Secure and Fine-Grained Self-Controlled Outsourced Data Deletion in Cloud-Based IoT

Jialu Hao, Student Member, IEEE, Jian Liu, Wei Wu, Fengyi Tang, and Ming Xian

Abstract—The emerging cloud-based Internet of Things (IoT) paradigm enables IoT devices to directly upload their collected data to the remote cloud and allows data owners (DOs) to conveniently manage those data through cloud APIs, which has greatly reduced infrastructure investment and data management cost in many IoT applications. Considering that the outsourced data are out of the physical control of DOs and the cloud server (CS) cannot always be fully trusted, how to securely delete the unneeded sensitive data stored in cloud to prevent potential data leakage issues is a big challenge. Most of the existing solutions only support coarse-grained deletion and rely on the participation of the CS, so their flexibility and practicability are seriously restricted. In this article, based on an enhanced policy-based puncturable encryption (P-PUN-ENC) primitive, we propose a secure and fine-grained self-controlled outsourced data deletion scheme in cloud-based IoT. The main contribution of our scheme is that it enables DOs to precisely and permanently delete their outsourced IoT-driven data in a policy-based way without relying on the CS. To achieve this, we subtly utilize the logical relationship between the puncture policy and access policy, and design a policy transform method to convert the puncture process based on the puncture policies into the update process of access policies. Then, we utilize a key delegation technique in attribute-based encryption (ABE) to complete the corresponding key update operations. Additionally, to address the issue of growing key storage and decryption cost in P-PUN-ENC, we propose the outsourced policy-based puncturable encryption (OP-PUN-ENC) primitive by combining the key and decryption outsourcing technique with P-PUN-ENC. Comprehensive comparisons show that our proposed scheme can better meet the data deletion requirements in cloud-based IoT, and formal security proof and extensive simulation results demonstrate the reliability and efficiency of the proposed scheme.

Index Terms—Attribute-based encryption (ABE), cloud-based Internet of Things (IoT), data deletion, decryption outsourcing, puncturable encryption.

I. INTRODUCTION

In recent years, the Internet of Things (IoT) has become the focus of academia and industry [1]. IoT applications, such as environment monitoring, smart e-healthcare, and transportation [2], have greatly revolutionized our lifestyle and enhanced the development of smart cities [3], [4]. According to a related survey [5], almost 50 billion IoT devices will be connected together to provide various services by 2020. Although the immense amounts of data generated by these devices would be of great benefit for data owners (DOs), it also becomes a burden to manage them, because DOs need to purchase and maintain storage devices, which will incur huge investment and management cost. Since the cloud computing service provides abundant computation capability and storage space for customers, it enables IoT devices to directly upload their collected data to the remote cloud and allows DOs to conveniently manage those data through cloud APIs [6]. Therefore, this kind of cloud-based IoT paradigm is flourishing.

However, considering the value and privacy of the data collected by IoT devices [7], [8], security concerns from DOs arise dramatically with the frequently happened data leakage incidents in the cloud [9], [10]. Therefore, how to guarantee the confidentiality of the outsourced data is a crucial problem. Meanwhile, for fear of potential data leakage issues in the future, when some valuable or private data has been accessed or is no longer needed, DOs would like to permanently and independently delete them without relying on the cloud. This is because the cloud cannot be fully trusted and it may fail to respond to the data deletion request from the DO intentionally (for business purpose, e.g., data analysis) or inadvertently (since usually multiple backup copies of data are maintained for fault-tolerance reasons). To address these concerns, encrypting the data with the traditional symmetric or asymmetric cryptography algorithms before uploading it to the cloud seems like a straightforward solution. In this case, the DO is able to successfully achieve secure deletion on specified data by discarding its corresponding local decryption key. Nevertheless, a flexible deletion mechanism requires the DO to encrypt each data file with a unique key, which will lead to huge key management cost and raise the risk of key compromise.

Recently, Yu et al. [11] and Xue et al. [12] utilized attribute-based encryption (ABE) to realize efficient and assured outsourced data deletion. Both of them introduced an additional dummy attribute when encrypting the data, such that the previous secret key cannot be used for decryption after the ciphertext components related to the dummy attribute are updated. However, their schemes only enable the DO to delete one file each time and a third party [fog node or cloud server (CS)] is required to update the ciphertext to complete the deletion. Similarly, FADE, a policy-based assured data deletion scheme proposed by Tang et al. [13], also relies on third parties, i.e., a quorum of key managers, to
delete the corresponding control key. To enable DOs independently delete their outsourced data without relying on the cloud service provider, Xiong et al. [14] proposed a self-destructing key-policy ABE with time-specified attributes (KP-TSABE) scheme. In their scheme, each attribute of the ciphertext is labeled with a time interval during encryption, and each secret key component is associated with a time instant. As a result, after the specified expiration time, the sensitive data is securely self-destructed as it cannot be decrypted anymore. Unfortunately, the expiration time should be determined during encryption, thus the DO cannot delete the data in an on-demand and immediate way. Some other time-based key destructing schemes [15], [16] encounter the same issues.

To achieve reliable and instant data deletion simultaneously, puncturable encryption, a new form of tag-based public-key encryption primitive, proposed by Green and Miers [17], can be applied. Specifically, the data collected by IoT devices is encrypted under a set of tags, and the DO can puncture his/her secret key on tags, such that any ciphertext including the punctured tags cannot be decrypted with the punctured key. Therefore, through puncturing on tags, reliable self-controlled data deletion can be achieved immediately without help from the cloud and interaction with the IoT devices. Nevertheless, in their proposed tag-based puncturable encryption (T-PUN-ENC), only one tag is allowed in each puncture, thus data deletion can only be accomplished at a coarse-grained level. For example, when the Temperature data collected in Area A should be deleted, the DO has to sequentially puncture his/her key on tags, such that the tag Temperature and the tag Area A. In this case, since all the data including the Temperature tag cannot be decrypted with the punctured key, the Temperature data in Area B will also be affected, which disobeys the deletion requirements of the DO.

In this article, based on an enhanced policy-based puncturable encryption (P-PUN-ENC) primitive, we propose a secure and fine-grained self-controlled outsourced data deletion scheme in cloud-based IoT environment, which enables the DO to precisely and permanently delete the outsourced IoT-driven data in a policy-based way without relying on a third party. Concretely, IoT devices first encrypt their collected data with the associated tags and upload the ciphertext to the cloud. The initial secret key of the DO is able to decrypt all the ciphertext. When some ciphertext needs to be deleted, the DO specifies a fine-grained puncture (deletion) policy1 instead of a single tag, e.g., “Temperature AND (Area A OR Area B)”, to determine which kind of data will not be accessed, and then punctures his/her secret key accordingly. As a result, the data satisfying the puncture policy can be deleted at a fine-grained level and cannot be accessed by the cloud service provider or other adversaries even the punctured key is compromised or leaked by accident. Meanwhile, decryption of the data that has not been deleted is not affected.

Our contributions are summarized as follows.

1) We present P-PUN-ENC, an extension of T-PUN-ENC, to support the expressive form of puncture policies, including AND, OR, and threshold gates. We subtly utilize the logical relationship between the puncture policy and access policy, and design a policy transform method to convert the puncture process based on a puncture policy into update process of an access policy. Then, we apply a key delegation technique to update the decryption key according to the access policy only with the public key. To guarantee reliability of key puncturing, we formally prove the security of P-PUN-ENC based on the decisional bilinear Diffie–Hellman (DBDH) assumption [18]. Compared with T-PUN-ENC, the proposed P-PUN-ENC is clearly more flexible and practical, and has wider application foreground.

2) We apply P-PUN-ENC in the cloud-based IoT environment and propose a secure and fine-grained self-controlled data deletion scheme enabling the DO to independently achieve reliable and immediate deletion of the data stored in cloud. In the proposed scheme, the data collected by IoT devices is encrypted with a set of associated tags and directly uploaded to cloud without intermediate steps for the DO, and the DO is able to autonomously control his/her decryption capability via key puncturing. As a result, reliable data deletion is achieved without distributing fresh key materials to IoT devices and the participation of the CS.

3) To deal with the issue of growing key storage and decryption cost existing in both T-PUN-ENC and P-PUN-ENC, we propose an improved primitive called outsourced policy-based puncturable encryption (OP-PUN-ENC) by combining the key and decryption outsourcing technique with P-PUN-ENC. In OP-PUN-ENC, most of the punctured secret key components are blinded with a random value and outsourced to cloud to save local storage space. In addition, the cloud is able to use the outsourced key to partially decrypt the ciphertext without breaking data confidentiality, which significantly reduces the decryption cost for the DO.

4) We compare our proposed scheme with some related works comprehensively to demonstrate that it can better meet the outsourced data deletion requirements in cloud-based IoT environment. We also numerically analyze the storage and computation overheads of the proposed scheme and conduct extensive simulations to demonstrate its efficiency.

The remainder of this article is organized as follows. In Sections II and III, we review some related works and preliminaries, respectively. Some definitions are formalized in Section IV. The details of the basic P-PUN-ENC scheme and the improved OP-PUN-ENC scheme are presented in Sections V and VI. Some comparisons and analysis on the proposed scheme are given in Sections VII and VIII, followed by the conclusion in Section IX.
II. RELATED WORK

Assured outsourced data deletion is to prevent any adversary from gaining access to the data after deletion, which can be achieved via encrypting data before uploading and deleting the related local secret key when necessary [19]. Perlman [16] proposed the first assured data deletion scheme by creating a data file with an expiration time, such that data can be securely deleted through removing the ephemeral keys related to the file with an expiration time, such that data can be assuredly deleted. However, a trusted key manager, named Ephemeralizer, is required to execute the remove operation. Geambasu et al. [20] proposed Vanish enabling the data to be automatically destroyed after a period of time. In Vanish, the data is encrypted with the symmetric encryption algorithm, and the decryption key is split into shares and distributed to a large peer-to-peer network. After the shares of the key disappear from the network, the corresponding ciphertext cannot be accessed. Based on the Vanish system, Wang et al. [15] proposed a self-destructing scheme (SSDD) for electronic data. In their scheme, both the decryption key and the encrypted ciphertext are distributed into a distributed hash table (DHT) network to make it resist against not only the attacks in the DHT network, but also the traditional cryptanalysis and the brute-force attack. Xiong et al. [14] proposed a novel solution called the KP-TSABE scheme, which associates each data item with a set of attributes and every attribute with a specified time interval. After the specified expiration time, the sensitive data is securely self-destructed as it cannot be decrypted anymore.

Nevertheless, only time-based deletion is supported in the above schemes, in which the deletion time should be determined when the ciphertext is created. To enable on-demand and instant data deletion, Tang et al. [13] proposed FADE, a policy-based assured data deletion scheme, which is built upon a set of cryptographic key operations that are self-maintained by a quorum of key managers. But their scheme only supports one or two layers of Boolean expressions and requires complex decryption operations. Yu et al. [11] and Xue et al. [12] utilized ABE to achieve fine-grained data access control and assured deletion simultaneously. Both of them introduced an additional dummy attribute into the ciphertext during the encryption, such that after the ciphertext components related to the dummy attribute are updated, the previous secret key cannot be used for decryption. Unfortunately, only file-based deletion is supported and a third party (fog node or CS) is required, which lowers the flexibility and reliability of their schemes.

Since puncturable encryption (T-PUN-ENC) is initially proposed to deal with the issue of forward security, and takes nonmonotonic ABE (NM-ABE) as the building block, we also review some works on ABE and forward security.

A. Attribute-Based Encryption

Sahai and Waters [21] first introduced ABE as a one-to-many encryption primitive, which later develops into two variants: 1) key-policy ABE (KP-ABE) [18] and 2) ciphertext-policy ABE (CP-ABE) [22], [23]. In KP-ABE, the data is encrypted with a set of attributes, and the access policy is used to derive the secret key. While in CP-ABE, the access policy is embedded into the ciphertext, and the secret key is generated according to the attribute set. To support nonnegative attributes in the access policy, Ostrovsky et al. [24] first proposed nonmonotonic ABE supporting any access policy represented by a Boolean formula with AND, OR, NOT, and threshold operations.

Delegation in ABE is conceptualized by Goyal et al. [18] in their KP-ABE scheme, and then concretely discussed in [25] and [26]. Key delegation empowers a user to modify a key associated with a given access policy into a key embedding a more restrictive one, and ciphertext delegation allows an untrusted party to update a ciphertext to be accessible only under a more restrictive policy. Green and Miers [17] combined NM-ABE with the key delegation technique to construct T-PUN-ENC, but only a simple form of puncture policy, i.e., single tag, is supported in their scheme.

In a way, puncturable encryption is similar to revocation in ABE [27], [28], as both of them focus on the issue of decryption ability revocation. The difference is that revocation in ABE enables the DO to revoke the decryption ability of certain users or attributes, while puncturable encryption emphasizes on revocation of his/her own decryption ability.

B. Forward Security

Forward security protects past sessions against future compromises of secret keys [29]. Most of the existing schemes to achieve forward security often require highly available network infrastructure to distribute fresh key materials to senders, or force changes to client interaction. Canetti et al. [30] proposed a forward secure public key encryption scheme (FS-PKE), which does not require changes to the key distribution model. However, their scheme has not been widely adopted, because removing decryption capability for a given time period makes the user lose access to all messages sent during prior time periods. By combining T-PUN-ENC with a variant of FS-PKE and regarding tags in T-PUN-ENC as time periods, Green and Miers’s scheme [17] enables users to instantly delete selected messages with precision and achieves more practical forward-secure messaging with low overhead. Later, Derler et al. [31] introduced a new bloom filter encryption (BFE) primitive, and yielded a new puncturable encryption mechanisms with extremely efficient puncturing. In their scheme, only a small number of very efficient computation and deletion of certain parts of the secret key are involved in a puncturing operation. However, the forward secure schemes only consider the factor of time slot, which can only derive restricted time-based data deletion solutions.

III. PRELIMINARIES

A. Lagrange Polynomial Interpolation

Given a set of \( d + 1 \) points \( (x_0, y_0), (x_1, y_1), \ldots, (x_d, y_d) \) with no duplicate \( x \), a unique polynomial \( q(x) \) of degree at most \( d \) exists with \( q(x_i) = y_i \), for each \( i = 0, 1, \ldots, d \). Specifically, the interpolating polynomial in the Lagrange form
is a linear combination as follows:

\[ q(x) = \sum_{i=0}^{d} y_i \Delta_i(x) \]

where the Lagrange coefficient \( \Delta_i(x) \) is

\[ \Delta_i(x) = \prod_{j=0, j \neq i}^{d} \frac{x - x_j}{x_i - x_j}. \]

B. Access Structure and Access Tree

1) Access Structure: An access structure [18] on an attribute universe \( U \) is a collection \( A \) of nonempty sets of attributes. The sets in \( A \) are called the authorized sets. In addition, an access structure which satisfies the following requirement is monotonic: if \( B \in A \) and \( B \subseteq C \), then \( C \in A \).

In our context, the concept of access structure is also referred to as access policy and the role of attributes is replaced with tags. Moreover, the puncture (deletion) policy in our scheme has the same definition with access policy and is considered as monotonic. Note that a puncture policy can be seen as the opposite of an access policy, thus in our scheme, we first transfer the puncture policy into a nonmonotonic access policy (including negated tags) and utilize nonmonotonic ABE algorithms to compute keys based on the generated nonmonotonic access policy.

2) Access Tree: Goyal et al. [18] utilized a tree structure to represent the access policy in their ABE scheme. Specifically, a nonleaf node \( x \) in the tree structure represents a \((k_x, n_x)\) threshold gate, where \( k_x \) denotes the threshold value and \( n_x \) denotes the number of child nodes. Note that, \( k_x = 1 \) means the OR gate and \( k_x = n_x \) means the AND gate. Each leaf node is associated with an attribute and has a threshold value \( k_x = 1 \). Let \( \text{par}(x) \) denote the parent node of the node \( x \), and \( \text{id}(x) \) be the index of \( x \) ordered by its parent.

To share a secret \( \alpha \) based on an access tree, a random polynomial \( p_x \) is defined for each node \( x \) in the top-down manner, where the degree \( d_x = k_x - 1 \). For the root node \( r \), \( p_r(0) = \alpha \). For each nonroot node \( x \), \( p_x(0) = p_{\text{par}(x)}(\text{id}(x)) \). As a result, for an attribute (leaf) node \( i \), \( p_i(0) = p_{\text{par}(i)}(\text{id}(i)) \) is seen as the secret share of \( \alpha \) assigned to it.

If a set of attributes \( S \) satisfies the access tree \( T_r \) with root node \( r \), we denote that \( T_r(S) = 1 \). Specifically, if \( x \) is an attribute node, \( T_x(S) \) returns 1 only if \( x \in S \). If \( x \) is a nonleaf node, \( T_x(S) = 1 \) only if at least \( k_x \) child nodes of \( x \) return 1. In addition, for an access tree \( T_r \) with root node \( r \), if \( T_r(S) = 1 \), the secret \( \alpha \) associated with \( r \) can be recovered by combining the shares assigned to the attributes belonging to \( S \) in the bottom-up manner using the Lagrange polynomial interpolation technique recursively.

C. De Morgan’s Laws

De Morgan’s Laws\(^1\) describe how mathematical statements and concepts are related through their opposites and can be applied to sets, propositions, or logic gates. Generally, the structure of De Morgan’s Laws is represented as follows.

\(^1\)https://brilliant.org/wiki/de-morgans-laws/

D. Decisional Bilinear Diffie–Hellman Assumption [18]

Let \( a, b, c, z \) be chosen randomly from \( Z_p \) and \( g \) be a generator of \( G \). The DBDH assumption is that no probabilistic polynomial time (PPT) algorithm \( B \) can distinguish a valid BDH tuple \((A = g^a, B = g^b, C = g^c, Z = e(g, g)^{abc})\) from a random tuple \((A = g^a, B = g^b, C = g^c, Z = e(g, g)^r)\) with more than a negligible advantage. The advantage of \( B \) is defined as

\[ \left| \Pr[B(A, B, C, e(g, g)^{abc}) = 0] - \Pr[B(A, B, C, e(g, g)^r) = 0] \right| \]

where the probability is taken over the random choice of \( g, a, b, c, z, \) and the random bits consumed by \( B \).

IV. Definitions

In this section, we formalize the system and threat models in cloud-based IoT, give the algorithm definitions and security model of P-PUN-ENC, and define the design goals of our proposed scheme.

A. System and Threat Models

As shown in Fig. 1, three parties are included in the self-controlled data deletion model in cloud-based IoT environment, that is, a variety of IoT devices, the DO and the CS.

1) IoT Devices (Dev): Dev means IoT devices, such as mobile phones, sensors, or wearable devices that are deployed or managed by the DO to collect kinds of data. Each Dev is described with some tags related to its characteristics, such as device identifier, location, and content subject. The collected data will be encrypted under its corresponding tags and uploaded to the CS directly.

2) Data Owner: DO could be an enterprise deploying the sensors, a platform publishing the sensing tasks or a person wearing the health devices, who is the owner of the IoT-driven data stored in cloud. DO is able to access and manage the outsourced...
data through cloud APIs, and delete the unneeded data according to specified puncture policies on data tags by actively puncturing his/her secret key. DO is also responsible for system public key generation and distribution.

3) **Cloud Server:** CS provides online services of data storage and retrieval for DO. All the data collected by Dev is encrypted and stored in CS along with the associated tags, and can be retrieved by DO at anytime and from anywhere.

With regard to the threat model in the above system, DO is fully trusted in our system as it is the key beneficiary. We assume CS to be honest-but-curious, which means that it will faithfully follow the protocols, i.e., store and distribute data, but it may be interested in mining any sensitive information from those stored data for their own marketing purpose [32]. Meanwhile, CS may be reluctant to respond the data deletion request due to business purpose or not be able to successfully and thoroughly clear those data due to technical reasons. Considering that our scheme mainly focuses on the deletion of the outsourced data, the adversaries (the cloud or outside attackers) are assumed to have the ability to compromise the fully and thoroughly clear those data due to technical reasons.

C. Security Model for P-PUN-ENC

We define the following selective-set model (s-IND-PUN-CPA) for proving the security of P-PUN-ENC under chosen plaintext attack.

1) **Init:** The adversary \( A \) commits to the challenger \( C \) a set of tags \( T = \{ t_1, \ldots, t_d \} \).

2) **Setup:** The challenger runs the KeyGen algorithm to generate \( PK \) and \( SK_0 \) with the input of a security parameter \( \xi \) and the number of tags associated with a ciphertext \( d \), and sends the public key \( PK \) to the adversary. The challenger also initializes an empty set \( P \) and a counter \( k = 0 \).

3) **Query Phase 1:** The following two kinds of queries can be repeatedly issued by the adversary and answered by the challenger.

   a) **Puncture** (\( PP \)): The adversary submits a puncture policy \( PP \) to the challenger. The challenger increments \( k \), computes Puncture\((PK, SK_{k-1}, PP) \rightarrow SK_k\) and adds \( PP \) to \( P \).

   b) **Corrupt**(): The challenger returns the most recent secret key \( SK_k \) to the adversary as the response for the first time of this kind of query and return \( \perp \) for all subsequent queries. In addition, if the challenge set \( T \) does not satisfy any puncture policy \( PP \) in \( P \), the challenger returns \( \perp \).

4) **Challenge:** The adversary submits two messages \( M_0 \) and \( M_1 \) with the same size to the challenger. The challenger runs the Encrypt\((PK, M, T)\) algorithm with a randomly selected bit \( v \) in \( \{0, 1\} \) and returns the generated ciphertext \( CT \) to the adversary.

5) **Query Phase 2:** Query phase 1 is repeated.

6) **Guess:** The adversary outputs a guess \( v' \) of \( v \). If \( v' = v \), it wins the game.

In this game, we use \( |\text{Pr}[v' = v] - (1/2)| \) to define the advantage of an adversary. Note that the above security game can be extended to handle chosen ciphertext attacks by allowing the decryption queries in the query phase.

**Intuition of Security Notion:** The s-IND-PUN-CPA model is similar with the indistinguishability definition for public key encryption in the selective-set model but adds new oracles of Puncture and Corrupt. Specifically, the Puncture oracle updates the current secret key based on the puncture policy \( PP \) to guarantee that the punctured secret key cannot be used to decrypt the ciphertext satisfying \( PP \), and the Corrupt oracle gives the most recent state of the secret key to the adversary. The adversary cannot corrupt the punctured secret key unless the challenge set \( T \) satisfies at least one of the puncture policies in \( P \). This restriction in the Corrupt queries prevents attacks in which the adversary may trivially decrypt the challenge ciphertext if it has not been punctured.

**Definition 1:** A P-PUN-ENC scheme is secure in the selective-set model of indistinguishability against the chosen plaintext attack (s-IND-PUN-CPA) if no PPT adversary has non-negligible advantage in the above security game.
D. Design Goals

We intend to design a secure and fine-grained data deletion scheme in cloud-based IoT to enable the DO to securely and independently delete the sensitive IoT-driven data stored in cloud in a policy-based way. Thus, even when the secret key of the DO is leaked, the deleted data cannot be recovered. More concretely, the following goals should be fulfilled in our scheme:

1) Data Confidentiality: The data stored in CS cannot be accessed without the private key of DO. The data that has been deleted cannot be accessed by any party anymore, even the punctured secret key is compromised.

2) Fine-Grained Data Deletion: Fine-grained policy-based data deletion should be supported, such that data can be deleted flexibly and precisely. Specifically, the puncture policy should be represented as an expressive Boolean formula involving AND, OR, and threshold operations.

3) Efficiency: The system operations, especially those for Dev and DO, should be completed with low computation and storage overheads.

V. Secure and Fine-Grained Self-Controlled Outsourced Data Deletion

In this section, we first introduce the main idea about our proposed scheme. Then, we present the scheme in two levels: algorithm level and system level. The concrete constructions of the algorithms in P-PUN-ENC are described in the algorithm level, and the system level focuses on the upper operations invoking the underlying algorithms. Finally, the security of P-PUN-ENC is proved formally.

A. Main Idea

In our scheme, we endeavor to achieve fine-grained key puncturing to enable the DO to autonomously delete the outsourced IoT-driven data in a flexible and reliable manner, such that the ciphertext of the deleted data still remains confidential against the CS or other adversaries even the punctured private key of the DO is leaked. By taking the NM-ABE construction as the building block, T-PUN-ENC [17] allows one-tag-based data deletion. Specifically, in T-PUN-ENC, the data is encrypted under a set of tags \( T = \{t_1, \ldots, t_d\} \), and an efficient Puncture algorithm, on input the current secret key \( SK \) and a tag \( t_i \), enables the DO to generate a new secret key \( SK' \) which can decrypt any ciphertext \( SK \) can decrypt, except for those including the tag \( t_i \). As a result, the data including the tag \( t_i \) is deleted successfully and cannot be recovered in any case. The puncture process can be executed repeatedly and sequentially at many points to guarantee that all the involved ciphertext is deleted. To deal with the coarse-grain issue in T-PUN-ENC, we propose the P-PUN-ENC primitive, as an extension of T-PUN-ENC, which enables the DO to puncture his/her secret key based on expressive policies, such that the outsourced data can be precisely and securely deleted in a fine-grained way.

Based on our observation, puncturing the secret key on a specified puncture policy \( PP_i \) can be achieved by adding the secret key with some restrictions related to the opposite of the puncture policy \( PP_i \). Here, we define the restriction as an additional access policy \( AP_i \), where \( AP_i = \neg PP_i \). According to the De Morgan’s Laws, any monotonic puncture policy containing the interior nodes of AND or OR gate can be easily transformed into an access policy with only negated tags (leaf nodes), e.g., \( \neg t \). In addition, we have deduced that, the opposite of a \((k, n)\) threshold gate in the puncture policy can be represented as an \((n-k+1, n)\) gate along with the negation of all its child nodes. Fig. 2 shows an example of transforming a puncture policy to an access policy, in which AND and NOT gates are exchanged, the \((1, 3)\)-gate is converted into a \((3, 3)\)-gate, and each tag \( t \) of the leaf node is changed as its opposite \( \neg t \). Here, \( \neg t \) has the same meaning with \( \neg t \).

Furthermore, the puncture process can be executed repeatedly and sequentially, and the relationship between a new puncture policy with the existing puncture policies is OR logic, which means the compositive puncture policy \( PP = PP_1 \lor PP_2 \lor \ldots \lor PP_k \) and the ciphertext satisfying any puncture policy \( PP_i \) cannot be decrypted with the punctured key. Correspondingly, the relationship between the access policies is AND logic, and the compositive access policy \( AP = AP_1 \land AP_2 \land \ldots \land AP_k \). The two figures in Fig. 3 show the logical relationships of the multiple puncture policies and access policies, respectively. Indeed, the compositive access policy can also be seen as the opposite of the compositive puncture policy.

Considering that the compositive access policy is becoming more restrictive after each puncture, the key delegation technique in KP-ABE can be utilized to update the secret key. Specifically, appending a new puncture policy \( PP_1 \) is equal to add its corresponding access policy \( AP_1 \) as a child node of the root AND node of the current compositive access policy. To update the key correspondingly, we can reshare an existing share \( \lambda \) of the root AND node secret \( \alpha \) by splitting it as \( \lambda = \lambda_i \) and \( \lambda_i \), and assigning \( \lambda_i \) to the new access policy \( AP_1 \). Then, \( \lambda_i \) is further shared with the leaf nodes (negated tags) of \( AP_i \). Note that, the secret key components bound with the previous secret share \( \lambda \) can be updated to the new share \( \lambda - \lambda_i \) with only the public key. If the tags of the ciphertext do not satisfy the
access policy \( AP_i \) associated with the punctured key, i.e., satisfy the puncture policy \( PP_i \), the share \( \lambda_j \) that is requisite for recovering \( \alpha \) cannot be calculated, thus the ciphertext cannot be decrypted with the punctured key. Fig. 4 shows a simple example for secret resharing with a new access policy \( AP_k \), in which the existing secret share \( \lambda \) related to \( AP_1 \) is changed as \( \lambda - \lambda_k \), and \( \lambda_k \) is assigned to the new access policy \( AP_k \) as its associated secret share.

B. Algorithm Constructions of P-PUN-ENC

The concrete constructions of P-PUN-ENC are given as follows.

**KeyGen(\( \xi, d \)) \rightarrow (PK, SK_0):** This algorithm takes as input a security parameter \( \xi \) and a number of tags associated with a ciphertext \( d \), and outputs the public key \( PK \) and the initial secret key \( SK_0 \). Specifically, it first generates a bilinear map \( e : G \times G \rightarrow G_T \), where \( G \) and \( G_T \) are multiplicative cyclic groups of prime order \( p \), and \( g \) is a generator of \( G \). Then, it computes \( g_1 = g^\alpha \) and \( g_2 = g^\beta \) with random \( \alpha, \beta \in Z_p^* \). The algorithm randomly selects a degree-\( d \) polynomial \( g(x) \) with the restriction that \( g(0) = \beta \), and defines \( Q(x) = g^{\alpha(x)} \).

The public key \( PK \) is generated as

\[
PK = (e, G, G_T, g, g_1, g_2, Q(1), Q(2), \ldots, Q(d)).
\]

The initial secret key \( SK_0 \) is

\[
SK_0 = \left\{ (t_0, K_0, g_2^{t_0 + m}, L_0, g^\alpha, Q(0) = 0) \right\}
\]

where \( t_0 \in Z_p^* \) is a unique virtual tag not used during normal operations, and \( m \) is a random number in \( Z_p \).

**Remarks:** Given the parameters of \( (g_2, Q(1), Q(2), \ldots, Q(d)) \) in \( PK \), \( Q(\cdot) \) can be easily computed through Lagrange polynomial interpolation on the exponents of \( \{Q(i)\}_{i=0}^d \), i.e.,

\[
Q(x) = \sum_{i=0}^d g^{\sum_{j=0}^d (i)(j)(x)} = \prod_{i=0}^d g^{\sum_{j=0}^d Q(i)\Delta_j(x)}.
\]

In addition, all the ciphertext encrypted under tags not containing the virtual tag \( t_0 \) can be successfully decrypted with the original secret key \( SK_0 \).

**Encrypt(PK, M, T) \rightarrow CT:** This algorithm takes as input the public key \( PK \), a message \( M \) and a set of tags \( T = \{t_1, \ldots, t_d\} \), and outputs the ciphertext \( CT \). Specifically, it first randomly selects \( s \in Z_p^* \), and computes \( C = M \cdot e(g_1, g_2)^s \) and \( D = g^s \). Then, for each tag \( t_i \in T \), it computes \( E_i = Q(t_i)^{s} \). Finally, the ciphertext is generated as

\[
CT = (T, \tilde{C}, D, \{E_i\}_{i=1}^d).
\]

**Puncture(PK, SK_{k-1}, PP_k) \rightarrow SK_k:** This algorithm takes as input the public key \( PK \), the current secret key \( SK_{k-1} \) and the puncture policy \( PP_k \), and outputs a new punctured secret key \( SK_k \). The algorithm first converts the puncture policy \( PP_k \) into an access policy \( AP_k \). The policy transform method is designed in the top-down manner as follows.

1) **Nonleaf Node (Threshold Gate):** For each nonleaf node expressed as a threshold gate in \( PP_k \):
   a) if it is an AND gate, change it as a OR gate;
   b) if it is a OR gate, change it as an AND gate;
   c) if it is a \( (k, n) \) gate, change it as an \( (n - k + 1, n) \) gate.

2) **Leaf Node (Tag):** For each leaf node associated with a tag \( \tau \) in \( PP_k \), it is changed as its opposite, i.e., \( NOT \) \( \tau \).

After that, the algorithm parses the current secret key \( SK_{k-1} \) as \( \{s_{k_0}, s_{k_1}, \ldots, s_{k_{k-1}}\} \), and \( s_{k_0} \) as \( \{t_0, K_0, L_0, 0\} \). It selects two random value \( \lambda_k, r_k \in Z_p^* \), and computes

\[
s_0' = \left( t_0, K_0', g_2^{t_0 - \lambda_k} = g_2^{t_0 + m + r_0 - \lambda_1 - \cdots - \lambda_k}, L_0' = L_0 \cdot g^s = g^{r_0 + \cdots + r_k}, Q_0' = Q(t_0) \right) \in Z_p^*.
\]

Then, the algorithm splits the \( \lambda_k \) according to the transformed access policy \( AP_k \) by defining a polynomial for each node in \( AP_k \), such that the secret share assigned to the leaf node \( \neg t_{k,j} \in LN(AP_k) \) is \( \lambda_k \), where \( LN(AP_k) \) means the set of leaf nodes in \( AP_k \). For each leaf node \( \neg t_{k,j} \in LN(AP_k) \), it selects a random value \( r_{k,j} \in Z_p^* \) and computes

\[
s_{k,j} = \left( t_{k,j}, K_{k,j} = g_2^{t_{k,j} + r_{k,j}}, L_{k,j} = g^{r_{k,j}}, Q_{k,j} = Q(t_{k,j})^{r_{k,j}} \right).
\]

Finally, the new punctured secret key is generated as

\[
SK_k = \langle PP_1, AP_k \rangle = \{s_{k_0}, s_{k_1}, \ldots, s_{k_{k-1}}, s_{k_k}\}
\]

where \( SK_k = \{s_{k_{k-1}}, \neg t_{k,j} \in LN(AP_k) \} \).
nodes to the root node by using the Lagrange polynomial interpolation technology on the exponents recursively, such that \( Z_i = e(g, g)^{\beta_i} \) can be recovered.

In addition, the algorithm parses \( SK_k^0 \) in \( SK_k \) as \( \{0, 0, 0, 0, 0\} \). It also calculates a set of Lagrange coefficients \( \tau_1, \ldots, \tau_d, \tau^* \) such that \( \sum_{m=1}^{d} (\tau_m \cdot q(t_m)) + \tau^* \cdot q(t_0) = q(0) = \beta \), and computes

\[
Z_0 = \frac{e(K_0, D)}{e(L_0, \prod_{m=1}^{d} (F_m \cdot t_m)) \cdot e(Q_0, D)^{\tau^*}} = \frac{e(g, g)^{\beta(t_0+\cdots+t_2-\cdots-\lambda_k)}}{e(g, g)^{(m+\cdots+n_i)(\sum_{m=1}^{d}(t_m-\lambda_k))}} = e(g, g)^{\beta(a-\cdots-\lambda_k)}.
\]

Finally, the message is recovered by computing \( M = \tilde{C}/\prod_{i=0}^{d} Z_i \).

C. System Operations

Our scheme consists of four kinds of system operations corresponding to the algorithms in P-PUN-ENC: 1) system initialization by DO; 2) data encryption by Dev; 3) data deletion by DO; and 4) data decryption by DO.

System Initialization: In this phase, DO first defines a system security parameter \( \xi \) and the number of tags \( d \) that will be used to describe the IoT devices. Then, it calls the KeyGen(\( \xi \), \( d \)) algorithm to generate the system public key \( PK \) and the initial secret key \( SK_0 \). When deploying the IoT devices, DO will assign a set of tags \( T = \{t_1, \ldots, t_d\} \) to them and embed the corresponding public key components into them.

Data Encryption: To save local storage resources, Dev will regularly upload their collected data \( M \) to CS. To protect data confidentiality, Dev calls the Encrypt(\( PK \), \( M \), \( T \)) algorithm to generate the ciphertext \( CT \) and upload \( CT \) to CS. CS stores all the ciphertext collected by IoT devices, and each ciphertext is described by a set of tags.

Data Deletion: To delete certain kinds of ciphertext in CS, DO first defines a deletion policy \( PP_D \) to specify the ciphertext to be deleted. Then, it calls the Puncture(\( PK \), \( SK_k^{d-1} \), \( PP_D \)) algorithm to update its current secret key \( SK_k^{d-1} \) to the new punctured secret key \( SK_k \). As a result, the ciphertext stored in CS with tags satisfying \( PP_D \) cannot be decrypted with \( SK_k \), which can be considered to be successfully and securely deleted.

Data Decryption: DO is still able to access those undeleted data. After downloading the ciphertext \( CT \) that has not been punctured, DO is able to obtain the plain data by calling the Decrypt(\( CT \), \( SK_k \)) algorithm.

D. Security Proof

We prove that the security of P-PUN-ENC in the s-IND-PUN-CPA model reduces to the hardness of the DBDH assumption.

Theorem 1: If a PPT adversary can break our scheme with advantage \( \epsilon \) in the s-IND-PUN-CPA security game, then a simulator can be constructed to play the DBDH game with advantage \( \epsilon/2 \).

Proof: Suppose there exists a PPT adversary \( A \) that can attack our scheme in the s-IND-PUN-CPA model with advantage \( \epsilon \). We build a simulator \( B \) that can play the DBDH game with advantage \( \epsilon/2 \). The simulation proceeds as follows.

We first let the challenger \( C \) set the groups of \( G \) and \( GT \) with an efficient bilinear map \( e \) and prime order \( p \). Then, \( C \) selects a random bit \( u \) outside of \( B \)'s view. If \( u = 0 \), \( C \) sets \( (A, B, C, Z) = (g^{a^i}, g^{b^i}, g^{c}, e(g, g)^{abc}) \); otherwise, it sets \( (A, B, C, Z) = (g^{a^i}, g^{b^i}, g^{c}, e(g, g)^{ef}) \) with random \( a, b, c, z \in \mathbb{Z}_p \) and generator \( g \in G \).

Init: The simulator \( B \) runs. \( A \) chooses a set of tags \( T = \{t_1, \ldots, t_d\} \) as the challenge set and commits it to \( B \).

Setup: The simulator \( B \) assigns the public parameters \( g_1 = A \) and \( g_2 = B \), which implicitly sets \( a = a \) and \( \beta = b \). Then, \( B \) randomly chooses \( d \) points \( t_1, \ldots, t_d \) from \( \mathbb{Z}_p \), and implicitly sets \( q(x) \) such that \( q(0) = \beta \) and \( q(t_i) = \theta_i \) for \( i = 1, \ldots, d \). Thus, for \( x = 1, \ldots, d \), \( Q(x) = g^{q(x)} \) in the public key can be computed via Lagrange polynomial interpolation on the exponents of \( \{g^{q(x)}\} \) and \( B \). Finally, \( B \) sets the public key as

\[
PK = \langle e, G, GT, p, g, g_1, g_2, Q(1), Q(2), \ldots, Q(d) \rangle.
\]

Observe that the above values in the public key are distributed identically as that in the actual scheme. In addition, the challenger also initializes an empty set \( P \) and a counter \( n = 0 \).

Query Phase 1: Since only the corrupted key is given to the adversary, \( B \) just needs to simulate the generation of the corrupted key, which is the same with the key generated from the Punctured algorithm one by one. Suppose that the Corrupt query is issued by the adversary after \( k \) punctures, we have \( P = \{PP_1, \ldots, PP_k\}, n = k \) and at least one puncture policy in \( P \) is satisfied by the challenge set \( T \). Thus, the compositive access policy related to the corrupted key is \( AP = AP_0 \) AND \( AP_1 \) AND \cdots AND \( AP_k \), where \( AP_0 = NOT \) \( t_0 \) and \( AP_i = NOT \) \( PP_i \) for \( i = 1, \ldots, k \).

Note that, the main idea to generate the secret key is splitting the secret \( a \) into multiple shares for the leaf nodes of each access policy \( AP_i \). We first define the following two procedures: 1) ShareKnown and 2) ShareUnknown. The first procedure is used to share a clearly known secret and the second one is to share a secret only based on its corresponding group element. Here, we intuitively use the tree structure \( T_s \) to represent access policy.

1) ShareKnown(\( T_s, \lambda, T \)): This procedure takes as input an access policy \( T_s \), the challenge set \( T \) and an integer \( \lambda \in \mathbb{Z}_p \). Note that, \( \lambda \) is the secret to be shared according to \( T_s \), which is clearly known. It first sets up a polynomial \( p_s \) of degree \( d_s \) such that \( p_s(0) = \lambda \) and the secret share for its child node \( x' \) is \( p_s(id(x')) \). Then, it recursively calls ShareKnown(\( T_{s'}, p_s(id(x')) \), \( T \)) for each child node of \( x \) until to the leaf nodes.

2) ShareUnknown(\( T_s, g^{\lambda}, T \)): This procedure takes as input an access policy \( T_s \), the challenge set \( T \) and a group element \( g^{\lambda} \in G \). Note that, \( \lambda \) is the secret to be shared according to \( T_s \), but only \( g^{\lambda} \) is given. In addition, we have \( T_s(T) \neq 1 \) in this case. To share the secret \( \lambda \), it first defines a polynomial \( p_s \) of degree \( d_s \) such
that \( p_s(0) = \lambda_x \). Since \( T_x \) is not satisfied, no more than \( d_x \) children of \( x \) is satisfied. Let \( h_x \leq d_x \) be the number of satisfied children of \( x \). For each satisfied child \( x' \) of \( x \), it chooses a random point \( \lambda_{x'} \in Z_p \) and sets \( p_s(\text{id}(x')) = \lambda_{x'} \). It then fixes the remaining \( d_x - h_x \) points of \( p(x) \) randomly to completely define \( p_t \). Then, it recursively computes the shares for the rest of the nodes in \( T_x \) as follows.

a) If \( x' \) is a satisfied child node of \( x \), it calls \( \text{ShareUnknown}(T_{x'}, p_s(\text{id}(x'))), T \). Because the secret \( p_s(\text{id}(x')) \) related to the node \( x' \) is clearly known as \( \lambda_{x'} \) in this case.

b) If \( x' \) is an unsatisfied child node of \( x \), it calls \( \text{ShareUnknown}(T_{x'}, g^{|\text{id}(x')|}, T) \). Since \( g^{p_s(0)} = g^{\lambda_x} \) is given in the form of group element, only \( g^{p_s(\text{id}(x'))} \) can be obtained via interpolation on the exponents.

To generate the corrupted secret key components related to the access policy \( \mathcal{AP} \), the simulator \( S \) calls \( \text{ShareUnknown}(T_x, g^d, T) \), where \( T_x \) with root node \( r \) represents the tree structure of \( \mathcal{AP} \), and \( a \) is the secret to be shared.

After recursively calls of ShareKnown or ShareUnknown procedures, the secret share of the leaf node \( \tau_{i,j} \) will be given in the form of \( \lambda_{i,j} \) or \( g^{\lambda_{i,j}} \). Specifically:

1) if the leaf node \( \tau_{i,j} \) is finally called via ShareKnown, its share is clearly known as \( \lambda_{i,j} \) no matter whether \( t_{i,j} \in T \) or not. In this case, \( B \) randomly chooses \( r_{i,j} \in Z_p \), and computes the key components related to \( \tau_{i,j} \) as follows:

\[
K_{i,j} = g^{\lambda_{i,j} - r_{i,j}}, \quad L_{i,j} = g^{r_{i,j}}, \quad Q_{i,j} = Q(t_{i,j})^{r_{i,j}}.
\]

Note that, if \( t_{i,j} \in T, \theta_{i,j} \) is known and \( Q(t_{i,j}) = g^{|\theta_{i,j}|} \). Otherwise, \( Q(t_{i,j}) = g^{\theta_{i,j}} \) can be computed via Lagrange polynomial interpolation on the exponents of \( \{g^{|\theta_{i,j}|}\}_{i,j \in [1,d]} \) and \( B \);

2) if the leaf node \( \tau_{i,j} \) is finally called via ShareUnknown, which means the leaf node is not satisfied, i.e., \( t_{i,j} \in T \), its secret share is given as \( g^{\lambda_{i,j}} \). In this case, \( B \) randomly chooses \( r_{i,j} \in Z_p \), and implicitly sets \( r_{i,j} = r_{i,j} - \lambda_{i,j} \). Then, it computes the key components related to \( \tau_{i,j} \) as follows:

\[
K_{i,j} = g^{r_{i,j}}, \quad L_{i,j} = g^{\theta_{i,j} - \lambda_{i,j}}, \quad Q_{i,j} = Q(t_{i,j})^{r_{i,j} - \lambda_{i,j}}.
\]

Note that, given \( g^{\theta_{i,j}} \), the second element \( L_{i,j} = g^{\theta_{i,j} - \lambda_{i,j}} = g^{\theta_{i,j}} / g^{\lambda_{i,j}} \) can be easily computed. While for the last element, since \( t_{i,j} \in T, Q(t_{i,j}) = g^{|\theta_{i,j}|} \), where \( \theta_{i,j} \) is clearly known by \( B \). Thus, \( Q_{i,j} = g^{|\theta_{i,j}|(r_{i,j} - \lambda_{i,j})} = g^{|\theta_{i,j}|r_{i,j}} / g^{|\theta_{i,j}| \lambda_{i,j}} \) can also be computed.

Therefore, \( B \) is able to construct the corrupted secret key related to the composite access policy \( \mathcal{AP} \). Note that, the distribution of the corrupted secret key is identical to that in the actual scheme.

Challenge: The adversary submits two messages \( M_0 \) and \( M_1 \) with the same size to the simulator. \( B \) flips a fair binary coin \( v \), and returns an encryption of \( M_v \). The ciphertext is outputted as

\[
CT = \langle T, \tilde{C} = M_v \cdot Z, D = C, \{E_i = C^{\theta_i}\}_{i \in [1,d]} \rangle.
\]

As mentioned in the beginning, \( u \) is randomly selected by the challenger to determine the elements \( Z \) in the BDH tuple. If \( u = 0 \), then \( Z = e(g, g)^{\lambda_x} \). In this case, we let \( s = c \), then \( e(g, g)^{\lambda_x} = e(g_1, g_2)^s \), \( D = C = g^s \) and \( E_i = C^{\theta_i} = g^{\theta_i} Z = Q(t_i)^s \). By inspection, the ciphertext \( CT \) has the same distribution with the ciphertext in the actual scheme, thus it is a valid ciphertext for the message \( M_v \) under the challenge set \( T \).

Otherwise, if \( u = 1 \), then \( Z = e(g, g)^s \) and \( \tilde{C} = M_v \cdot e(g, g)^v \). Since \( z \) is random, \( \tilde{C} \) will be a random element in \( G_T \) from the adversary’s view and contains no information about \( M_v \).

Query Phase 2: The simulator acts exactly as it did in query phase 1.

**Guess:** The adversary submits a guess \( v' \) of \( v \). If \( v' = v \), the simulator will output \( u' = 0 \) to indicate that it was given a valid BDH tuple. Otherwise, \( B \) will output \( u' = 1 \) to indicate it was given a random tuple.

As shown above, the generated public key and corrupted secret key are identical to that of the actual scheme.

In the case where \( u = 1 \), since \( \tilde{C} \) is randomized by \( e(g, g)^v \), the adversary gains no information about \( v \). Thus, we have \( \Pr[v' \neq v | u = 1] = (1/2) \). Since \( B \) outputs \( u' = 1 \) when \( v' \neq v \), we have \( \Pr[u' = u | u = 1] = (1/2) \).

If \( u = 0 \), the adversary sees the valid ciphertext of \( M_v \). The adversary’s advantage to distinguish the message is supposed to be \( \epsilon \). Thus, we have \( \Pr[v' \neq v | u = 0] = (1/2) + \epsilon \). Since \( B \) outputs \( u' = 0 \) if \( v' = v \), we have \( \Pr[u' = u | u = 0] = (1/2) + \epsilon \).

As a result, the overall advantage of the simulator in the DBDH game is \( (1/2) \Pr[u' = u | u = 0] + (1/2) \Pr[u' = u | u = 1] = (1/2)(1/2) + (1/2)((1/2) + \epsilon) = (1/2) + \epsilon \).
A. Algorithm Constructions of OP-PUN-ENC

The KeyGen and Encrypt algorithms in OP-PUN-ENC are exactly the same with that in P-PUN-ENC.

\( \text{Puncture}(PK, SK_{k-1}, \mathcal{AP}_k) \rightarrow (SK_k, rk_k) \): In this algorithm, \( sk_0^k \) is computed as that in P-PUN-ENC, and a random number \( z_k \in Z_p^* \) is selected to randomize the secret key components related to the leaf nodes of the access policy \( \mathcal{AP}_k \). Specifically, for each leaf node \( t_{k,j} \in LN(\mathcal{AP}_k) \), it selects a random value \( r_{k,j} \in Z_p \) and computes

\[
r_{k,j} = (i_{k,j}, K_{k,j} = 2^{(i_{k,j}+r_{k,j})}/z_k)
\]

\[L_{k,j} = g^{i_{k,j}/z_k}, Q_{k,j} = Q(i_{k,j})^{r_{k,j}/z_k}.\]

Finally, the random secret key related to the access policy \( \mathcal{AP}_k \) is generated as

\[rk_k = \{PP_k, \mathcal{AP}_k, \{rk_{k,j} \}_{t_{k,j} \in LN(\mathcal{AP}_k)}\}\]

Correspondingly, the local secret key is

\[SK_k = \{sk_0^k, z_1, \ldots, z_{k-1}, z_k\} \]

OutDecryt(\( CT, RK_k \)) \( \rightarrow CT'/\bot \): Here, \( RK_k = \{rk_1, \ldots, rk_k\} \). Concretely, the algorithm first checks whether \( CT \) has been punctured or not, i.e., whether the tag set \( T \) of \( CT \) satisfies the compositive puncture policy \( \mathcal{PP}_1 \) or \( \mathcal{PP}_2 \) or \( \cdots \) or \( \mathcal{PP}_k \) associated with the random secret keys \( RK_k \). If \( CT \) has not been punctured, it can be decrypted partially as follows.

For \( i = 1, \ldots, k \) and each node \( t_{i,j} \in LN(\mathcal{AP}_i) \cap t_{i,j} \notin T \), it first calculates a set of Lagrange coefficients \( w_1, \ldots, w_d, w^* \) such that \( \sum_{m=1}^d (w_m \cdot q(t_m)) + w^* \cdot q(t_{i,j}) = q(0) = \beta \), and then computes

\[Z_{i,j}' = \frac{e(K_{i,j}, D)}{e(L_{i,j}, \prod_{m=d}^d (E_m)^{w_m}) \cdot e(Q_{i,j}, D)^{w^*}} \]

\[e(g, g)^{\alpha i_{i,j} + r_{i,j}} \]

\[e(g, g)^{\beta i_{i,j} + r_{i,j}} \]

Further, for each \( i = 1, \ldots, k \), it combines \( Z_{i,j}' \) \( t_{i,j} \in LN(\mathcal{AP}_i) \cap t_{i,j} \notin T \) according to \( \mathcal{AP}_i \) from the leaf nodes to the root node by using the Lagrange polynomial interpolation technology recursively, such that \( Z_{i,j}' = e(g, g)^{\beta i_{i,j}/z_i} \) can be recovered.

Finally, the partially decrypted ciphertext is generated as

\[CT' = \{T, C, D, \{E_i\}_{i=1,d}, \{Z_i\}_{i=1,k,1}\}\]

FinalDecryt(\( CT', SK_k \)) \( \rightarrow M \): The algorithm first parses the term \( sk_0^k \) in \( SK_k \) as \( \{t_0, K_0, L_0, Q_0\} \), and selects a set of Lagrange coefficients \( w_1, \ldots, w_d, w^* \) such that \( \sum_{m=1}^d (w_m \cdot q(t_m)) + w^* \cdot q(t_0) = q(0) = \beta \). Then, it computes

\[Z_0 = \frac{e(K_0, D)}{e(L_0, \prod_{m=d}^d (E_m)^{w_m}) \cdot e(Q_0, D)^{w^*}} \]

\[e(g, g)^{\beta x(r_0 + \cdots + r_k - \lambda_1 - \cdots - \lambda_k)} \]

In addition, for \( i = 1, \ldots, k \), it computes \( Z_i = Z_i'' = e(g, g)^{\beta i_{i,j}/z_i} \). Finally, the plain data is obtained by computing \( M = C_0 / \prod_{i=0}^{k-1} Z_i \).

B. System Operations

The system initialization and data encryption phases are the same with the basic scheme. In the phase of data deletion, DO calls the Puncture algorithm, and uploads the generated \( rk_k \) to CS and replaces \( SK_{k-1} \) with the new generated \( SK_k \) locally. The decryption is divided into two steps: outsourced decryption by CS and final decryption by DO. In outsourced decryption, CS executes the OutDecryt(\( CT, RK_k \)) algorithm and returns \( CT' \) to DO. In final decryption, DO decrypts the received \( CT' \) through the FinalDecryt(\( CT', SK_k \)) algorithm to obtain the plain data \( M \).

C. Property Discussion

1) Security: With only the random secret keys, CS is not able to recover the data. Because there exists a random value \( z_i \) in the exponent of \( Z_i \). In addition, based on the security of P-PUN-ENC, the deleted ciphertext, i.e., the ciphertext satisfying the puncture policies, cannot be accessed even in the event that the local secret key of DO is compromised by CS. Thus, data confidentiality and fine-grained data deletion can be achieved in our improved scheme.

2) Efficiency: In our improved scheme, the secret key components related to the puncture policies are outsourced to CS. DO only needs to store the secret key components related to the tag \( t_0 \) and a random value \( z_i \in Z_p^* \) for each puncture policy. Thus, the storage cost for DO is reduced significantly and it will increase by only one element in \( Z_p \) for each puncture policy. With regard to the computation cost, since CS has the random secret keys, most of the pairing operations can be completed by CS. DO only needs to pair the key components related to \( t_0 \) with the ciphertext components and do some exponentiation operations on the pairing results obtained from CS. Considering that pairing is the most expensive operation, the computation cost for DO is greatly reduced and grows very slowly with the number of punctures.

VII. COMPREHENSIVE COMPARISON

In this section, we present a comprehensive comparison with some existing data deletion schemes to highlight the merits of our proposed scheme. The comparison result is shown in Table I.

Underlying Scheme: Most of the existing schemes are based on the monotonic CP-ABE or KP-ABE. The scheme in [34] supports generic symmetric encryption, which will incur huge key distribution and management cost in cloud-based IoT environments due to the large number of IoT devices. Inspired
TABLE I
COMPREHENSIVE COMPARISONS WITH EXISTING DATA DELETION SCHEMES

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Underlying scheme</th>
<th>Deletion mode</th>
<th>Third party</th>
<th>Decryption outsource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yu et al. [11]</td>
<td>CP-ABE (Monotonic) [22]</td>
<td>File-based</td>
<td>Required</td>
<td>No</td>
</tr>
<tr>
<td>Xue et al. [12]</td>
<td>KP-ABE (Monotonic) [18]</td>
<td>File-based</td>
<td>Required</td>
<td>No</td>
</tr>
<tr>
<td>Green and Miers [17]</td>
<td>KP-ABE (Non-monotonic) [24]</td>
<td>Tag-based</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Tang et al. [13]</td>
<td>CP-ABE (Monotonic) [33]</td>
<td>Policy-based (restricted)</td>
<td>Required</td>
<td>No</td>
</tr>
<tr>
<td>Xiong et al. [14]</td>
<td>KP-ABE (Monotonic) [18]</td>
<td>Time-based</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Cachin et al. [34]</td>
<td>Symmetric encryption (generic)</td>
<td>Policy-based (expressive)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>P-PUN-ENC</td>
<td>KP-ABE (Non-monotonic) [24]</td>
<td>Policy-based (expressive)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>OP-PUN-ENC</td>
<td>KP-ABE (Non-monotonic) [24]</td>
<td>Policy-based (expressive)</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

TABLE II
STORAGE OVERHEAD OF P-PUN-ENC AND OP-PUN-ENC

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Public Key</th>
<th>Initial Secret Key</th>
<th>Ciphertext</th>
<th>Punctured Secret Key</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(d + 3)</td>
<td>G</td>
<td>+</td>
<td>Zp</td>
</tr>
<tr>
<td>OP-PUN-ENC</td>
<td>(d + 3)</td>
<td>G</td>
<td>+</td>
<td>Zp</td>
</tr>
</tbody>
</table>

by [17], our proposed schemes are also constructed on the basis of the nonmonotonic KP-ABE.

Deletion Mode: Both [11] and [12] support file-based data deletion, which means the DO is able to delete one unique file each time. Green and Miers [17] only allowed one tag in each deletion, such that the data including the punctured tag are permanently and securely deleted. Xiong et al. [14] enabled the sensitive data to be securely self-destructed after a user-specified expiration time, but deletion in their scheme is only related to the time and should be predetermined when data is encrypted. Tang et al. [13] supported restricted policy-based data deletion with only one or two-level Boolean expressions. Cachin et al. [34] and our proposed schemes enable the DO to define arbitrary policies, including AND, OR, and threshold gates to determine the kind of data to be deleted, thus fine-grained data deletion is achieved.

Third Party: Both [11] and [12] required a third party (fog node or CS) to re-encrypt the ciphertext to guarantee that it cannot be decrypted anymore. Tang et al. [13] relied on a set of trusted key managers to remove the private control key related to a policy. The deletion in other schemes can be done without interaction with a third party.

Decryption Outsource: Only in our improved OP-PUN-ENC scheme, the key storage and decryption operations are partially outsourced to the CS to save local cost.

In conclusion, compared with existing works, our proposed schemes enable the DO to reliably and independently delete the outsourced IoT-driven data in a policy-based and self-controlled way, thus they can better meet the practical data deletion requirements in cloud-based IoT.

VIII. PERFORMANCE EVALUATION

A. Numerical Analysis

We give a detailed numerical analysis of the algorithms in our schemes on computation and storage overheads. Notations are clarified as follows.

2) d: The number of tags in ciphertext.
3) k: The number of punctures related to the current secret key.
4) n: The number of tags (leaf nodes) in each puncture policy.\(^1\)
5) E, P, M: The overhead caused by the operations of exponentiation, paring, and multiplication in G and G.

In Table II, we show the sizes of public key, initial secret key, ciphertext, and punctured secret key (i.e., storage overhead). Except for the punctured secret key, the storage overheads in P-PUN-ENC and OP-PUN-ENC are the same. It can be seen that both the sizes of public key and ciphertext depend on the number of tags defined in the phase of system initialization. The size of the initial secret key is just 3|G| + |Zp|, and the key storage cost for DO incurred by each puncture in P-PUN-ENC is about n(3|G| + |Zp|). As a result, after multiple punctures, the key storage cost may become a burden for DO. While in OP-PUN-ENC, most of the keys are outsourced to CS, and the key storage cost for DO increases only by |Zp| with each puncture, which makes it quite efficient.

Table III shows the number of operations required in the algorithms of KeyGen, Encrypt, Puncture, and Decrypt (i.e., computation cost). Some computation cost related to the Lagrange interpolation is ignored as it can be precomputed only once. The computation cost in KeyGen and Encrypt is totally determined by the number of tags d. During each puncture, few operations are related to the first part sk of the secret key, and the rest are about computing the new key components for the puncture policy, which mainly depends on the number of tags n in the puncture policy.

1\(^1\)Note that T-PUN-ENC can be seen as a special case of P-PUN-ENC with n = 1. The comparative result of the proposed P-PUN-ENC with T-PUN-ENC is left out in this section, since it can be simply derived by letting n = 1.
### TABLE III

**Computation Cost of the Algorithms in P-PUN-ENC and OP-PUN-ENC**

<table>
<thead>
<tr>
<th>Schemes</th>
<th>KeyGen</th>
<th>Encrypt</th>
<th>Puncture</th>
<th>Decrypt</th>
<th>DO</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-PUN-ENC</td>
<td>((d + 6)E)</td>
<td>((d + 2)E + P + M)</td>
<td>(3(E + M) + 3nE)</td>
<td>0</td>
<td>(\leq nk((d + 1)(M + E) + 3P))</td>
</tr>
<tr>
<td>OP-PUN-ENC</td>
<td>((d + 6)E)</td>
<td>((d + 2)E + P + M)</td>
<td>(3(E + M) + 3nE)</td>
<td>(\leq nk((d + 1)(M + E) + 3P))</td>
<td>(3P + M + (d + k + 1)(M + E))</td>
</tr>
</tbody>
</table>

### TABLE IV

**Computational Complexity of the Algorithms in P-PUN-ENC and OP-PUN-ENC**

<table>
<thead>
<tr>
<th>Schemes</th>
<th>KeyGen</th>
<th>Encrypt</th>
<th>Puncture</th>
<th>Decrypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-PUN-ENC</td>
<td>(O(d))</td>
<td>(O(d))</td>
<td>(O(n))</td>
<td>(O(nkd))</td>
</tr>
<tr>
<td>OP-PUN-ENC</td>
<td>(O(d))</td>
<td>(O(d))</td>
<td>(O(n))</td>
<td>(O(d + k))</td>
</tr>
</tbody>
</table>

---

**Fig. 5.** Execution time of the KeyGen and Encrypt algorithms.

**Fig. 6.** Execution time of the Puncture algorithm.

Fig. 5 shows the execution time of the algorithms of KeyGen and Encrypt. Both the execution time of KeyGen and Encrypt are increasing with the number of tags \(d\) in the ciphertext. Note that, the encrypting operations with ten tags can be efficiently completed with 25 ms.

In the puncture process, we test the execution time with different number of tags \(n\) in each puncture policy, and only OR operations are considered for simplicity. Fig. 6 demonstrates that the running time of the Puncture algorithm is almost unaffected by the number of tags \(d\) in the ciphertext, but grows with the number of tags \(n\) involved in each puncture.

Figs. 7 and 8 show the execution time of the Decrypt algorithm in P-PUN-ENC and the FinalDecrypt algorithm in OP-PUN-ENC, respectively, which also represents the computation cost for DO to decrypt the ciphertext in the two schemes. The number of tags \(n\) included in each puncture policy is set as 1 to 4, and the number of punctures \(k\) is from 0 to 3, where \(k = 0\) means decrypting with the initial secret key. In both Figs. 7 and 8, the execution time is increasing with \(d\) and \(k\). The difference is that, with increased \(n\), the time of the Decrypt algorithm is growing, but the time of FinalDecrypt is almost not affected. In addition, comparing the results in Figs. 7 and 8 with the same \(n, k, \) and \(d\), the computation cost for DO in OP-PUN-ENC is significantly reduced, which is only 8 ms with \(d = 10, n = 4, \) and \(k = 3\). This is due to the outsource technique, which transfers most of the costly decrypting operations to CS.
IX. CONCLUSION

In this article, we have proposed a secure and fine-grained self-controlled outsourced data deletion scheme in cloud-based IoT, which allows DOs to precisely and permanently delete the IoT-driven data stored in cloud in a policy-based way without relying on a third party. To do so, we have presented the algorithm constructions of P-PUN-ENC as an extension of T-PUN-ENC. Specifically, we have designed a policy transform method based on the logical relationship between the puncture policy and access policy, and utilized a key delegation technique in ABE to complete the key update operations. In addition, to deal with the issue of growing key storage and decryption cost existing in T-PUN-ENC and P-PUN-ENC, we have proposed OP-PUN-ENC primitive by combining the puncture policy and access policy, and utilized a key delegation technique in ABE to complete the key update operations. We have formally confirmed the security properties and comprehensively compared our proposed scheme with some related works to demonstrate that it can better meet the data deletion requirements in cloud-based IoT environment. Finally, extensive numerical analysis and experiment simulations have been conducted to demonstrate the efficiency of our scheme in terms of computation and storage overheads. In the future work, we will consider more specific security challenges in the IoT environment and try to seek a tradeoff between flexible data sharing and secure deletion. Moreover, we will consider other application scenarios to make the proposed algorithms more practical.

REFERENCES


