol. 18, no. 5, pp. 940–950, May 2016.
[10] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Secur. Privacy*, 2007, pp. 321–334.
[11] V. C. Hu *et al.*, *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*, NIST Special Publication 800-162), 2013.
[12] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Proc. Annu. Int. Cryptol. Conf.*, 2001, pp. 213–229.
[13] S. Wang, J. Zhou, J. K. Liu, J. Yu, J. Chen, and W. Xie, "An efficient file hierarchy attribute-based encryption scheme in cloud computing," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 6, pp. 1265–1277, Jun. 2016.
[14] K. Benson, H. Shacham, and B. Waters, "The k-BDH assumption family: Bilinear map cryptography from progressively weaker assumptions," in *Proc. Cryptographers Track RSA Conf.*, 2013, pp. 310–325.
[15] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman, "A flexible attribute based access control method for grid computing," *J. Grid Comput.*, vol. 7, no. 2, pp. 169–180, 2009.
[16] N. Döttling and S. Garg, "Identity-based encryption from the Diffie-Hellman assumption," in *Proc. Annu. Int. Cryptol. Conf.*, 2017, pp. 537–569.
[17] D. Boneh and X. Boyen, "Short signatures without random oracles and the SDH assumption in bilinear groups," *J. Cryptol.*, vol. 21, no. 2, pp. 149–177, 2008.
[18] D. Boneh, X. Boyen, and E.-J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2005, pp. 440–456.
[19] Y. Zhu, R. Yu, E. Chen, and D. Huang, "Dual-mode broadcast encryption," *Sci. China Inf. Sci.*, vol. 61, no. 11, 2018, Art. no. 118101.
[20] H. Hu, X. Yang, and S. Tang, "New classes of ternary bent functions from the Coulter-Matthews bent functions," *IEEE Trans. Inf. Theory*, vol. 64, no. 6, pp. 4653–4663, Jun. 2018.
[21] M. Zheng and H. G. Hu, "Cryptanalysis of prime power RSA with two private exponents," *Sci. China Inf. Sci.*, vol. 58, no. 11, pp. 1–8, 2015.
[22] S. Souissi, L. Sliman, and B. Charroux, "A novel security architecture based on multi-level rule expression language," in *Proc. Int. Conf. Hybrid Intell. Syst.*, 2016, pp. 259–269.
[23] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 89–98.
[24] M. Chase, "Multi-authority attribute based encryption," in *Proc. Theory Cryptography Conf.*, 2007, pp. 515–534.

Yan Zhu received the Ph.D. degree in computer science from Harbin Engineering University, Harbin, China, in 2005.

He is currently a Professor of Computer Science with the University of Science and Technology Beijing, Beijing, China. He was an Associate Professor with Peking University, Beijing, China, from 2007 to 2012. He was a Visiting Associate Professor with Arizona State University, from 2008 to 2009, and a Visiting Research Investigator with the University of Michigan–Dearborn in 2012. He has authored more than 100 journal and conference papers in computer and network security. His research interests include cryptography, secure computation, and access control.

Ruyun Yu received the M.S. degree in computer science and technology from the University of Science and Technology Beijing, Beijing, China, in 2015, where she is currently working toward the Ph.D. degree in software engineering.

She was a Visiting Scholar as a Joint Ph.D. Student with the University of Michigan–Dearborn from 2018 to 2019. Her research interests include group-oriented encryption and fine-grained access control.

Di Ma received the Ph.D. degree in computer science from the University of California, Irvine, CA, USA, in 2009.

She is currently an Associate Professor of Computer Science with the Computer and Information Science Department, College of Engineering and Computer Science (CECS), University of Michigan–Dearborn, Dearborn, MI, USA. She is also the Interim Associate Dean for Graduate Education and Research and the Director of the Cybersecurity Center for Education, Research, and Outreach, CECS. Her research is supported by the National Science Foundation, the National Highway Traffic Safety Administration, the Air Force Office of Scientific Research, Intel, Ford, and Research in Motion. She was with IBM Almaden Research Center in 2008 and the Institute for Infocomm Research, Singapore, from 2000 to 2005. She is broadly interested in the general area of security, privacy, and applied cryptography. Her research interests span a wide range of topics, including connected and autonomous vehicle security, smartphone and mobile device security, radio frequency identification and sensor security, and data privacy.

Dr. Ma was the recipient of the Tan Kah Kee Young Inventor Award in 2004, the Distinguished Research Award from the College of Engineering and Computer Science of the University of Michigan–Dearborn in 2017, and the 2018 Trevor O. Jones Outstanding Paper Award from SAE International.

William Cheng-Chung Chu (SM'19) received the M.S. and Ph.D. degrees in computer science from Northwestern University, Evanston, IL, USA, in 1987 and 1989, respectively.

He is currently a Distinguished Professor with the Department of Computer Science, Tunghai University, Taichung, Taiwan, where he had served as the Director of Software Engineering and Technologies Center from 2004 to 2016 and as the Dean of Research and Development office from 2004 to 2007. From 1994 to 1998, he was the Dean of Engineering College and an Associate Professor with the Department of Information Engineering and Computer Science, Feng Chia University. He was a Research Scientist with the Software Technology Center, Lockheed Missiles and Space Company, Inc. In 1992, he was also a Visiting Scholar with Stanford University. He has edited several books and authored or coauthored more than 100 referred papers and book chapters, as well as participated in many international activities, including organizing international conferences, serving as the steering committee for the IEEE Computer Society Signature Conference on Computers, Software and Applications, the Asia-Pacific Software Engineering Conference, the IEEE International Conference on Software Quality, Reliability and Security, the International Symposium on System and Software Reliability, and the program committee of more than 70 international conferences. His current research interests include software engineering artificial intelligence and big data analytics.

Prof. Chu was the recipient of special contribution awards in both 1992 and 1993 and a PIP Award in 1993 at Lockheed Missiles and Space Company, Inc. He is an Associate Editor for the IEEE TRANSACTIONS ON RELIABILITY, the *Journal of Software Maintenance and Evolution*, the *International Journal of Advancements in Computing Technology*, and the *Journal of Systems and Software*.