A Novel Access Control Model with the Skew Tent Map for Decision Making (STM-ABAC)

Omessead BenMbarak, Anis Naanaa, Sadok ElAsmi
Higher School of Communication, Tunis, Tunisia

***Abstract*—The proliferation of cloud computing exposes sensitive data to the risk of unauthorized access, as traditional access control mechanisms are often inadequate for this dynamic environment. To address these shortcomings, this article proposes a novel access control scheme, named STM-ABAC, which is based on the Skew Tent Map (STM). This scheme is specifically designed to overcome the inherent limitations of traditional Attribute-Based Access Control (ABAC) and Attribute-Based Encryption (ABE) schemes when deployed in dynamic cloud environments. The methodology involves constructing a multi-authority ABAC model, generating verifiable attribute tokens using chaotic sequences, applying LSSS-based policy encryption, and evaluating performance through rigorous formal analysis and experimental benchmarking. The results demonstrate that STM-ABAC reduces the computational overhead during decryption by up to 60% and maintains lower initialization and key-generation costs compared to existing CP-ABE and MA-ABE schemes. Furthermore, security proofs confirm strong resistance to chosen-attribute and chosen-nonce attacks.**

***Keywords*—Cloud computing; skew tent map; ABAC; ABE; attribute token**

I. Introduction

Cloud computing has become an essential foundation for modern information technology landscapes. Distributed storage architectures in the cloud are increasingly adopted by both large enterprises and individual users, offering superior adaptability, scalable storage capabilities, and universal data availability. Furthermore, their inherently distributed nature strengthens data integrity and simplifies collaborative sharing across geographically diverse users.

Despite these significant benefits, several persistent challenges remain. The Top Threats to Cloud Computing 2024 report [1], which gathered insights from over 500 industry experts, identified eleven major security concerns. The study reveals a profound evolution in cloud security risks. Traditional threats, such as denial-of-service (DoS) attacks, shared technology vulnerabilities, and data loss originating from the provider side, are perceived as less critical, reflecting increased confidence in the resilience of cloud infrastructure. Similarly, data breaches—historically the most significant concern—have dropped from the top spot. New issues have surfaced, with Identity and Access Management (IAM) [2] now recognized as the foremost challenge, surpassing concerns like misconfiguration and insufficient change control. This critical shift underscores that contemporary security risks are no longer solely the responsibility of cloud providers but are increasingly tied to user-side governance and effective access control mechanisms.

Consequently, access control serves as a foundational mechanism for guaranteeing the Confidentiality, Integrity, and Availability (CIA) triad) of information. Various access control models have been established in the literature, including Mandatory Access Control (MAC) [3], Discretionary Access Control (DAC) [4], and Role-Based Access Control (RBAC) [5]. However, these models exhibit intrinsic limitations when implemented in the highly distributed and volatile cloud environment. In contrast, Attribute-Based Access Control (ABAC) [6], [7] provides finer-grained authorization and context-aware policy evaluation. Yet, in practical deployment, ABAC often struggles with insufficient adaptability, high communication overhead between its core components (Policy Enforcement Point (PEP), Policy Decision Point (PDP), Policy Administration Point (PAP), and Policy Information Point (PIP)), and limited suitability for real-time applications—especially in vast, multimedia, or latency-sensitive systems.

To address these shortcomings, cryptographic solutions like Attribute-Based Encryption (ABE) have been introduced. Nevertheless, both Key-Policy ABE (KP-ABE) and Ciphertext-Policy ABE (CP-ABE) still face drawbacks, including susceptibility to collusion attacks, inefficient user/attribute revocation, exposure of access policies, reliance on a single trusted authority, and substantial computational demands [8], [9], [10]. To overcome these enduring limitations, this paper introduces a novel ABAC architecture that integrates chaotic dynamics—specifically, the Skew Tent Map (STM). The utilization of chaotic systems in cryptographic design has attracted increasing scholarly interest [11], [12], [13], [14], [15] in recent years, primarily due to their inherent characteristics that align closely with robust security requirements. Research in this area typically follows two main avenues: the use of chaotic sequences for data encryption and secure transmission and the exploitation of chaos to generate shared cryptography keys or secrets between authorized parties. These methods are mutually reinforcing and can be seamlessly incorporated into a cohesive security framework.

Chaotic systems are deterministic nonlinear dynamical systems that demonstrate highly intricate and seemingly random behavior under specific initial conditions. Their chaotic nature is defined by the following core characteristics:

- Extreme Sensitivity to Initial Conditions: Small modifications to the starting state of a chaotic system can lead to exponentially divergent output sequences.
- Bounded State Space: The chaotic behavior remains confined within defined parameters or states.
- Deterministic Operation: Lacking true randomness, the future state of a chaotic system can be simulated accurately given the initial conditions and parameters.

- Aperiodic Dynamics: The behavior of the system is non-repeating, meaning there is no regular pattern in its dynamics.

The remainder of this paper is structured as follows: Section II provides a review of existing work focused on enhancing Attribute-Based Access Control (ABAC) models, particularly those derived from Attribute-Based Encryption (ABE) techniques. Sections III and IV establish the theoretical underpinnings of ABAC and present the proposed Skew Tent Map-based ABAC (STM-ABAC) framework, detailing its architecture, operational flow, and core algorithms. Section V elaborates on the cryptographic mechanisms utilized to ensure secure policy conversion and the dynamic administration of attributes. Section VI furnishes a formal security analysis of the STM-ABAC scheme. Section VII outlines the experimental evaluation and discusses the performance results. Finally, Section VIII offers the conclusion and suggests avenues for future research.

II. Related Work

Traditional authorization frameworks, such as RBAC and ABAC, offer mechanisms for granular and flexible management of access permissions. However, their efficacy diminishes in vast and diverse cloud computing environments, pushing researchers to seek more advanced and integrated methodologies. Prior investigations by Smith et al. [16], Johnson et al. [17], and Brown and Miller [18] emphasize the deficiencies of these standard models, highlighting the critical need for solutions that can provide more precise and secure oversight of data access.

To surmount these constraints, encryption-based access schemes, most notably Attribute-Based Encryption (ABE), have been introduced to guarantee data confidentiality and user privacy. The primary iterations of ABE are Key-Policy ABE (KP-ABE) [19] and Ciphertext-Policy ABE (CP-ABE) [20]. In KP-ABE [21], the access structure is embedded within the user's secret decryption key, while data attributes are linked to the ciphertext. Although this scheme safeguards data secrecy, it faces considerable challenges in dynamic and large-scale policy administration, as modifying any access condition necessitates the regeneration of the key. Conversely, CP-ABE [22] embeds the access policy directly into the ciphertext and associates attributes with the user's key, affording greater adaptability to data owners. Nevertheless, this paradigm also suffers from disadvantages, including substantial computational overhead, inefficient attribute revocation, and reliance on a single central authority. This single point of failure increases the system's susceptibility to compromise and collusion risks.

Furthermore, existing ABE approaches [23] remain restricted in their ability to handle real-time and constantly changing scenarios. ABE and its variants operate using a classic cryptographic structure comprising four stages (Setup, Key Generation, Encryption, and Decryption), whereas the ABAC methodology utilizes four service modules (PEP, PDP, PAP and PIP). This fundamental structural disparity makes ABE schemes inherently ill-suited for the demands of dynamic and time-critical environments.

In addition to this incompatibility, ABE faces several key limitations:

- Inability to assign attributes dynamically: Users' secret keys remain fixed following their initial creation.
- Incompatibility with evolving policies: Access policies are fixed to the ciphertext after encryption and cannot be automatically updated to reflect changes in the system.
- Dependence on a unitary authority: All keys and policies hinge on a single central entity, creating a single point of failure and amplifying the danger of collusion or system breach.

To overcome these constraints, particularly the reliance on a single authority and the fine-grained management of attributes, the Multi-Authority ABE (MA-ABE) model was introduced [24]. This model allows trust to be distributed among multiple authorities while providing more flexible and secure access control. In [25], a MA-ABE provides fine-grained access control over encrypted data and partially addresses the problem of attribute revocation. In [26] and [27], MA-ABE is used to reduce computational and storage overhead on resource-constrained devices while preventing collusion among users. In [28], MA-ABE allows a single execution of the authorization process even when policies originate from multiple authorities, thereby improving scalability. However, despite its advantages in terms of scalability and reduced risk of key compromise, MA-ABE introduces new challenges: synchronization among authorities, increased complexity in key management, and higher latency during policy verification and user revocation. Furthermore, existing works fail to simultaneously integrate all key functionalities: multi-authority, attribute revocation, outsourced encryption, and outsourced decryption. A study in [29] sheds light on several shortcomings in different access control models, requiring solutions capable of:

- Designing an efficient ABAC model that reduces communication and computational complexity while preserving data security.
- Hiding transmitted access policies to protect the confidentiality of recipients.
- Limiting the risk of unauthorized internal access, even when users possess attributes that satisfy the policies.
- Effectively managing dynamic updates of data, attributes, and user revocation without the need to redistribute keys.

To address the aforementioned challenges, the ABAC model should be enhanced with the following four advanced security mechanisms:

- A cryptographically secured policy representation ensures protection against forgery, tampering, and replay attacks.
- Dynamically generated one-time attribute tokens used to represent real-time attribute identifiers. This approach enhances flexibility and minimizes the risk associated with key compromise.
- A cryptographically verifiable decision-making process to ensure the integrity and validity of access decisions.

- Multiple attribute authorities should be deployed to establish a decentralized authorization structure.

Despite significant advancements in ABAC, ABE, and MA-ABE models, a critical functional gap persists that existing systems fail to address simultaneously: the requirement for natively dynamic and cryptographically verifiable access management. ABE-based models, even in their multi-authority versions, are inherently static—they cannot integrate real-time attribute updates nor natively support the ABAC workflow (PEP-PDP-PAP-PIP), remaining vulnerable to collusion and policy exposure. For the first time, the STM-ABAC framework proposed in this paper implements a conceptual breakthrough by integrating the chaotic properties of the Skew Tent Map at three critical levels: the policies, the dynamic attribute tokens, and the access keys. This integration simultaneously ensures: 1) the dynamic generation of single-use attribute tokens, 2) the cryptographic verifiability of the access decision, 3) Policy Hiding, and 4) decentralized multi-authority management. It is this tripartite and dynamic integration of chaos theory that gives STM-ABAC its true novelty and distinctive advantage over conventional schemes based on bilinear cryptography.

III. Construction of the STM-ABAC System

A. Standard ABAC Model

The Attribute-Based Access Control (ABAC) model regulates access based on attributes associated with subjects, objects, and the environment. It dynamically evaluates attribute conditions to decide whether to grant or deny access. As illustrated in Fig. 1 (Appendix), the standard ABAC architecture includes four main components:

- PEP: Receives access requests, forwards them to the PDP, and enforces the returned decision.
- PDP: Evaluates policies from the PAP using attributes supplied by the PIP, and determines access outcomes.
- PIP: Provides required attribute values from external sources such as directories or sensors.
- PAP: Defines and manages access control policies evaluated by the PDP.

B. Fundamental Concepts

The authorization engine compares attribute values with policy rules, producing one of four outcomes: permit, deny, indeterminate, or not applicable [30], [31], [32].

1) Entity: An entity is defined as (S, O, E, A), where S, O, E, and A represent the sets of attributes describing the subject, object, environment, and action.

2) Attributes: Attributes represent entity properties. Each category has its subset: AT_S, AT_O, AT_E, and AT_A.

3) Policy: A policy π specifies the conditions under which subjects can perform actions on objects, formally represented as:

$$\pi = \text{sign}(AT_S, AT_O, AT_E, AT_A), \quad \text{sign} \in \{\text{permit}, \text{deny}\}.$$

The policy set is denoted by $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$.

C. Proposed STM-ABAC Model

Access control and user data protection during authentication are paramount to preventing security threats. To achieve this, integrating chaotic encryption techniques is crucial. An enhancement to the NIST ABAC model is proposed by combining it with the Skew Tent Map (STM). This approach aims to enhance security while ensuring flexibility and efficiency for dynamic access management in various contexts. Fig. 2 (Appendix) illustrates the interactions between the Attribute Authority (AA), the Policy Center (PC), and the Authorization Service (AS), as well as the information transfers between the algorithms utilized. These interactions demonstrate how the STM-ABAC model handles access requests based on encrypted policies and the defined security attributes.

The new mechanism is described as follows:

- Policy Generation Unit (PGU): This unit interfaces with the PAP and the PolicyGen Algorithm 7 (Appendix) to obtain the relevant policy for the current request. It then produces a cryptographic representation of this policy, referred to as the cryptographic policy (C_Π).
- Attribute Token Generation Unit (ATGU): This unit utilizes the PIP and AttTokGen Algorithm 8 (Appendix) to collect information on attributes and generate corresponding verifiable attribute tokens.
- Policy Decision Unit (PDU): This unit relies on the PDP to make decisions. It evaluates the cryptographic policy (C_Π) against the ATGU attribute tokens provided by the ATGU after performing Policy Decryption to verify the user's authorization status.
- Resource Encryption and Decryption Unit (REDU): This unit operates with the PEP to interpret and redirect access requests to the PDU, decrypt the object, and execute the request based on the permissions granted by the PDU.

D. Workflow of the STM-ABAC Model

The proposed Skew Tent Map-ABAC (STM-ABAC) model integrates a Skew Tent Map encryption scheme within the ABAC framework, leveraging ABE for policy enforcement. The core workflow is divided into two phases: Data Owner Preparation (Setup, PolicyGen, ResourceEncrypt) and User Access (AttTokGen, Validation, Decrypt). This structure ensures secure, fine-grained, and dynamic access control. The following workflow focuses specifically on the User Access phase, initiated upon an authenticated user's request to access a protected object (see Fig. 2) (Appendix).

- Token Generation and Request Initiation: Upon a user's request, the Request Decryption Unit (RDU) retrieves the current user attributes A. It uses the current nonce τ and the chaotic parameters b and T to locally generate the list of attribute tokens, $Tlist$, via the Attribute Token Generation Algorithm 8. The RDU then forwards the request, along with the required token and the current τ, to the Policy Decision Unit (PDU) for validation (steps 1–2).

- Token Validation and Policy Decryption: The PDU retrieves the pre-stored cryptographic policy C_Π and the user's structured secret key K_S.
    - Token Validation: The PDU first verifies the integrity and freshness of the received token against the user's current attributes via the Secure Token Validation Algorithm 9 (Appendix).
    - Policy Decryption: If the token is valid, the PDU proceeds to decrypt the policy C_Π using the user's secret key K_S via the Policy Decryption Algorithm 5 (Appendix). If the attributes embedded in K_S satisfy the LSSS (Linear Secret Sharing Scheme) policy Π, the PDU retrieves the Data Encryption Key e_k.
- Data Decryption and Resource Execution: The PDU securely transmits the retrieved data encryption key e_k to the Request Execution and Decryption Unit (REDU).
    - Chaotic Decryption: The REDU retrieves the encrypted resource C and uses the key e_k as a seed to regenerate the exact chaotic sequence K_{STM} (using the parameters b and T). It then performs the inverse operation (XOR $\oplus$) on C via the Data Decryption Algorithm 6 (Appendix) to recover the original resource data M.
    - Execution: Finally, the REDU executes the requested operation on the decrypted object M on behalf of the user (steps 6–8).

IV. Mathematical and Cryptographic Construction of STM-ABAC

This section focuses on the practical construction of the Attribute-Based Access Control (ABAC) model combined with the STM group system. The key notations used throughout the STM-ABAC construction are summarized in Table I

A. Process of STM-ABAC

The STM-ABAC process is based on the integration of ABE for fine-grained access control, and a Skew Tent Map with a parameter b for data encryption.

The following functions describe the complete workflow, from initialization to data decryption:

- Setup & KeyGen $(A) \rightarrow (K_P, K_M, K_S)$: Algorithm 1 (Appendix), the system initializes public and private parameters. The function takes the user's attributes A and master secrets $K_M = (\delta, \alpha)$ to generate the Structured Secret Key K_S, composed of attribute-specific components required for ABE decryption.
- ResourceEncrypt $(M, e_k, b, T) \rightarrow C$: Algorithm 2 (Appendix), the data owner uses a symmetric data key e_k (session key) as a chaotic seed. This key is transformed by the STM (parameters b and T) into a sequence $K_{Chaotic}$. This sequence encrypts the object M by XOR ($C = M \oplus K_{Chaotic}$), producing the ciphertext C.
- PolicyGen $(K_M, \Pi, e_k) \rightarrow C_\Pi$: Algorithm 7 (Appendix), the Policy Generation Unit (PGU) takes the master key K_M, the access policy Π, and the data key e_k. It uses Π to encrypt e_k according to ABE (LSSS) rules, producing the cryptographic representation of the policy C_Π.
- AttTokGen $(A, \tau, b, T) \rightarrow T_{list}$: Algorithm 8 (Appendix), during an access request, this function takes the current attributes A, the nonce τ, and the chaotic parameters b, T. It generates a list of chaotic, time-limited tokens T_{list} for attribute integrity evaluation.
- PolicyDecrypt $(C_\Pi, K_S) \rightarrow e_k$ or $\perp$: Algorithm 5 (Appendix), the Policy Decision/Decryption Unit (PDU/PDP) uses the user's Secret Key K_S to decrypt C_Π through bilinear pairing operations (ABE). If the attributes in K_S satisfy Π, the data key e_k is successfully recovered.
- DataDecrypt $(C, e_k, b, T) \rightarrow M$: Algorithm 6 (Appendix) Once e_k is recovered, the algorithm uses it as a seed to regenerate the chaotic sequence $K_{Chaotic}$ (with parameters b and T). This sequence is then used to decrypt the ciphertext C via XOR, restoring the original data M.

In STM-ABAC construction, the algorithms are described in detail as follows:

1) System Initialization: Initially, the system performs two main processes: the generation of entity-specific public/private keys and the creation of a structured secret key derived from these keys and the user's attributes.

a) Public and private key generation: Let $\mathbb{G}$ be a bilinear pairing group of prime order p with generators g and h. These parameters are publicly available. The Master Key (K_{MK}) of the Attribute Authority (AA) must contain the master secrets $(\delta, \alpha) \in \mathbb{Z}_p^*$ required for generating the key components. For system setup, the initial private and public keys are defined as:

$$K_M = (\delta, \alpha), \quad K_P = (g, h, g^\delta, g^\alpha)$$

b) Secret key generation: Let A denote the set of attributes associated with the user. The Secret Key K_S is generated by the AA and is structured as a set of attribute-specific components. K_S includes a main component, K_{S_0}, and a pair of components, $(K_{S_j}^{(1)}, K_{S_j}^{(2)})$, for each attribute $a_j \in A$. This ensures ABE granularity.

The generation requires the selection of a random exponent $r_j \in \mathbb{Z}_p^*$ for each attribute a_j. The final structured secret key K_S is defined as:

$$K_S = \{K_{S_0}\} \cup \left\{ \left(a_j, K_{S_j}^{(1)}, K_{S_j}^{(2)} \right) \right\}_{a_j \in A}$$

where the key components are computed using the Master Key secrets (δ, α) and the attribute hash $H(a_j)$ as follows:

TABLE I. SUMMARY OF NOTATIONS

Notation	Description
A	Set of all attributes of an entity
df	Set of all attributes of entity a including the policy center
K_p	Public key
K_M	Private key
K_S	Structured Secret Key (User's attribute keys)
τ	Nonce or time-varying parameter
b	Parameter of the Skew Tent Map
$T_{a,\tau}$	Token for a specified entity attribute a and the nonce τ
C_Π	Policy cryptographic representation
e_k	Session key

$$K_{S_0} = g^{\delta} \cdot g^{\alpha}$$

$$K_{S_j}^{(1)} = g^{r_j}$$

$$K_{S_j}^{(2)} = H(a_j)^{\frac{\delta}{\alpha}} \cdot g^{r_j}$$

The Secret Key K_S, which contains the set of attribute-linked components, is securely transmitted to the data user.

2) Chaotic resource encryption: To ensure secure data confidentiality, the system transforms the data encryption key e_k (obtained via ABE) into a chaotic key stream $K_{Chaotic}$ used to encrypt the resource $\mathbf{M}$.

a) Normalization (Algorithm 3) (Appendix): This process transforms the data encryption key e_k into a proper initial condition for the chaotic system. The key e_k is first converted into an integer. The final value X_0 (the chaotic seed) is obtained by applying a cryptographic hash function H (like SHA-256) to e_k, and normalizing the result to obtain $X_0 \in [0, 1]$.

$$X_0 = \frac{\text{hex2dec}(H(e_k))}{2^{256} - 1}$$

The normalized value X_0 is the initial condition for the Skew Tent Map (STM).

b) Application of the skew tent map transformation (Algorithm 4) (Appendix): The normalized seed X_0 is used to generate the Chaotic Key Stream $K_{Chaotic}$ by performing T iterations of the Skew Tent Map $\mathbf{T_b}(x)$, where T is the required key stream length. The STM function is defined with the control parameter b ($0 < b < 1$):

$$\mathbf{T_b}(x) = \begin{cases} \dfrac{x}{b}, & \text{if } 0 \le x < b \\ \dfrac{1-x}{1-b}, & \text{if } b \le x \le 1 \end{cases}$$

c) Resource data encryption (Algorithm 2) (Appendix): The final encryption step uses the generated Chaotic Key Stream $K_{Chaotic}$ as a one-time pad for a stream cipher. The raw resource data $\mathbf{M}$ is converted into a stream of bits/bytes, and the encryption is performed using the bitwise XOR operation ($\oplus$):

$$\mathbf{C} = \mathbf{M} \oplus K_{Chaotic}$$

The final output $\mathbf{C}$ is the encrypted resource data.

3) Policy generation (Algorithm 7) (Appendix): In the proposed STM-ABAC model, the access policy Π is transformed into a cryptographic representation C_Π to ensure its confidentiality while preserving the authorized access rights.

First, the policy is converted into a matrix representation $(\mathbf{M}, \pi)$ such that $\mathbf{M}$ satisfies the Linear Secret Sharing Scheme (LSSS) requirements and π maps each row of the matrix to the corresponding attribute. A random secret vector $\mathbf{v} = (t, r_2, \ldots, r_n)^T \in \mathbb{R}_p^n$ is generated, containing the secret share t and random values r_i, to compute the coefficients necessary for the cryptographic encoding of the policy.

For each literal $k = 1, \ldots, l$, the coefficient λ_k (the share of the secret t) is calculated as:

$$\lambda_k = M_k \cdot v$$

The corresponding encrypted component p_k is then derived using group exponentiations and multiplication (bilinear operations) to link the encryption to the associated attribute $a_{\pi(k)}$:

$$p_k = g^{\lambda_k} \cdot H(a_{\pi(k)})^{-\lambda_k}$$

where $H(\cdot)$ is a cryptographic hash function, and $a_{\pi(k)}$ represents the attribute associated with the k-th row. The central component p_0 is calculated by encrypting the data encryption key e_k (needed for the chaotic encryption) using the secret share t:

$$p_0 = e_k \cdot g^t$$

The metadata of the policy is defined as:

$$\text{policy\_metadata} = \begin{array}{l}\{\,\text{version}: \text{policy\_version},\\ \text{expiry}: \text{expiry\_time},\\ \text{attributes} = \pi(k)\,\}\end{array}$$

Finally, the complete cryptographic policy is constructed as:

$$C_\Pi = (\Pi, p_0, \{p_k\}_{k=1}^{l}, \text{sign}(\text{policy\_metadata}, K_M))$$

where $\text{sign}(\text{policy\_metadata}, K_M)$ ensures the authenticity and integrity of the policy using the Master Key K_M. This mathematical representation formalizes the PolicyGen algorithm and enhances its cryptographic precision.

4) Attribute token generation (Algorithm 8): The algorithm takes as input the set of attributes A, the nonce τ, and the two parameters of the chaotic system: the control parameter b and the number of iterations T. The nonce τ represents a timestamp indicating the precise moment of the token's generation. For each attribute value $val\_att \in A$, the process is as follows:

a) Chaotic transformation: The attribute value val_att is first normalized into a chaotic seed $N \in [0, 1]$ (using Algorithm 3) (Appendix). This seed then undergoes T iterations of the Skew Tent Map $\mathbf{T_b}(x)$ (using Algorithm 4) (Appendix) to obtain the final chaotic value T_a. This leverages the sensitivity of the chaotic system to ensure that any minute change in the input attribute radically alters the resulting value T_a.

b) Token hashing: The final chaotic value T_a is combined with the nonce τ using a cryptographic hash function H to generate the final, time-limited token $T_{a,\tau}$:

$$T_{a,\tau} = H(T_a \parallel \tau)$$

The nonce τ ensures the token is disposable, immediate, and reduces the need for storage or management. The validity of a token $T_{a,\tau}$ is verified according to the following rules by the Secure Token Validation Algorithm 9 (Appendix):

- The regenerated token $T_{a,\tau}^{\text{Generated}}$ must exactly match the presented token $T_{a,\tau}^{\text{Provided}}$ (constant_time_compare).
- The token's timestamp must be within the valid duration: $\Delta t = t_{\text{current}} - t_{\text{token}} \leq \Delta t_{\text{max}}$.
- Tokens listed in the revocation list are invalid.

If any of these conditions fail, a new token may be generated using a refresh function. Finally, the unique tokens and corresponding timestamps are generated for each entity in the dataset and used for real-time validation by the PDP.

5) Policy and data decryption phase: According to the standard ABAC model and the specific construction, the final access phase is executed by the (AS), which is responsible for implementing the policy decision and decryption processes. Specifically, these steps are carried out by the PDU and the REDU. Assuming that the token verification is successful (i.e., the PDU receives a **TRUE** after executing the Secure Token Validation Algorithm 9), the process continues with two critical decryption steps: ABE and Data Decryption.

a) Policy decryption (Algorithm 5) (Appendix): The PDU executes the Algorithm 5 (Appendix) to retrieve the Data Encryption Key (e_k) using the user's secret key K_S. The PDU takes as input the cryptographic policy C_Π (from the PGU) and the user's Structured Secret Key K_S (from the AA). The core of this algorithm is based on bilinear pairing operations to check if the set of attributes embedded in K_S satisfies the LSSS matrix $\mathbf{M}$ defined by Π. The algorithm first checks if the attributes linked to K_S can satisfy the policy Π. If not, it returns a failure symbol ($\perp$).

After that, the algorithm checks whether the attributes match a subset of policy rows I, the algorithm computes a set of constant weights $\{w_k\}_{k\in I}$ such that:

$$\sum_{k\in I} w_k \cdot M_k = (1, 0, \ldots, 0)$$

The decryption proceeds by calculating the pairing products A and B:

$$A = e(K_{S_0}, p_0), \qquad B = \prod_{k\in I} e(K_{S_k}, p_k)^{-1}$$

The session key e_k is successfully isolated and recovered by dividing A by B in the pairing target group:

$$e_k \leftarrow \frac{A}{B}$$

If e_k is successfully recovered, the PDU has granted authorization and sends the key e_k securely to the REDU.

b) Data decryption (Algorithm 6) (Appendix): The REDU executes the Algorithm 6 (Appendix) using the recovered key e_k to restore the original resource $\mathbf{M}$. The REDU takes as input the encrypted resource $\mathbf{C}$ ($ch\_encrypt$), the key e_k, and the chaotic parameters b and T.

The key e_k is used to regenerate the original chaotic seed X_0 through hashing and normalization:

$$X_0 \leftarrow H(e_k) \bmod 1$$

X_0 is iterated T times using the Skew Tent Map $\mathbf{T_b}(x)$ to precisely recreate the Chaotic Key Stream $K_{Chaotic}$.

The sequence $K_{Chaotic}$ is converted into a bit stream (S_{bits}) and used as a one-time pad to decrypt the ciphertext $\mathbf{C}$ (C_{encrypt}) via the bitwise XOR operation:

$$D_{\text{bytes}} \leftarrow C_{\text{encrypt}} \oplus S_{\text{bits}}$$

The resulting decrypted bytes are converted back into the original resource data $\mathbf{M}$ ($df\_original$), and the requested operation is executed by the REDU on behalf of the user.

V. Cryptographic Transformation of Access Policies

In the STM-ABAC model, cryptographic techniques are employed to represent and evaluate access policies. Converting XACML policies into a cryptographic form remains a key challenge.

XACML policies consist of rules that verify conditions on the attributes of subjects, objects, environments, and actions. These rules are combined through Boolean operators AND ($\wedge$) and OR ($\vee$), forming a logical policy Π. Typical examples include:

```
Role(S) = Engineer,
Date(E) = Weekday,
Object(O) = BackupFile.
```

each returning a Boolean result.

The logical policy Π is then transformed into a mathematical structure defined by a share-generation matrix M and a permutation function π, within the STM-ABAC model. This process involves two phases: translating Π into a decision tree, and converting it into the pair (M, π). The resulting pair serves as input to the `PolicyGen` algorithm, which produces the cryptographic representation of the access policy. To formalize (M, π), the model relies on the Linear Secret Sharing Scheme (LSSS).

Definition 1 (Linear Secret Sharing Scheme (LSSS))**.** *A scheme is said to be a Linear Secret Sharing Scheme (LSSS) for an access policy* Π *over a finite field* $\mathbb{Z}_p$ *and a set of participants*

$P = \{P_1, P_2, \ldots, P_l\}$ *if it satisfies the following properties:*

- π *is a mapping function that associates each row index* i *of the share-generating matrix with an attribute label* $\pi(i)$.
- M *is an* $l \times n$ *matrix used to generate shares corresponding to the access policy* Π.
- t *denotes the secret value shared over* $\mathbb{Z}_p$.
- *The function* `Share` *computes* $\lambda = M \cdot v$*, producing the share vector* $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_l)$*, where* $v = (t, r_2, \ldots, r_n)^\top$ *is a column vector and* $r_2, \ldots, r_n$ *are random elements from* $\mathbb{Z}_p$.
- *The reconstruction function* `Recons` *retrieves the secret* t *from any authorized set* U *by computing the following equation:*

$$\sum_{i \in I} \omega_i \lambda_i = t, \tag{1}$$

where $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ *are known reconstruction constants, and* $I = \{i \mid \pi(i) \in U\}$ *denotes the index set of attributes belonging to* U.

The function π associates each row of M with a specific attribute, meaning that for every index $i = 1, \ldots, l$, the share $\lambda_i = (Mv)_i$ is allocated to the party $P_{\pi(i)}$.

Define the set $I \subseteq \{1, 2, \ldots, l\}$ as $I = \{i : \pi(i) \in U\}$, where U represents any authorized subset of attributes. If the shares $\{\lambda_i = (Mv)_i = M_i \cdot v : i \in I\}$ are possessed by the user, they can calculate the reconstruction constants $\{\omega_i \in \mathbb{Z}_p : \forall i \in I\}$. Consequently, the Eq. (2) holds:

$$\sum(\omega_i M_i) = (1, 0, 0, \ldots, 0) \tag{2}$$

and therefore the secret t can be derived as illustrated in the Eq. (3):

$$\sum \omega_i \lambda_i = \sum \omega_i (M_i \cdot v) = \sum (\omega_i M_i) \cdot v = t \tag{3}$$

where all constants ω_i are public and computable in polynomial time with respect to M.

A. STM-ABAC's Policies and Rules

Drawing from the XACML framework, the STM-ABAC policy repository employs a streamlined policy specification language. An attribute instance is represented as a key-value pair, where the key denotes an attribute name corresponding to an entity in $\{S, O, E, A\}$. For example, in a resource management scenario, `ResourceID(O) = Authform` indicates that the resource ID of object O is `Authform`.

Other examples include `Role(S) = Technician` and `Time(E) = 8:30`.

The policy target is defined through conditions formatted as:

```
subject : Role : Engineer
environment : Date : Weekday
object : Type : BackupFile
```

For example, `Action : ActionID : Read` corresponds to `ActionID(A) = Read`.

As shown in the Fig. 3 (Appendix), the Target specifies that either an Engineer or a Technician is permitted to access the BackupFile:

$$\begin{aligned}\text{target}(S, O, A) := {} & \text{ActionID}(A) = \text{Read} \\ & \wedge \text{ObjectName}(O) = \text{BackupFile} \\ & \wedge \text{Role}(S) \in \{\text{Engineer}, \text{Technician}\}.\end{aligned}$$

The integration of multiple rules is achieved using the following methods:

- **Deny-overrides:** Access is only granted if all rule conditions evaluate to "Permit".
- **Permit-overrides:** Access is granted if at least one rule evaluates to "Permit".

- **Supermajority rule:** Access is granted if more than two-thirds of rule conditions evaluate to "Permit".

Based on the access control policy in Fig. 3 (Appendix), the `Permit-Overrides` method is used to combine rules:

$$can\_access(S,O,E,A) := target(S,O,A) \\ \wedge (rule1(S) \vee rule2(S,E))$$

B. Cryptographic Representation of Policies

The translation from Boolean policy representation to (M,π) is achieved via the policy tree. The cryptographic policy is dynamically generated using (M,π) and the `PolicyGen` algorithm.

Consider an IT equipment management system that stores critical information such as IP addresses, MAC addresses, and encryption keys. A robust policy framework is necessary to protect this data. An example policy is defined as:

$can\_access(S,O,A,E)$:= [(Role(S) = Engineer $\wedge$ Localisation(E) = LAC) $\vee$ (Role(S) = Technician $\wedge$ Time(E) = Weekday)] $\wedge$ ObjectName(O) = BackupFile $\wedge$ ActionID(A) = Read

Each predicate in the policy is associated with a unique identifier P_i. The policy tree, illustrated in Fig. 4 (Appendix), represents Boolean operators (AND/OR) as internal nodes and attributes as leaves. To construct the share generation matrix M, random elements e_1, e_2, e_3, e_4, e_5 are selected. Since a threshold access structure (n,n) is equivalent to an AND gate, the shares of the secret k are generated using a secret sharing scheme as follows. Consider a random polynomial of degree 2 $f(x) = k + r_2x + r_3x^2$, so that the shares k_1, k_2, and k_3 are given by $k_1 = f(e_1)$, $k_2 = f(e_2)$, and $k_3 = f(e_5)$. For an OR gate, k_{21} and k_{22} are simply defined to be equal to t_2. . Next, a polynomial $g(x) = k_{22} + r_4x$ is constructed to derive the shares of k_{22}, specifically $k_{221} = g(e_2) = f(e_2) + r_4e_3 = k + r_2e_2 + r_3e_2^2 + r_4e_3$ and $t_{222} = g(e_2) = f(e_2) + r_4e_4 = k + r_2e_2 + r_3e_2^2 + r_4e_4$. Using the values e_1, e_2, e_3, e_4, and e_5, along with the equations above, the share generation matrix M can be constructed by the Eq. (4).

$$M = \begin{pmatrix} 1 & e_1 & e_1^2 & 0 \\ 1 & e_2 & e_2^2 & 0 \\ 1 & e_2 & e_2^2 & e_3 \\ 1 & e_2 & e_2^2 & e_4 \\ 1 & e_3 & e_3^2 & 0 \end{pmatrix} \tag{4}$$

The vector $u = (k, r_2, r_3, r_4)$ is defined, where r_2, r_3, and r_4 are selected randomly, as previously described. The share $\lambda_j = M_j \cdot u$ corresponds to the attribute $P_\pi(j)$, with M_j being the j-th row of the matrix M. It is clear that the resulting share vector $(\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5)$ aligns with $(k_1, k_{21}, k_{221}, k_{222}, k_3)$. Thus, the following conclusion is drawn in the Eq. (5):

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

And the mapping function π is

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

The results of Eq. (5) and Eq. (6) will be used in the $PolicyGen$ algorithm to generate the cryptographic.

C. Cryptographic Evaluation of Policies

The cryptographic decision-making process for the cryptographic policy is based on (M,π). To illustrate this process, consider the following request: the subject $Mohammed$, who is an $Engineer$, has a $Read$ request for the file named $BackupFile$ from $Tunis$. This means that three attribute assignments are valid.

$$\begin{aligned} \text{Role}(S) &= \text{Engineer} \\ \text{Localisation}(E) &= \text{Tunis} \\ \text{Name}(O) &= \text{BackupFile} \end{aligned}$$

In the evaluation process of the $Decryptalgorithm$, the set $I = \{i : P_{\pi(i)} \in U\} = \{1, 4, 5\}$ is defined based on the permutation π. Subsequently, the three rows M_1, M_2, and M_5 are selected from the matrix M, corresponding to P_1, P_2, and P_5, to construct a reconstruction matrix. Following this, the existence of values ω_1, ω_2, and ω_5 is checked to ensure they satisfy specific conditions, as expressed in Eq. (7).

$$\omega_1 \cdot M_1 + \omega_2 \cdot M_2 + \omega_5 \cdot M_5 = (1,0,0,0). \tag{7}$$

By doing this, the corresponding values of M_1, M_2, and M_5 are substituted into the equation above, allowing the expression 8 below to be obtained.

$$\begin{gathered} (\omega_1, \omega_2, \omega_5) \cdot \begin{pmatrix} M_1 \\ M_2 \\ M_5 \end{pmatrix} = \\ (\omega_1, \omega_2, \omega_5) \cdot \begin{pmatrix} 1 & e_1 & e_{21} & 0 \\ 1 & e_2 & e_2^2 & 0 \\ 1 & e_3 & e_3^2 & 0 \end{pmatrix} = (1,0,0,0) \end{gathered} \tag{8}$$

So the secret k can be computed according to Eq. (5) and (8) as follows in the Eq. (9):

$$\begin{aligned}\sum_{i\in I}\omega_i\lambda_i &= \omega_1\lambda_1+\omega_2\lambda_2+\omega_5\lambda_5\\ &= (\omega_1,\omega_2,\omega_5)\cdot(\lambda_1,\lambda_2,\lambda_5)^T\\ &= (\omega_1,\omega_2,\omega_5)\cdot\begin{pmatrix}M_1\\M_2\\M_5\end{pmatrix}\cdot(k,r_2,r_3,r_4)^T\\ &= (1,0,0,0)\cdot(k,r_2,r_3,r_4)^T\\ &= k.\end{aligned} \tag{9}$$

This result can be used to determine the session key s_k, which will be retrieved from the policy generation algorithm.

VI. Security Proofs

A. Unforgeability Against Chosen-Attribute Attacks (EUF-CAA)

Attribute Tokens are considered *Existentially Unforgeable Against Chosen-Attribute Attacks (EUF-CAA)* if the scheme guarantees that an adversary cannot create a valid new token for a specific attribute a^*, even after observing numerous previously issued tokens for other attributes. In an ABAC system, this property is crucial against adversaries who select specific attributes to forge a valid token without possessing the associated private key.

The EUF-CAA property can be modeled as an interaction between an adversary and a challenger:

- Challenger: The entity that executes the setup protocol and securely holds the private key.
- Adversary (A): The entity attempting to forge an attribute token.
- Attributes $\{a_i\}$: The set of attributes for which the adversary requests tokens.
- Forge Condition: The adversary wins if they output a triple $(\tau, a^*, T_{a^*,\tau})$ such that $T_{a^*,\tau}$ is a valid token for the attribute a^* with a nonce τ, and a^* has not been previously queried by A.

Theorem 1 (Non-Forgeability of the Attribute Token)**.** *The attribute token scheme is considered existentially (ϵ, q, t)-unforgeable if no adversary A, making at most q queries within a computational limit t, can forge a valid token with probability greater than ϵ. Specifically, the advantage $Adv^{EUF\text{-}CAA}_{STM\text{-}ABAC}(A)$ must be less than ϵ.*

The STM-ABAC scheme ensures that each token $T_{a_i,\tau}$ is highly dependent on the private key and the attribute a_i (via the chaotic transformation). It is therefore practically infeasible for an adversary to generate a valid token $T_{a^*,\tau}$ for any new attribute a^* without the private key. Hence, the attribute token is existentially (ϵ, q, t)-unforgeable against CAA, assuming the computational advantage is bounded. The total time is $t' \preceq t + q \cdot t_Q$, where t_Q is the time required for one token query.

B. Unforgeability Against Chosen-Nonce Attacks (EUF-CNA)

Attribute tokens are *Existentially Unforgeable Against Chosen-Nonce Attacks (EUF-CNA)* if an attacker cannot create a valid token for a new nonce τ^*, even after obtaining multiple tokens for previously chosen nonces.

The EUF-CNA property can be modeled as an interaction between an adversary and a challenger (any entity from $\{S, O, E, A\}$):

- Setup Phase: The challenger prepares the system and securely holds the private key sk_s. The adversary receives the public key pk_s.
- Token Queries Phase: The adversary requests tokens for pairs $\{(a_i, \tau_i)\}_{i=1}^{q}$, where the nonces τ_i are chosen by the adversary. For each pair, the challenger generates a token T_{a_i,τ_i} using the TokenGen function and provides it to the adversary.
- Output Phase (Forge Condition): The adversary produces a triple $(\tau^*, a^*, T_{a^*,\tau^*})$. The adversary succeeds if T_{a^*,τ^*} is a valid token for (a^*, τ^*) and τ^* has not been included in the previously queried set $T = \{\tau_i\}_{i=1}^{q}$.

Theorem 2 (Unforgeability Against Chosen-Nonce Attacks)**.** *The attribute token is existentially (ϵ, q, t)-unforgeable against CNA if no adversary A, with a computational limit t and at most q queries, can produce a valid token with probability greater than ϵ. Formally, the advantage $Adv^{EUF\text{-}CNA}_{STM\text{-}ABAC}(A)$ must be less than ϵ.*

STM-ABAC scheme guarantees that attribute tokens are existentially (ϵ, q, t)-unforgeable against chosen-nonce attacks, assuming bounded computational advantage, where $t' \preceq t + q \cdot t_Q$, and t_Q is the time required for a single token query.

VII. Performance Evaluation

This section presents the performance analysis of the proposed STM-ABAC model. Compared to the schemes presented in [33], [34], [35], [36], which are based on the CP-ABE framework, the proposed model is evaluated in terms of computational efficiency and experimental analysis. A detailed comparison is also conducted between STM-ABAC and existing ABAC-based access control models.

A. Computational Efficiency

To evaluate the computational efficiency of the proposed STM-ABAC model, its computational costs are compared with those of the existing schemes [33], [34], [35], [36], as summarized in Table III (Appendix). The notations and their corresponding descriptions are presented in Table II (Appendix). It should be noted that the computational overhead of multiplication and hash operations is negligible compared to the other operations listed, and the overhead of G_1 is significantly lower than that of G_0 and C_e.

- Setup Phase: The computational cost of all schemes is primarily determined by the exponential operations in G_0 and G_1. Schemes generating parameters for each attribute or authorization incur higher overhead due to

the increased number of exponential operations. STM-ABAC requires $(n+2)G_0+G_1$, which remains lower than most existing schemes, demonstrating efficient initialization.

- Key Generation Phase: All schemes depend on exponential operations in G_0, which increase with the number of attributes in the secret key (k). Multi-authorization schemes, such as those in [35], require additional G_0 operations, resulting in higher overhead. STM-ABAC minimizes G_0 operations to $kG_0 + G_1$, reducing the computational cost during this phase.
- Encryption Phase: The cost of existing schemes [33], [35], [36] is influenced by exponential operations in both G_0 and G_1, while STM-ABAC primarily relies on G_1 operations ($|S|G_1 + LS$). Since G_1 operations are less expensive than G_0, STM-ABAC achieves lower overhead and better scalability as the number of ciphertext attributes ($|S|$) increases.
- Decryption Phase: The overhead in existing schemes varies depending on which groups (G_0, G_1) and ciphertext components (C_e) are involved. STM-ABAC performs decryption using $|S|C_e + LX$, avoiding G_0 operations and thus achieving lower computational cost compared to other schemes, while maintaining efficiency even as the number of attributes satisfying the access structure ($|S|$) grows.

Overall, STM-ABAC consistently demonstrates reduced computational overhead across all phases compared to the existing schemes, confirming its efficiency for secure and scalable attribute-based access control.

In the proposed STM-ABAC scheme, the cost of a pairing operation (C_e) is the most computationally expensive, while the cost of a XOR operation ($\mathbf{X}$) is negligible and serves as the reference, with $\mathbf{C_e} \approx 1000 \times \mathbf{X}$.

The decryption cost is therefore dominated by the pairings used during token generation, whereas data decryption relies on fast XOR operations. Formally, the total cost can be expressed as

$$\text{Cost}_{\text{STM-ABAC}} = \mathcal{O}(|\mathbb{S}| \cdot \mathbf{C_e} + L \cdot \mathbf{X}) \approx 10 \cdot \mathbf{C_e} + 1000 \cdot \mathbf{X}.$$

The data-related cost ($1000 \cdot \mathbf{X}$) is negligible compared to the cryptographic cost and can be approximated as $1 \cdot \mathbf{C_e}$. Hence, the overall decryption cost is largely dominated by token generation ($10 \cdot \mathbf{C_e}$), confirming that the chaotic STM mechanism ensures that the cost of decrypting data remains insignificant relative to the cost of cryptographic operations.

Building upon these results, the STM-ABAC model will be evolved by further optimizing token generation and reducing the pairing overhead. The planned focus is to enhance both the computational efficiency and the scalability of the system, particularly for large-scale cloud environments.

B. Experimental Analysis

To further demonstrate the computational efficiency of the proposed STM-ABAC scheme, a series of comparative experiments was conducted against the reference schemes [33], [34], [35], [36]. The experimental results evaluated the time cost of the schemes as the number of the user's attributes increased. The experiments were implemented using the Python language with the open-source cryptographic library Charm-Crypto (or another suitable Python-based pairing library). The hardware configuration featured an 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80 GHz processor running on the Windows 11 operating system.

The experimental results, illustrated collectively in Fig. 5 (Appendix), unequivocally demonstrate the computational efficiency of the proposed STM-ABAC scheme across all phases. Fig. 5a shows that the STM-ABAC model consistently achieves the lowest initialization time, maintaining a near-constant cost despite the increasing number of attributes. During the Key Generation phase (Fig. 5b) (Appendix), the STM-ABAC achieves a significant reduction in computational time (over 70% improvement compared to certain reference schemes) due to the integration of the Skew Tent Map for chaotic key generation. The Encryption time cost (Fig. 5c) (Appendix) also exhibits a lower growth rate, confirming better scalability with larger attribute sets. Finally, Fig. 5d (Appendix) highlights the model's major advantage during Decryption, where it achieves the best overall performance (up to 60% faster than comparable reference schemes), validating its suitability for secure, real-time access control in cloud environments.

C. Comparisons with Related Work

The Table IV (Appendix) compares four attribute-based access control (ABE-ABAC) technologies across five technical criteria, highlighting the limitations of existing schemes (CP-ABE, KP-ABE, MA-ABE) and the advantages of the proposed STM-ABAC model.

Regarding system architecture, CP-ABE and KP-ABE are centralized, creating a single point of failure and potential performance bottlenecks. In contrast, STM-ABAC, like MA-ABE, adopts a distributed architecture, enhancing robustness and scalability for large-scale systems such as cloud environments.

Traditional ABE schemes focus on subject (user) and object (data) attributes. Following the NIST ABAC standard, access control should also consider action (e.g., read/write) and environment (e.g., time/location). STM-ABAC is the only scheme fully implementing this model, enabling more granular and context-aware access control.

For private key management, existing models rely on static cryptographic keys. STM-ABAC introduces a dynamic mechanism where keys depend on initial conditions, parameters, and iteration counts, enhancing security through iterative key derivation.

Regarding policy provision and authorization, STM-ABAC, like CP-ABE and MA-ABE, delegates access decisions to the Data Owner while supporting a multi-authority framework, which is inherited from MA-ABE, thereby increasing flexibility and security.

In summary, STM-ABAC combines the benefits of distributed and multi-authority architectures while introducing enhanced granularity through action and environment attributes,

as well as a dynamic key mechanism, to offer more precise and secure access control.

D. Summary of Performance

This paper makes the following measurable and architectural contributions to the field:

1) High decryption efficiency: The proposed framework demonstrates a reduction in computational overhead during decryption by up to 60% compared to existing CP-ABE and MA-ABE schemes. This efficiency gain is achieved by decoupling the cryptographic policy decryption (ABE) from the final data decryption (lightweight STM XOR operation).

2) Optimized key setup and initialization phases: Lower initialization and key-generation costs are realized compared to existing schemes. Specifically, the initialization time remains nearly constant, irrespective of the number of attributes involved as shown in Fig. 5a (Appendix).

3) Enhanced anti-collusion security: A dynamic access control mechanism is integrated, which ensures strong resistance to chosen-attribute and chosen-nonce attacks (formally proven by Theorem 2).

4) Novel architectural integration: A new framework is presented that natively integrates a multi-authority access control model with policy-hiding and chaos-based encryption. This addresses a significant architectural gap left by previous ABAC and ABE models.

VIII. Conclusion

This paper introduced STM-ABAC, a chaotic-enhanced access control model that addresses the core limitations of traditional ABAC and ABE schemes in dynamic cloud environments. By integrating the Skew Tent Map, the model enables the generation of verifiable and non-forgeable attribute tokens, ensuring secure and real-time policy evaluation while significantly reducing computational overhead. STM-ABAC also incorporates an LSSS-based policy transformation mechanism and a multi-authority architecture fully compatible with ABAC workflows, thereby improving confidentiality, decentralization, and overall system efficiency. The scientific value of the model lies in the unified integration of chaos theory into both policy encryption and attribute validation, offering a resilient and scalable foundation for modern distributed infrastructures. Although reliance on bilinear pairings and the lack of large-scale deployment testing remain limitations, future work will focus on enhancing scalability, integrating machine learning–based trust management, and optimizing chaotic token generation for cloud and IoT environments.

Declaration on Generative AI

The authors declare that generative AI tools (ChatGPT, version GPT-5, by OpenAI) were used solely to assist with language refinement and formatting during the preparation of this manuscript. The use of AI was limited to improving grammar and readability. All content, including the conceptualization, analysis, algorithms, and conclusions, was developed, verified, and edited entirely by the authors, who take full responsibility for the accuracy and integrity of the paper. No generative AI tool was credited as an author.

References

[1] Cloud Security Alliance, "Top Threats to Cloud Computing 2024," [Online]. Available: https://cloudsecurityalliance.org/research/top-threats/.

[2] P. Jain, "Identity and Access Management in the Cloud," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT)*, 2025.

[3] F. Cai, N. Zhu, J. He, *et al.*, "Survey of access control models and technologies for cloud computing," *Cluster Computing*, vol. 22, no. Suppl. 3, pp. 6111–6122, 2019, doi: 10.1007/s10586-018-1850-5.

[4] W. Stallings and L. Brown, *Computer Security: Principles and Practice*, Harlow, U.K.: Pearson, 2019. [Online]. Available: https://content/one-dot-com/one-dot-com/us/en/highereducation/program.html

[5] S. V. Bezzateev, T. N. Elina, V. A. Mylnikov, and I. I. Livshitz, "Risk assessment methodology for information systems based on user behavior and IT-security incident analysis," *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, vol. 21, pp. 553–561, 2021.

[6] R. Sandhu, "Attribute-based access control models and beyond," *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pp. 677–686, 2015.

[7] N. K. Sharma and A. Joshi, "Representing attribute-based access control policies in OWL," in *IEEE Tenth International Conference on Semantic Computing*, 2016, pp. 333–336.

[8] B. Lang, J. Wang, and Y. Liu, "Achieving flexible and self-contained data protection in cloud computing," *IEEE Access*, vol. 5, pp. 1510–1523, 2017.

[9] G. S. Mahmood, D. J. Huang, and B. A. Jaleel, "A secure cloud computing system by using encryption and access control model," *Journal of Information Processing Systems*, vol. 15, no. 3, pp. 538–549, 2019.

[10] Q. Huang, Y. Yang, and M. Shen, "Secure and efficient data collaboration with hierarchical attribute-based encryption in cloud computing," *Future Generation Computer Systems*, vol. 72, pp. 239–249, Jul. 2017.

[11] K. Sixiao, L. Chunbiao, H. Shaobo, Ç. Serdar, and L. Qiang, "A memristive map with coexisting chaos and hyperchaos," *Chinese Physics B*, vol. 30, no. 11, p. 110502, 2021.

[12] N. Kuznetsov, T. Mokaev, and P. Vasilyev, "Numerical justification of Leonov conjecture on Lyapunov dimension of Rössler attractor," *Communications in Nonlinear Science and Numerical Simulation*, vol. 19, pp. 1027–1034, 2014.

[13] N. Nagaraj, "The unreasonable effectiveness of the chaotic tent map in engineering applications," *Chaos Theory and Applications*, vol. 4, no. 4, pp. 197–204, 2022.

[14] S. Zhou, X. Wang, and Y. Zhang, "Novel image encryption scheme based on chaotic signals with finite-precision error," *Information Sciences*, vol. 621, pp. 782–798, 2023.

[15] M. Frunzete, L. Yu, J.-P. Barbot, and A. Vlad, "Compressive sensing matrix designed by tent map for secure data transmission," in *Proceedings of the Signal Processing Algorithms, Architectures, Arrangements, and Applications (SPA)*, IEEE, 2011, pp. 1–6.

[16] J. Smith, A. Johnson, and M. Doe, "Enhancing Cybersecurity Through Advanced Access Control Mechanisms," *Journal of Cybersecurity*, vol. 12, no. 3, pp. 45–62, 2018.

[17] K. Johnson, S. Brown, and R. Lee, "A Comprehensive Analysis of Advanced Access Control Systems in Computer Security," *Journal of Computer Security*, vol. 20, no. 4, pp. 189–205, 2019.

[18] R. Brown, S. Miller, and T. Smith, "Fine-Grained Control: The Role of Attribute-Based Encryption in Modern Information Security," *Journal of Information Security*, vol. 18, no. 2, pp. 102–118, 2020.

[19] S. Canard, D. H. Phan, and V. C. Trinh, "Attribute-based broadcast encryption scheme for lightweight devices," *IET Information Security*, vol. 12, pp. 52–59, 2017.

[20] 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. Computer and Communications Security*, Alexandria, VA, USA, Oct. 30–Nov. 3, 2006, pp. 89–98.

[21] S. Yu, K. Ren, W. Lou, and J. Li, "Defending against key abuse attacks in KP-ABE enabled broadcast systems," in *Proc. Int. Conf. Security and Privacy in Communication Systems*, Washington, DC, USA, Oct. 21–23, 2009, Berlin/Heidelberg, Germany: Springer, pp. 311–329.

[22] Z. Zhang, J. Zhang, Y. Yuan, and Z. Li, "An expressive fully policy-hidden ciphertext policy attribute-based encryption scheme with credible verification based on blockchain," *IEEE Internet of Things Journal*, vol. 9, pp. 8681–8692, 2021.

[23] S. Tu, M. Waqas, F. Huang, G. Abbas, and Z. H. Abbas, "A revocable and outsourced multi-authority attribute-based encryption scheme in fog computing," *Computer Networks*, vol. 195, p. 108196, 2021.

[24] K. Li and H. Ma, "Outsourcing decryption of multi-authority ABE ciphertexts," *International Journal of Network Security*, vol. 16, pp. 286–294, 2014.

[25] Z. Liu, Z. L. Jiang, X. Wang, and S. M. Yiu, "Practical attribute-based encryption: Outsourcing decryption, attribute revocation and policy updating," *Journal of Network and Computer Applications*, vol. 108, pp. 112–123, 2018.

[26] Y. Miao, R. H. Deng, X. Liu, K. K. R. Choo, H. Wu, and H. Li, "Multi-authority attribute-based keyword search over encrypted cloud data," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, pp. 1667–1680, 2019.

[27] I. M. Ibrahim, M. G. Mostafa, S. H. N. El-Din, R. Elgohary, and H. Faheem, "A robust generic multi-authority attributes management system for cloud storage services," *IEEE Transactions on Cloud Computing*, vol. 9, pp. 435–446, 2018.

[28] L. Xu, S. Sun, X. Yuan, J. K. Liu, C. Zuo, and C. Xu, "Enabling authorized encrypted search for multi-authority medical databases," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, pp. 534–546, 2019.

[29] S. Sabitha and M. S. Rajasree, "Access control schemes in cloud: Taxonomy and performance," *International Journal of Applied Engineering Research*, vol. 14, no. 23, pp. 4209–4220, 2019.

[30] M. Cheminod, L. Durante, F. Valenza, and A. Valenzano, "Toward attribute-based access control policy in industrial networked systems," in *Proceedings of the IEEE International Workshop on Factory Communication Systems (WFCS)*, Imperia, Italy, 13–15 June 2018.

[31] M. Joshi, K. Joshi, and T. Finin, "Attribute based encryption for secure access to cloud based EHR systems," in *Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, San Francisco, CA, USA, 2–7 July 2018.

[32] M. Joshi, S. Mittal, K. P. Joshi, and T. Finin, "Semantically rich, oblivious access control using ABAC for secure cloud storage," in *Proceedings of the IEEE International Conference on Edge Computing (EDGE)*, Honolulu, HI, USA, 25–30 June 2017.

[33] Sethi, K.; Pradhan, A.; Bera, P. Practical traceable multi-authority CP-ABE with outsourcing decryption and access policy updation. *J. Inf. Secur. Appl.* 2020, 51, 102435.

[34] Zhong, H.; Zhou, Y.; Zhang, Q.; Xu, Y.; Cui, J. An efficient and outsourcing-supported attribute-based access control scheme for edge-enabled smart healthcare. *Future Gener. Comput. Syst.* 2021, 115, 486–496.

[35] Wu, Q.; Lai, T.; Zhang, L.; Mu, Y.; Rezaeibagha, F. Blockchain-enabled multi-authorization and multi-cloud attribute-based keyword search over encrypted data in the cloud. *J. Syst. Archit.* 2022, 129, 102569.

[36] Li, C.; He, J.; Lei, C.; Guo, C.; Zhou, K. Achieving privacy-preserving CP-ABE access control with multi-cloud. In *Proc. IEEE International Conference on Parallel & Distributed Processing with Applications*, Vancouver, BC, Canada, 11–13 Dec. 2018; pp. 978–981.

APPENDIX

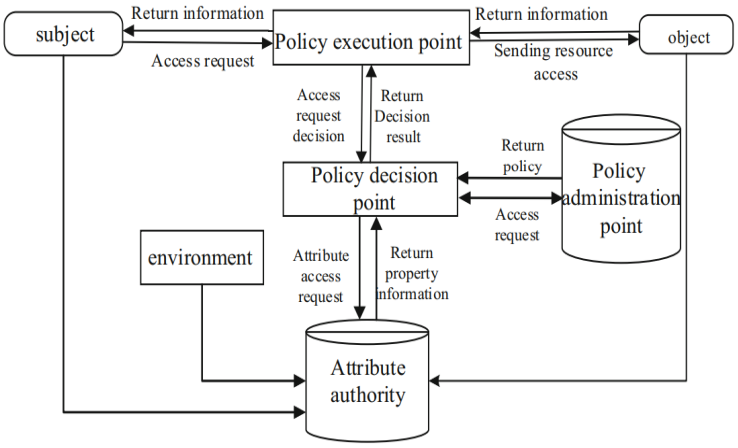


Fig. 1. Framework of the standard ABAC model.

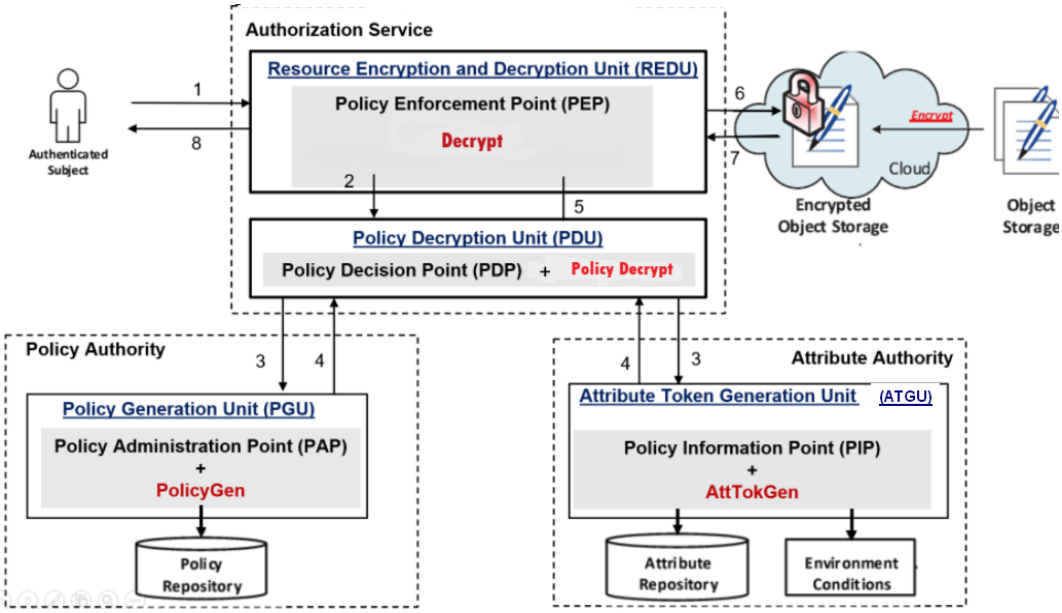


Fig. 2. Framework of the STM-ABAC model.

TABLE II. NOTATION

Symbol	Description
k	Number of attributes in the user's secret key ($k = \|A\|$)
$\|S\|$	Number of satisfied attributes in the access policy
L	Data size (length of the STM sequence)
G	Group exponentiation in G_1 or G_2 (equivalent to G_0 or G_1)
S	Skew Tent Map (STM) iteration cost
C_e	Pairing operation cost (equivalent to C_e in the literature)
X	XOR operation cost during data decryption

```
<Policy ID="Plicy1" rule-methode="Permit-overrides">
    <Target combine-method="one-and-only">
        subject     : Role          = Engineer
        subject     : Role          = Technician
        Object      : ObjectName    = BackupFile
        Action      : ActionID      = Read
    </Target>

    <Rule ID="rule1" Effect="Permit">
        <Condition function=and>
            subject     : Role          = Engineer
            Environment : Localisation  = LAC
        </Condition>
    </Rule>
<Rule ID="rule2" Effect="Permit">
        <Condition function=and>
            subject     : Role          = Technician
            Environment : Time          = Weekday
        </Condition>
    </Rule>
</Policy>
```

Fig. 3. Example of access policy for ward record.

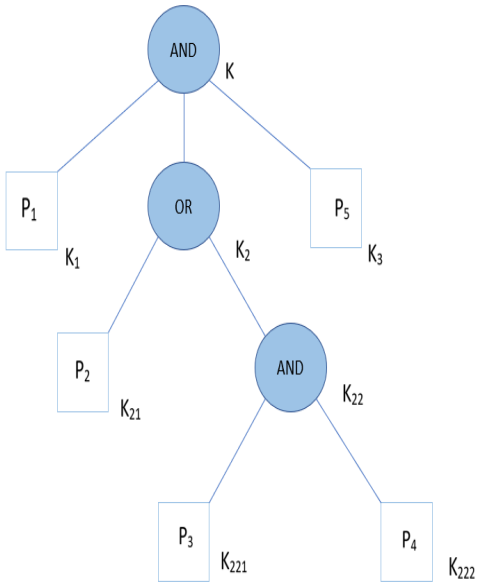
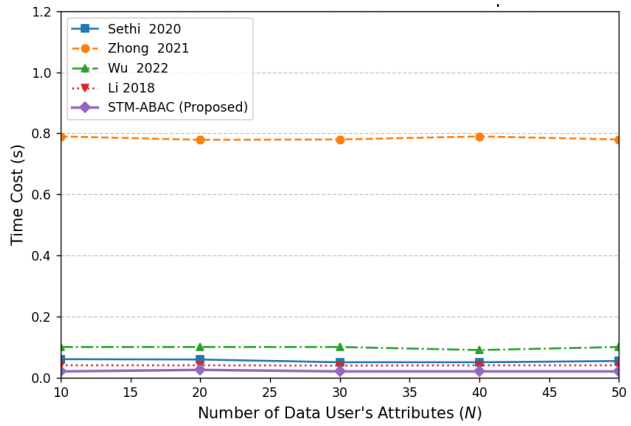


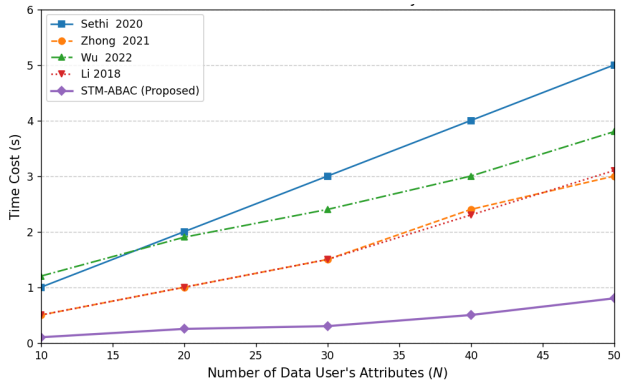
Fig. 4. Policy tree of the given access policy.

TABLE III. COMPUTATIONAL COMPLEXITY IN EACH PHASE OF THE STM-ABAC MODEL COMPARED TO [33], [34], [35], [36]

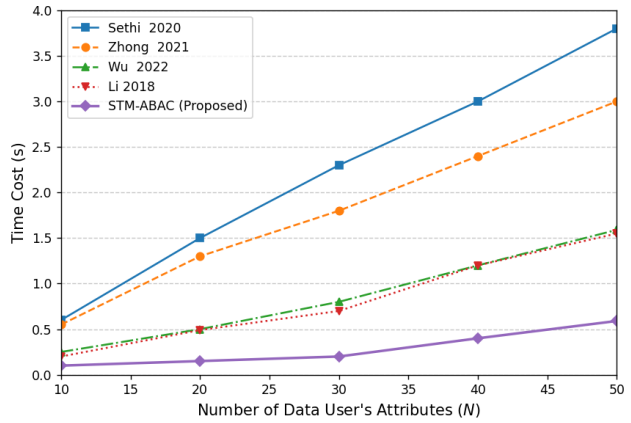
Scheme	Setup	KeyGen	Encryption	Decryption
[33]	$3G_0 + G_1$	$7G_0$	$5tG_0 + (2t+1)G_1$	$sG_0 + 3C_e$
[33]	$(m+2)G_0 + G_1$	$(4k+1)G_0$	$(4t+2)G_0$	$sG_0 + (2s+2)C_e$
[33]	$(n+3)G_0 + nG_1$	$5nG_0 + 4kG_0$	$(2t+2)G_0 + G_1$	$2sG_0 + sG_1 + (s+2)C_e$
[33]	$2G_0 + G_1$	$(4k+1)G_0$	$(2t+1)G_0 + G_1$	$(s+2)G_1 + 3C_e$
STM-ABAC (Proposed)	$(n+2)G_0 + G_1$	$kG_0 + G_1$	$\|S\|G_1 + L\S$	$\|S\|C_e + LX$



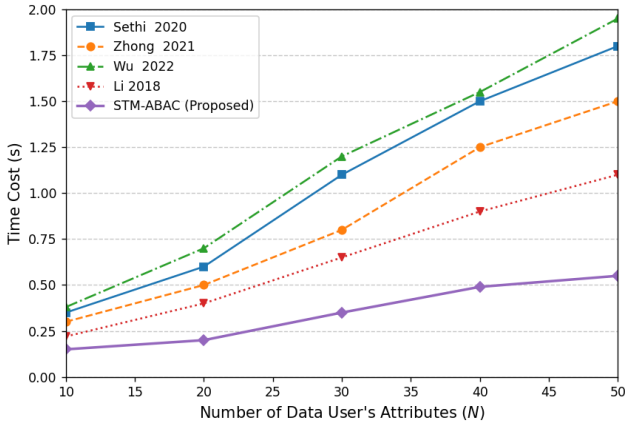
(a) The comparisons of time cost in the setup phase with the schemes.



(b) The comparisons of time cost in the Keygen phase with the schemes.



(c) The comparisons of time cost in the Encrypt phase with the schemes.



(d) The comparisons of time cost in the decryption phase with the schemes.

Fig. 5. Comparative performance analysis of the proposed STM-ABAC with the schemes [33], [34], [35], [36] .

TABLE IV. COMPARISON BETWEEN EXISTING ABE-ABAC SCHEMES AND STM-ABAC

	CP-ABE [22]	KP-ABE [21]	MA-ABE [25]	STM-ABAC
ABAC Model	Centralized	Centralized	Distributed	Distributed
Attribute Type	Subject, Object	Subject, Object	Subject, Object	Subject, Object, Action, Environment
Private Key Presentation	Encrypted	Private Key	Private Key	Initial Condition, Parameter, Number of Iterations
Policy Provider	Data Owner	User Manager	Data Owner	Data Owner
Attribute Value Authorization KDC	Data Owner	Multiple	Multiple Authorities	Multiple Authorities

Algorithm 1 Secret Key Generation

Input: K_P, K_M, A
Output: K_S
Variables: g, h (group generators); α, δ (master secrets from K_M)
Begin
Compute $K_{S_0} \leftarrow g^{\delta} \cdot g^{\alpha}$;
Initialize $K_S \leftarrow \{K_{S_0}\}$;
for each attribute $a_j \in A$ **do**
    Choose a random $r_j \in_R \mathbb{Z}_p^*$;
    Compute $K_{S_j}^{(1)} \leftarrow g^{r_j}$; // First key component
    Compute $K_{S_j}^{(2)} \leftarrow H(a_j)^{\frac{\delta}{\alpha}} \cdot g^{r_j}$;
    Add $(a_j, K_{S_j}^{(1)}, K_{S_j}^{(2)})$ to K_S;
end for
return K_S

Algorithm 2 Resource Data Encryption

Input: $\mathbf{M}$, $\mathbf{e_k}$
Output: $\mathbf{C}$
Begin
$K_{Chaotic} \leftarrow$ ChaoticSequence(e_k, b, T);
$\mathbf{C} \leftarrow \mathbf{M} \oplus K_{Chaotic}$;
return $\mathbf{C}$

Algorithm 3 Normalization using Hash Function

Input: LISTE $df = [A_S, A_E, A_A, A_O]$
Output: $df_{\text{normalized}}$
begin
for each attribute i in df **do**
  $val \leftarrow df[i]$
  $hash\_val \leftarrow$ SHA-256(val)
  $H \leftarrow$ hex2dec$(hash_val)$
  $N \leftarrow \frac{H}{2^{256}-1}$
  $df_{\text{normalized}}[i] \leftarrow N$
end for
return $df_{\text{normalized}}$

Algorithm 4 Skew Tent Map Sequence Generation

Input: $df\_normalized$, b, T { b: STM parameter $0 < b < 1$, T: number of iterations}
Output: $df\_skewtent$ {Table of generated chaotic sequences S}
STM Fonction:
$\mathbf{T_b}(x) = \begin{cases} x/b & \text{si } 0 \leq x < b \\ (1-x)/(1-b) & \text{si } b \leq x \leq 1 \end{cases}$
begin
for each attribute i in $df\_normalized$ **do**
  $X_0 \leftarrow df\_normalized[i]$
  $X_{current} \leftarrow X_0$
  Initialize $S_i \leftarrow []$
  for $t = 1$ to T **do**
    $X_{next} \leftarrow \mathbf{T_b}(X_{current})$
    S_i.append(X_{next})
    $X_{current} \leftarrow X_{next}$
  end for
  $df\_skewtent[i] \leftarrow S_i$
end for

return $df\_skewtent$

Algorithm 5 Policy Decryption Algorithm

Input: $C_\Pi = (\Pi, p_0, \{p_k\}_{k=1}^{l})$, K_S, I
Output: e_k or $\perp$
Begin
if K_S does not satisfy Π **then**
  return $\perp$
end if
Determine reconstruction coefficients$\{w_k\}_{k\in I} \in \mathbb{Z}_p^*$

$$\sum_{k\in I} w_k \cdot M_k = (1, 0, \ldots, 0)$$

$A \leftarrow e(K_{S_0}, p_0)$
$B \leftarrow 1$
for each $k \in I$ **do**
  $a_{\pi(k)} \leftarrow$ line-related attribute k
  Extract $K_{S,\text{comp1}}$ and $K_{S,\text{comp2}}$ de K_S for the attribute $a_{\pi(k)}$
  $D_k \leftarrow e(p_k, K_{S,1}) \cdot e(H(a_{\pi(k)}), K_{S,2})$
  $B \leftarrow B \cdot (D_k)^{-w_k}$
end for
$e_k \leftarrow A/B$ {Isolate the key e_k}
return e_k

Algorithm 6 Data Decryption Algorithm

Input: $ch\_encrypt$, e_k, b, T
Output: $df\_original$
Begin
$X_0 \leftarrow H(e_k) \bmod 1$
$X_{\text{current}} \leftarrow X_0$
$K_{\text{Chaotic}} \leftarrow []$

for $t = 1$ **to** T **do**
  $X_{\text{current}} \leftarrow T_b(X_{\text{current}})$
  K_{Chaotic}.append(X_{current})
end for
$S_{\text{bits}} \leftarrow$ ConvertirEnBinaire(K_{Chaotic})

$df\_original \leftarrow []$
for each cipher bloc I in $ch\_encrypt$ **do**
  Separate IV and C_{encrypt} from I
  $D_{\text{bytes}} \leftarrow C_{\text{encrypt}} \oplus S_{\text{bits}}$
  $D_{\text{string}} \leftarrow$ Convertir(D_{bytes}, "text")
  $df\_original$.append(D_{string})
end for
return $df\_original$

Algorithm 7 Policy Generation Algorithm

Input: K_M, Π, e_k, $policy\_version$, $expiry\_time$, g, H
Output: $(C_\Pi, policy\_metadata)$
begin
$(M, \pi) \leftarrow$ generate_matrix(Π)
assert M satisfies LSSS requirements
$v \leftarrow (t, r_2, \ldots, r_n)^T \in \mathbb{R}_p^n$
$p_0 \leftarrow e_k \cdot g^t$
$policy\_metadata \leftarrow \{\text{version} : policy\_version, \text{expiry} : expiry\_time\}$
$policy\_metadata.attributes \leftarrow []$
for $k = 1$ to l **do**
  $\lambda_k \leftarrow M_k \cdot v$
  $p_k \leftarrow g^{\lambda_k} \cdot H(a_{\pi(k)})^{-\lambda_k}$
  $policy\_metadata.attributes$.append($\pi(k)$)
end for
$sig \leftarrow \text{sign}(policy\_metadata, K_M)$
$C_\Pi \leftarrow (\Pi, p_0, \{p_k\}_{k=1}^l, sig)$
return $(C_\Pi, policy\_metadata)$

Algorithm 8 Attribute Token Generation

Input: A , τ, b , T
Output: T_{list}
Begin
Initialize $T_{list} \leftarrow []$
for each $val\_att \in A$ **do**
  $N \leftarrow$ Normalization(val_att)
  $X_{current} \leftarrow N$
  for $t = 1$ to T **do**
    if $0 \leq X_{current} < b$ **then**
      $X_{current} \leftarrow X_{current}/b$
    else if $b \leq X_{current} \leq 1$ **then**
      $X_{current} \leftarrow (1 - X_{current})/(1 - b)$
    end if
  end for
  $T_a \leftarrow X_{current}$
  $T_{a,\tau} \leftarrow H(T_a \parallel \tau)$
  T_{list}.append($T_{a,\tau}$)
end for
Return T_{list}

Algorithm 9 Secure Attribute Token Validation

Input: A, $T_{\text{list, received}}$, τ_{current}, b, T, $revocation\_list$
Output: Boolean (True/False)
begin
if $A \in revocation\_list$ **then**
  return False
end if
$T_{\text{list, expected}} \leftarrow \text{AttTokGen}(A, \tau_{\text{current}}, b, T)$
if constant_time_compare($T_{\text{list, received}}, T_{\text{list, expected}}$) **then**
  return True
else
  log_security_event("Invalid token attempt or expired Nonce")
  return False
end if