FEASE: Fast and Expressive Asymmetric Searchable Encryption

Long Meng
University of Surrey
long.meng@surrey.ac.uk

Liqun Chen
University of Surrey
liqun.chen@surrey.ac.uk

Yangguang Tian
University of Surrey
yangguang.tian@surrey.ac.uk

Mark Manulis
Universität der Bundeswehr München
mark.manulis@unibw.de

Suhui Liu
Southeast University
230219091@seu.edu.cn

Abstract

Asymmetric Searchable Encryption (ASE) is a promising cryptographic mechanism that enables a semi-trusted cloud server to perform keyword searches over encrypted data for users. To be useful, an ASE scheme must support expressive search queries, which are expressed as conjunction, disjunction, or any Boolean formulas. In this paper, we propose a *fast* and *expressive* ASE scheme that is adaptively secure, called FEASE. It requires only 3 pairing operations for searching any conjunctive set of keywords independent of the set size and has linear complexity for encryption and trapdoor algorithms in the number of keywords.

FEASE is based on a new fast Anonymous Key-Policy Attribute-Based Encryption (A-KP-ABE) scheme as our first proposal, which is of independent interest. To address optional protection against keyword guessing attacks, we extend FEASE into the first *expressive* Public-Key Authenticated Encryption with Keyword Search (PAEKS) scheme.

We provide implementations and evaluate the performance of all three schemes, while also comparing them with the state of the art. We observe that FEASE outperforms all existing expressive ASE constructions and that our A-KP-ABE scheme offers anonymity with efficiency comparable to the currently fastest yet non-anonymous KP-ABE schemes FAME (ACM CCS 2017) and FABEO (ACM CCS 2022).

1 Introduction

Outsourcing data storage to third-party providers offers an efficient way for clients with limited resources or expertise to manage and disseminate large volumes of encrypted data. However, traditional public or private key encryption methods hinder the ability to selectively retrieve specific data segments. To address this limitation, Searchable Encryption (SE) emerges as a cryptographic solution [15, 52]. SE empowers a user to securely outsource data to a server in an encrypted form and perform search operations on the data without revealing the plaintext to the server. SE finds diverse applications including cloud storage, secure messaging and email, healthcare, finance, academic and research databases, Internet of Things security, and more.

SE can be classified into two categories: Symmetric Searchable Encryption (SSE) [26, 55] and Asymmetric Searchable Encryption (ASE) [13]. In SSE, a user employs a secret key to encrypt a set of documents and keywords and uploads the resulting ciphertext to a cloud server. Later, the same secret key is used to generate a trapdoor for a specific search query containing one or more keywords. This trapdoor is sent to the server, which matches it with the ciphertext and returns the searched documents. In ASE, the distinction lies in that a data sender encrypts the document and keywords by using a public key and a data receiver subsequently generates the trapdoor by using the corresponding secret key.

A related field known as "Stream Encryption with Pattern Matching" (SEPM) [16, 17, 22], has emerged in recent years. SEPM achieves functionalities similar to traditional SE but is tailored for searching patterns [1] within encrypted streams. This means that data senders only need to encrypt data streams, eliminating the need for encrypting a keyword set as indexes. SEPM schemes share a similar syntax with ASE schemes that are usually constructed in a public-key setting. They find valuable applications in fields such as deep packet inspection, genomic data, medical data analysis, and more.

It is widely recognized that SSE offers high efficiency and has been extensively studied for its dynamic capabilities [33], allowing for efficient addition and deletion of keywords or documents in the encrypted dataset. ASE simplifies key management, offering strong security arguments and flexibility that can be extended to facilitate fine-grained access control on data receivers [61]. SEPM shares similar advantages with ASE but stands out for its capabilities for searching patterns instead of exact keywords [2]. In this paper, our focus is on ASE schemes, which find practical applications in various fields such as cloud storage [44], email filtering [13], cloud-based healthcare [21], smart grids [60], etc.

[1] E.g., a pattern "ab**cd" means any 6-character string with the first two letters "ab" and the last two letters "cd".

[2] A comparison for these three fields is given in our full version [43].

When designing an ASE scheme, the expressiveness of keyword search queries is usually considered a critical aspect. A search query is called *expressive* if contained keywords can be represented as a conjunctive, disjunctive, or any monotonic Boolean formula. For example, in the email filtering use case [13], a typical search query may be: "(Sender: Tom **AND** Subject: Rent) **OR** Priority: Urgent", which asks the email server to return emails sent from Tom with the subject "Rent", or emails with an "urgent" priority.

The efficiency of an ASE scheme is also crucial in large-scale applications. For an ASE scheme to be efficient, the communication overhead *and* computational overhead must be small. For instance, in a cloud-based healthcare system [21], a slow ASE scheme can delay access to critical patient information, which can lead to serious consequences. In the email filtering use case [13], an inefficient ASE scheme can result in slower email retrieval time, the requirement of more storage space, and thus higher resource utilization and higher costs.

In the literature, we observe that most of the existing ASE schemes that support expressive search queries [21, 37, 42, 44, 53, 57] are derived from Anonymous Key-Policy Attribute-Based Encryption (A-KP-ABE) schemes. Informally, Attribute-Based Encryption (ABE) is a form of public-key encryption enabling fine-grained access control. In ABE, ciphertexts and secret keys are linked to sets of attributes, and access policies specify which attributes are required for decryption. Typically, ABE schemes use linear secret-sharing techniques [6] to support expressive access policies. In a Key-Policy ABE (KP-ABE) scheme, a data sender encrypts a message with an attribute set $\mathbb{S}$ to create a ciphertext ct, and each data receiver owns a secret key sk tied to an access structure $\mathbb{A}$. The decryption is successful only if $\mathbb{S}$ in ct satisfies $\mathbb{A}$ in sk. However, KP-ABE prioritizes message privacy over attribute privacy. This is inadequate for applications where attributes, such as those in healthcare and E-commerce, contain sensitive information. To address this, A-KP-ABE schemes are developed to conceal attribute information within the ciphertext. Due to the similarity in syntax, expressiveness, and security properties, an A-KP-ABE scheme can be transformed into an expressive ASE scheme by treating attributes as keywords [3].

Nevertheless, existing expressive ASE schemes [21, 34, 37, 42, 44, 53, 57] suffer from significant drawbacks. The scheme in [34], based on inner-product encryption, experiences a superpolynomial blowup in both ciphertext and trapdoor size. The schemes in [37] and [42], relying on bilinear pairings over composite-order groups, are highly inefficient. Expressive ASE schemes proposed in prime-order groups, such as [21, 44, 53, 57], offer better efficiency than [37, 42]. Unfortunately, these schemes suffer from either insecure constructions that are vulnerable to attacks or intricate designs leading to inefficiency [4]. These limitations severely restrict their practicality in real-world applications. Given these challenges, a natural question arises: *Can we construct a fast and expressive ASE scheme by initially constructing a fast and expressive A-KP-ABE scheme?*

Continuing our research on ASE security. Typically, an ASE scheme is designed to achieve semantic security, protecting the privacy of the keyword sets encrypted within the ciphertext. This property, referred to as Indistinguishability against Chosen Keyword Attacks (IND-CKA), is foundational. However, this property does not guarantee the confidentiality of keywords in a trapdoor. Research has revealed a vulnerability in ASE schemes known as Keyword Guessing Attacks (KGA) [18]. In this scenario, a cloud server acting as an adversary can generate ciphertext for every possible keyword and test if it matches a trapdoor. If the number of potential keywords is polynomially bounded, the adversary can deduce the keyword hidden in the trapdoor. Several approaches have been proposed to counter such attacks, such as fuzzy keyword search [58], designated server [49], dual server [19], registered keyword search [56], public-key authenticated keyword search [31], secure-channel free keyword search [5], etc.

Among these countermeasures, Public-key Authenticated Encryption with Keyword Search (PAEKS) [31] stands out as a promising technique. The fundamental concept behind PAEKS is to enable a data sender to encrypt keywords with his own secret key sk_s and a data receiver's public key pk_r, while a data receiver generates a trapdoor by using his own secret key sk_r and a data sender's public key pk_s. In this case, a PAEKS scheme is required to simultaneously achieve "Ciphertext Indistinguishability" [5], and "Trapdoor Indistinguishability (TI)", where the latter ensures that a trapdoor does not reveal any keyword value. Crucially, since the cloud server lacks access to the secret keys sk_r and sk_s, it is unable to generate ciphertext for keywords and test them, effectively preventing KGA. In the literature of PAEKS [20, 24, 31, 40, 41, 45, 47, 48], we observe that they only focus on supporting equality search queries and lack expressiveness. Motivated by the situation, our second question arises: *Can we construct a fast and expressive PAEKS scheme?*

Contributions. In summary, we have the following contributions in this paper:

- We introduce a fast and expressive A-KP-ABE scheme, serving as the foundation for our research and it is independent of interest.
- Our primary achievement is to transform our A-KP-ABE scheme into FEASE – a *Fast* and *Expressive* ASE scheme.
- Building upon FEASE, we further extend it to create the first expressive PAEKS scheme, which is secure under the state-of-the-art security model [6].
- Our three schemes share the following features:

[3] This is intuitive from the transformation from anonymous IBE to ASE supporting equality queries [2]. See Sec. 4.2 for details.

[4] Details of the literature are given in Sec. 2.1.

[5] CI in PAEKS is similar to IND-CKA in traditional ASE schemes.

[6] The literature of this model is reviewed in our full version [43]. The detail of this model is introduced in Sec. 5.3 in the PAEKS field.

1. They support expressive search queries (or access policies) that are conjunctive, disjunctive or any monotonic Boolean formulas.
2. They have a linear complexity for both communication and computational overhead in the encryption and trapdoor/key generation algorithm, and require only 3 pairing operations for searching/decrypting any conjunctive set of keywords/attributes independent of the set size.
3. They are constructed in the prime-order group with the efficient Type-III pairing.
4. They have no restrictions on the size of keywords (attributes) or policies and allow any arbitrary string to be used as a keyword (attribute).
5. They satisfy the adaptive security in the generic group model and random oracle model [7].

- The implementation results show that our three schemes have almost the same efficiency and achieve the best performance in their corresponding fields. We stress that our A-KP-ABE scheme is even comparable to state-of-the-art non-anonymous KP-ABE schemes FAME [3] and FABEO [50] in terms of their efficiency. For 100 keywords/attributes, our schemes run around 0.07s for encryption, 0.24s for trapdoor/key generation, and 0.012s for searching a conjunctive set of 100 keywords. Compared to FAME, our A-KP-ABE is 2 times faster for key generation and 4 times faster for encryption. Compared to FABEO, our A-KP-ABE is 0.7 times slower for key generation and 0.1 times slower for encryption. This shows that anonymity in KP-ABE can be achieved without noticeable degradation in efficiency.

2 Related work

In this section, we first review the literature on ASE, SSE, and SEPM and provide a comprehensive comparison between these three fields. Then we review PAEKS and A-KP-ABE schemes. Due to the page limit, we only present the literature of ASE field here. The rest is shown in our full version [43].

2.1 ASE schemes

The concept of ASE traces back to Boneh et al [13]. They started up the ASE research with the first construction. Subsequently, Abdalla et al. [2] formalized ASE consistency and explored the relationship between ASE and Anonymous Identity-Based Encryption (AIBE). Several ASE constructions based on different techniques were proposed later in [7,23,35]. These ASE schemes primarily supported equality search and lacked expressiveness. Advancements came with Park et al. [46] and Golle et al. [27], who introduced ASE schemes capable of handling conjunctive search queries. Hwan and Lee [32] enhanced these schemes, optimizing ciphertext and secret key sizes and extending the techniques to multi-user scenarios. Zhang et al. [59] studied the cases where the keyword numbers in search queries formed subsets of those in ciphertexts. Boneh and Waters [14] introduced a comprehensive framework for analyzing and constructing Searchable Public Key Encryption (S-PKE) schemes, a generalization of ASE, supporting diverse families of predicates and arbitrary conjunctions.

In 2008, Katz et al. [34] introduced the concept of Inner-Product Encryption (IPE) that paved the way for the construction of the first expressive ASE scheme capable of handling both conjunctive and disjunctive keyword queries. However, this solution faced a superpolynomial increase in both ciphertext and trapdoor sizes. Addressing this, Lai et al. [37] and Lv et al. [42] presented expressive ASE schemes, ensuring linear complexity in ciphertext size concerning the number of keywords, which is a significant improvement over the superpolynomial complexity. Nevertheless, their schemes relied on inefficient bilinear pairings over composite-order groups. Though there exist techniques [25] to convert pairing-based schemes from composite-order groups to prime-order groups, there is still a significant performance degradation due to the required size of the special vectors [51].

In 2016, Cui et al. [21] proposed the first expressive ASE scheme in the prime-order groups that significantly improves the performance over existing schemes and proves the selective security of their scheme in the standard model. After that, Meng et al. [44] improved the construction of [21] to achieve constant-size ciphertext and seven pairings in the search algorithm without depending on the number of keywords. However, their scheme has a quadratic trapdoor size $O(\ell^2+\ell)$ where ℓ represents the number of keywords in the keyword policy. Additionally, this scheme requires all keywords that appeared in the ciphertext must be a part of the search query, otherwise, the search will fail. These trade-offs hugely decrease the practicality of their scheme. In 2019, Shen et al. [53] proposed a generic transformation from an A-KP-ABE scheme to an expressive ASE scheme, then they proposed an A-KP-ABE scheme and transformed it into an expressive ASE scheme. Recently, Tseng et al. [57] proposed a fast A-KP-ABE scheme and transformed it into an expressive ASE scheme that achieves only two pairings in the search algorithm without depending on the number of keywords. Unfortunately, the A-KP-ABE schemes in [53] and [57] bring the construction of KP-ABE schemes from [51] and [30] respectively with only removing the exposed attributes in the ciphertext. Their constructions do not satisfy the anonymity of an A-KP-ABE scheme [8]. Therefore, [21] and [44] remain at the forefront of the expressive ASE field.

As shown in Table 1, we compare different features between our FEASE and PAEKS and other ASE schemes. For expres-

[7] The use of random oracle is fairly common in many cryptographic protocols such as Full Domain Hash signatures [8] and OAEP encryption [9].

[8] The reasons are the same as in FABEO, as introduced in Sec. 4.1.

Scheme	Expressiveness	Group	Pairing	KGA	Security	Universe	Efficiency
BCOP04 [13]	AND	Prime	Type I	No	Full, RO	Large	-
KSW08 [34]	AND, OR	Composite	-	No	Sel., STD	Small	*
LZDLC13 [37]	AND, OR	Composite	-	No	Full, STD	Small	**
LHZF14 [42]	AND, OR, NOT	Composite	-	No	Full, STD	Small	**
CWDWL16 [21]	AND, OR	Prime	Type I	Partial	Sel., STD	Large	***
MZNLHS17 [44]	AND, OR	Prime	Type III	Partial	Sel., STD	Large	***
FEASE	AND, OR	Prime	Type III	No	Full, GGM & RO	Large	****
Our PAEKS	AND, OR	Prime	Type III	Yes	Full, GGM & RO	Large	****

Table 1: A property-wise comparison of the various ASE schemes for different features. "KGA" represents the security against the keyword guessing attack, "RO" stands for "Random Oracle", "STD" stands for "Standard Model", "GGM" stands for "Generic Group Model". The more number of "*", the better the efficiency (lower running time and communication overhead).

siveness, [13] is the first ASE scheme and it supports only equality queries, other schemes support at least any monotonic Boolean formulas [9]. For the bilinear pairing group, [34,37,42] are using composite-order groups, and other schemes are built on the prime-order group. For pairing type, [13,21] are using the Type I pairing, and other schemes are using the faster Type III pairing. For stronger security requirements, [21,44] can partially protect against KGA since they allow a designated server to perform KGA, and our PAEKS can fully prevent the KGA from any cloud server. For the security model, [13] is fully secure under the random oracle model. [37,42] satisfies full security under the standard model, while [21,34,44] are selectively secure in the standard model. Our FEASE and PAEKS are fully secure under the generic group model and random oracle model. For the restriction of keyword space, except from [34,36,42], all other schemes support a large universe of keywords and hence do not need to restrict the number of keywords in the system. Finally, we rate the level of efficiency of all the expressive ASE schemes in a qualitative way.

3 Preliminaries

In this section, we define the notation, access structures, monotone span programs for providing expressiveness, the partially hidden structure, and hardness assumptions.

3.1 Notation

For integers m, n where $m < n$, $[m, n]$ denotes the set $m, m+1, ..., n$. For $m = 1$, we simply write $[n]$. For a prime p, let $\mathbb{Z}_p$ denote the set $[0, ..., p-1]$ where addition and multiplication are computed modulo p. The set $\mathbb{Z}_p^*$ is same as $\mathbb{Z}_p$ but with 0 removed. Let λ denote the security parameter.

For a set S, $s \xleftarrow{\$} S$ denotes that s is sampled uniformly and independently at random from S. $y \leftarrow \mathcal{A}(x)$ denotes that y is the output of running algorithm $\mathcal{A}$ on input x with uniformly random bits. An adversary is a probabilistic algorithm. A probabilistic algorithm is called probabilistic polynomial time (PPT) if its running time is bounded by some polynomial in the length of its input.

We use bold letters to denote vectors and matrices, with the former in lowercase and the latter in uppercase. By default, a vector $\mathbf{v}$ is treated as a column vector. $\mathbf{v}_k$ denotes the k-th element of $\mathbf{v}$ and $\|$ denotes concatenation of vectors. M_i and $\mathrm{M}_{i,j}$ denote the i-th row and the (i, j)-th element of a matrix M, respectively. We use M^{T} for the transpose of M.

3.2 Access structures

In this paper, *access structures* and *attribute sets*, *keyword policy* and *keyword set* are defined in the same way. Below we only provide the definition of the former set of terms.

Definition 1. *Definition 2.1 (Access structure). If $\mathcal{U}$ denotes the universe of attributes, then an access structure $\mathbb{A}$ is a collection of non-empty subsets of $\mathcal{U}$, i.e., $\mathbb{A} \subseteq 2^{\mathcal{U}} \setminus \{0\}$. It is called monotone if for every $B, C \subseteq \mathcal{U}$ such that $B \subseteq C$, $B \in \mathbb{A} \Rightarrow C \in \mathbb{A}$.*

Monotone means that an authorized user who acquires more attributes will not lose any privileges. A (monotone) Boolean formula consists of **AND** and **OR** gates, where each input is associated with an attribute in $\mathcal{U}$. A set of attributes $S \subseteq \mathcal{U}$ satisfies a Boolean formula if we set all inputs of the formula that map to an attribute in S to true and the others to false and the formula evaluates to true.

Monotone span programs (MSP) (or linear secret sharing schemes [6]) are a more general class of functions and include Boolean formulas. We encode an access structure by a policy (M, π) where M of size $\ell \times n$ over $\mathbb{Z}_p$ and a general mapping function $\pi : \{1, ..., \ell\} \to \mathcal{U}$. In [39], Lewko and Waters describe a simple and efficient method to convert any (monotone) Boolean formula F into an MSP(M, π) such that every row of M corresponds to input in F and the number of columns is same as the number of **AND** gates in F. Furthermore, each entry in M is either a 0, 1, or -1.

Let $\mathbb{S} = \{u_i\}_{i \in [m]} \subseteq \mathcal{U}$ be a set of m attributes and $I = \{i \,|\, i \in \{1, ..., \ell\}, \pi(i) \in \mathbb{S}\}$ be the set of rows in M that belong to $\mathbb{S}$. We say that (M, π) accepts $\mathbb{S}$ if there exists

[9] [42] supports non-monotonic queries (e.g., ''NOT gate'') as well.

a linear combination of rows in I that gives $(1,\ 0, ...,\ 0)$. This means, there exist constants $\gamma_i \in \mathbb{Z}_p$ for $i \in I$ such that $\sum_{i\in I}\gamma_i \mathbf{M}_i = (1,\ 0,\ ...,\ 0)$. These constants can be computed in time polynomial in the size of M. It is worth noting that if Lewko and Water's method is applied to Boolean formulas, then it is always possible to pick coefficients that are either 0 or 1 for the resulting MSPs, irrespective of the set $\mathbb{S}$.

3.3 Partially hidden structures

We apply the partially hidden structure for an attribute set (keyword set) and an access policy (keyword policy) for our proposed schemes. This structure is firstly proposed in [36] for an anonymous CP-ABE and then applied in the expressive ASE schemes [21,44]. Taking A-KP-ABE as an example, the structure works as follows: Each attribute is divided into a generic attribute name and an attribute value. The attribute values used in both the secret key and ciphertext are not disclosed to the cloud server, whereas a partially hidden access structure and attribute set with only attribute names are included in a secret key and ciphertext respectively. The decryption algorithm matches the attribute names first and then decrypts the ciphertext by testing if the attribute values match.

More specifically, we define an attribute set $\mathbb{S} = \{u_i\}_{i\in[m]}$ has m attributes with each attribute belonging to a different category. Let n_i and v_i denote the attribute name and attribute value of an attribute u_i respectively, i.e., $u_i = \{n_i,\ v_i\}$. We express an access policy as $\mathbb{A} = (\mathrm{M},\ \pi,\ \pi(i))$, where M is a $\ell \times n$ share-generating matrix, M_i denotes the i^{th} row of M, π is a mapping function from M_i to an attribute $\pi(i)$. Let $n_{\pi(i)}$ and $v_{\pi(i)}$ denote the attribute name and attribute value of attribute $\pi(i)$ respectively, i.e., $\pi(i) = \{n_{\pi(i)}, v_{\pi(i)}\}$. In our schemes, the attribute values $v_{\pi(i)}$ of an access policy $(\mathrm{M},\ \pi,\ \{\pi(i)\})$ and the attribute values v_i of an attribute set $\mathbb{S}$ are not exposed in the ciphertext or secret key, while $(\mathrm{M},\ \pi,\ n_{\pi(i)})$ and attribute names n_i are disclosed.

Using our notation, a user's attribute set $\mathbb{S} = \{u_i\} = \{n_i, v_i\}$ satisfies an access policy $(\mathrm{M},\ \pi,\ \pi(i) = \{n_{\pi(i)},\ v_{\pi(i)}\})$ if and only if there exists $I \subseteq \{1, ..., \ell\}$ and constants $\{\gamma_i\}_{i\in I}$ such that

$$\sum_{i\in I}\gamma_i \mathrm{M}_i = (1, 0, ..., 0) \text{ and } \pi(i) = x_i \text{ for } \forall i \in I.$$

Similar to the scheme in [36], our schemes have the restriction that each attribute name can only be used once in an access policy. We can obtain a partially hidden access structure where attribute names are used multiple times (up to a constant number of uses fixed at setup) from a one-use scheme by applying the generic transformation given in [38]. While the transformation does incur some cost in key size, it does not increase the size of the ciphertext.

3.4 Bilinear maps

Bilinear maps. Let GroupGen be a PPT algorithm that takes as input a security parameter 1^λ and outputs a set of group parameters $\mathsf{par} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, where p is a prime of $\Theta(\lambda)$ bits, $\mathbb{G}_1$,$\mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of order p, g_1 and g_2 are the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear mapping that satisfies the following properties:

- Computable: Given $g_1 \in \mathbb{G}_1,\ g_2 \in \mathbb{G}_2$, there is a polynomial time algorithm to compute $e(g_1, g_2) \in \mathbb{G}_T$.
- Bilinear: For all $g_1 \in \mathbb{G}_1,\ g_2 \in \mathbb{G}_2$ and any integers $a, b \in \mathbb{Z}_p$, we have $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.
- Non-Degenerate: There exists $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ such that $e(g_1, g_2) \neq 1$.

In this work, we only consider asymmetric (or Type-III) pairing groups where there exists no efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. Remark that while our constructions are using bilinear maps the security of our constructions will be proven to hold in the generic group model rather than under standard assumptions in pairing-based cryptography.

4 Our proposed schemes

Our research begins with FABEO [50], the fastest KP-ABE scheme known for its linear complexity in key size and ciphertext size, and a constant 2 pairing operations in the decryption process. As shown in Fig. 1, our design strategy unfolds in stages. Initially, we transform from the FABEO KP-ABE scheme into an A-KP-ABE scheme as a solid foundation. Subsequently, this A-KP-ABE scheme serves as the basis for creating FEASE as our primary achievement. Building upon FEASE, we extend its capabilities to craft the first expressive PAEKS scheme. Notably, all our schemes maintain the same level of expressiveness and efficiency as FABEO, inheriting the strengths of its construction. In this section, we guide you through the step-by-step evolution of our designs, starting from FABEO and progressing through each scheme outlined in our roadmap.

4.1 Transform from FABEO KP-ABE into anonymous KP-ABE

First, we show the syntax of an (anonymous) KP-ABE scheme. A KP-ABE (or A-KP-ABE) scheme consists of the following algorithms:

- $(\mathsf{pk},\ \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$. The setup algorithm Setup is run by a key generation center (KGC). The algorithm takes as input a security parameter 1^λ. It outputs a system public key pk and a master secret key msk.
- $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{pk}, \mathsf{msk}, \mathbb{A})$. The key generation algorithm KeyGen is run by the KGC. The algorithm takes as input a public key pk, a master secret key msk, and an access structure $\mathbb{A}$. It outputs a secret key sk.

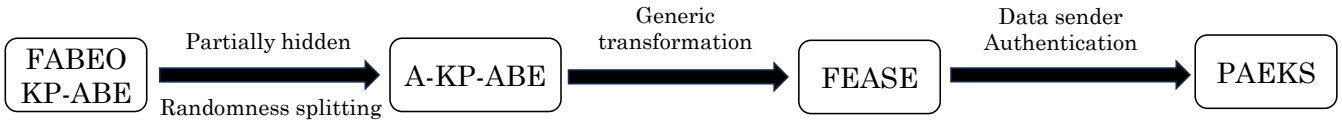


Figure 1: The technical roadmap of our proposed schemes. The texts on the arrows indicate the main techniques we used for the transformation from the left scheme to the right one.

- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathbb{S}, \mathsf{msg})$. The encryption algorithm Enc is run by the data sender. The algorithm takes as input a public key pk, a set of attributes $\mathbb{S}$, and a message msg. It outputs a ciphertext ct.
- $\mathsf{msg}/\perp \leftarrow \mathsf{Dec}(\mathsf{ct}, \mathsf{sk})$. The decryption algorithm Dec is run by the data receiver. The algorithm takes as input a ciphertext ct associated with an attribute set $\mathbb{S}$ and a message msg, and a secret key sk associated with an access policy $\mathbb{A}$. It outputs the message msg if $\mathbb{S}$ satisfies $\mathbb{A}$, or outputs a $\perp$ otherwise.

The concrete construction of FABEO KP-ABE scheme [50] is presented in Fig. 2. Note that the $\pi(i)$ is the attribute in $\mathbb{A}$ and u_i is the attribute in $\mathbb{S}$, following the notation defined in Sec. 3.2. Besides, the r value in sk_1 would have been a vector $\mathbf{r}'$, and the original version should be $\mathsf{sk}_{1,j} = g_2^{\mathbf{r}'[j]}$ where $j \in [\tau]$ indicates the number of attribute re-use. In this paper, we simplify it and let $j = 1$ since it is easier for further illustrations.

Why the FABEO KP-ABE does not ensure anonymity? In terms of the anonymity of an A-KP-ABE scheme defined in the Sec. 5.1, a close inspection of the FABEO construction reveals two fundamental issues:

1. Exposed attribute set: The ciphertext in FABEO includes the exposed attribute set $\mathbb{S}$ as an element, making it directly accessible to potential attackers.
2. Attribute guessing attack: Even if the exposed attributes are removed from the ciphertext, anonymity is not guaranteed. Specifically, when provided with two attributes, u_0 and u_1, and a ciphertext $(\mathsf{ct}_{1,u_b}, \mathsf{ct}_2)$ where $b \in \{0, 1\}$, attackers can determine b from the equation $e(\mathsf{ct}_{1,u_b}, g_2) = e(\mathsf{H}(u_b), \mathsf{ct}_2)$.

To address the above vulnerabilities, we apply the following techniques to transform FABEO KP-ABE into an A-KP-ABE scheme. The A-KP-ABE construction is shown in Fig. 3. We highlight the differences between the two schemes in red fonts.

Partially hidden structure. To consider how to conceal the exposed attribute set, the choice of the privacy level becomes pivotal. If the objective is to safeguard the complete privacy of the attribute set without any information leakage, the existing technique, Inner Product Encryption (IPE) by Katz et al. [34], is an option. However, it suffers from a significant drawback: a super-polynomial increase in both ciphertext and trapdoor size, making it highly inefficient. Considering our goal of developing a fast ASE scheme, a more viable alternative is the widely used

$\underline{(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)}$.
Run $\mathsf{GroupGen}(1^\lambda)$ to obtain the group parameters $\mathsf{par} := (p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2)$. Pick $\alpha \xleftarrow{\$} \mathbb{Z}_p$ and a hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$. Compute the public key pk and master secret key msk as

$$\mathsf{pk} = (\mathsf{par}, \mathsf{H}, e(g_1, g_2)^\alpha), \mathsf{msk} = \alpha.$$

$\underline{\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{pk}, \mathsf{msk}, \mathbb{A} = (\mathbf{M}, \pi, \{\pi(i)\}_{i\in[\ell]}))}$.
Pick $r \xleftarrow{\$} \mathbb{Z}_p^\tau, \mathbf{v} \xleftarrow{\$} \mathbb{Z}_p^{n-1}$. Compute

$$\mathsf{sk}_1 = g_2^r, \mathsf{sk}_{2,i} = g_1^{\mathbf{M}_i(\alpha \| \mathbf{v})^\top} \cdot \mathsf{H}(\pi(i))^r.$$

Output $\mathsf{sk} = (\mathbb{A}, \mathsf{sk}_1, \{\mathsf{sk}_{2,i}\}_{i\in[\ell]})$.

$\underline{\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathbb{S} = \{u_i\}_{i\in[m]}, \mathsf{msg})}$. Pick $s \xleftarrow{\$} \mathbb{Z}_p$. Compute

$$\mathsf{ct}_{1,i} = \mathsf{H}(u_i)^s, \mathsf{ct}_2 = g_2^s, \mathsf{ct}_3 = e(g_1, g_2)^{\alpha s} \cdot \mathsf{msg}.$$

Output $\mathsf{ct} = (\mathbb{S}, \{\mathsf{ct}_{1,i}\}_{i\in[m]}, \mathsf{ct}_2, \mathsf{ct}_3)$.

$\underline{\mathsf{msg}/\perp \leftarrow \mathsf{Dec}(\mathsf{ct}, \mathsf{sk})}$.
If $\mathbb{S}$ satisfies $\mathbb{A}$, there exists constants $\{\gamma_i\}_{i\in I}$ s.t. $\sum_{i\in I}\gamma_i \mathbf{M}_i = (1, 0, ..., 0)$ and reconstruct the message msg by computing:

$$\mathsf{msg} = \frac{e(\prod_{i\in I}(\mathsf{ct}_{1,\pi(i)})^{\gamma_i}, \mathsf{sk}_1) \cdot \mathsf{ct}_3}{e(\prod_{i\in I}(\mathsf{sk}_{2,i})^{\gamma_i}, \mathsf{ct}_2)}.$$

Figure 2: The FABEO KP-ABE scheme.

method known as the "partially hidden structure", illustrated in Sec. 3.3. The essence of this structure lies in the division of each attribute into a generic attribute name and an attribute value. While the attribute values remain undisclosed in both the private key and ciphertext, a partially hidden access structure and attribute set expose only the attribute names. For instance, considering an access structure like "(Sender: Tom **AND** Subject: Rent) **OR** Priority: Urgent" and an attribute set "[Sender: Bob, Subject: Meeting, Priority: Medium]", the partially hidden access structure becomes "Sender **AND** (Subject **OR** Priority)" and the partially hidden attribute set is "[Sender, Subject, Priority]". During decryption, the algorithm first matches the attribute names and then tests if the attribute values match.

As highlighted in Fig. 3, each attribute $\pi(i)$ in an access

structure $\mathbb{A}$ is separated into a name $n_{\pi(i)}$ and a value $v_{\pi(i)}$, in which $n_{\pi(i)}$ is exposed with (M, π) in sk. Similarly, each attribute u_i in an attribute set $\mathbb{S}$ is separated into a name n_i and a value v_i, in which n_i is disclosed in ct.

This technique, although leaking a certain level of information (i.e., attribute names), provides high efficiency. Attribute names, being less sensitive than attribute values, allow for efficient matching without involving pairing or exponentiation operations. This efficiency improvement is critical, enabling a fast location of specific attribute values under corresponding names, thereby significantly enhancing decryption efficiency.

$(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$.

Run $\mathsf{GroupGen}(1^\lambda)$ to obtain the group parameters $\mathsf{par} := (p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2)$. Pick $\alpha, b_1, b_2 \xleftarrow{\$} \mathbb{Z}_p$ and a hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$. Compute the public key pk and master secret key msk as

$$\mathsf{pk} = (\mathsf{par}, \mathsf{H}, g_2^{b_1}, g_2^{b_2}, e(g_1, g_2)^\alpha), \mathsf{msk} = (\alpha, b_1, b_2).$$

$\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{pk}, \mathsf{msk}, \mathbb{A} = (\mathsf{M}, \pi, \{\pi(i)\}_{i\in[\ell]})$.

Remind that $\{\pi(i)\}_{i\in[\ell]} = \{n_{\pi(i)}, v_{\pi(i)}\}_{i\in[\ell]}$. Pick $r \xleftarrow{\$} \mathbb{Z}_p$, $\mathbf{v} \xleftarrow{\$} \mathbb{Z}_p^{n-1}$. Compute $\mathsf{sk}_1 = g_2^r$,

$$\mathsf{sk}_{2,i} = (g_1^{\mathsf{M}_i(\alpha\|\mathbf{v})^\top} \cdot \mathsf{H}(\pi(i))^r)^{\frac{1}{b_1}}, \mathsf{sk}_{3,i} = (g_1^{\mathsf{M}_i(\alpha\|\mathbf{v})^\top} \cdot \mathsf{H}(\pi(i))^r)^{\frac{1}{b_2}}.$$

Output $\mathsf{sk} = ((\mathsf{M}, \pi, \{n_{\pi(i)}\}_{i\in[\ell]}), \mathsf{sk}_1, \{\mathsf{sk}_{2,i}, \mathsf{sk}_{3,i}\}_{i\in[\ell]})$.

$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathbb{S} = \{u_i\}_{i\in[m]} = \{n_i, v_i\}_{i\in[m]}, \mathsf{msg})$.

Pick $s_1, s_2 \xleftarrow{\$} \mathbb{Z}_p$, let $s = s_1 + s_2$. Compute

$$\mathsf{ct}_{1,i} = \mathsf{H}(u_i)^s, \mathsf{ct}_2 = g_2^{b_1 s_1}, \mathsf{ct}_3 = g_2^{b_2 s_2}, \mathsf{ct}_4 = e(g_1, g_2)^{\alpha s} \cdot \mathsf{msg}.$$

Output $\mathsf{ct} = (\{n_i\}_{i\in[m]}, \{\mathsf{ct}_{1,i}\}_{i\in[m]}, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{ct}_4)$.

$\mathsf{msg}/\perp \leftarrow \mathsf{Dec}(\mathsf{ct}, \mathsf{sk})$.

Tests if there is any subset I that matches the attribute names $\{n_i\}_{i\in[m]}$ in ct with $(\mathsf{M}, \pi, \{n_{\pi(i)}\}_{i\in[\ell]})$ in sk. If not, return $\perp$. Otherwise, it finds constants $\{\gamma_i\}_{i\in I}$ s.t. $\sum_{i\in I} \gamma_i \mathsf{M}_i = (1, 0, ..., 0)$ and reconstruct the message msg by computing:

$$\mathsf{msg} = \frac{e\left(\prod_{i\in I}(\mathsf{sk}_{2,i})^{\gamma_i}, \mathsf{ct}_2\right) \cdot e\left(\prod_{i\in I}(\mathsf{sk}_{3,i})^{\gamma_i}, \mathsf{ct}_3\right)}{e\left(\prod_{i\in I}(\mathsf{ct}_{1,\pi(i)})^{\gamma_i}, \mathsf{sk}_1\right) \cdot \mathsf{ct}_4}.$$

If the equation holds, return 1. Otherwise, continue to find another subset of I and repeat the checking. If the above equation does not hold for all subsets, return 0.

Figure 3: Our A-KP-ABE scheme.

Randomness splitting technique. We observe that the attribute guessing attack is available as $\mathsf{ct}_{1,u}$ and ct_2 sharing the same randomness s. To counter this problem, we introduce a technique to split the randomness in the ciphertext. Specifically, we divide the randomness s into two distinct components $s_1, s_2 \in \mathbb{Z}_p$ and let $s = s_1 + s_2$. As highlighted in Fig. 3, the ciphertext components are now structured as $\mathsf{ct}_{1,i} = \mathsf{H}(u_i)^s$, $\mathsf{ct}_2 = g_2^{b_1 s_1}$, and $\mathsf{ct}_3 = g_2^{b_2 s_2}$, where $g_2^{b_1}$ and $g_2^{b_2}$ are parts of the public key. At the secret key side, to recover s and eliminate the b_1, b_2 terms, the secret key element $g_1^{\mathsf{M}_i(\alpha\|\mathbf{v})^\top} \cdot \mathsf{H}(\pi(i))^r$ is doubling and exponentiation by $\frac{1}{b_1}$ and $\frac{1}{b_2}$ separately. By correctness, ct_4 remains the same as the ct_3 in FABEO. In this case, given two attributes, u_0 and u_1, and a ciphertext $(\mathsf{ct}_{1,u_b}, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{ct}_4)$ where $b \in \{0,1\}$, an attacker who owns $g_2^{b_1}$ and $g_2^{b_2}$ can no longer discern the attribute u_b. Consequently, the ciphertext successfully conceals the attribute value.

4.2 FEASE: A Fast and Expressive ASE scheme

In this section, we demonstrate how to convert the A-KP-ABE scheme proposed in Sec. 4.1 into the FEASE, which is our main research target. We first introduce the syntax of an expressive ASE scheme, which includes the following four algorithms:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. The key generation algorithm KeyGen is run by the data receiver. The algorithm takes as input a security parameter 1^λ. It outputs a public key pk and a secret key sk.
- $\mathsf{td} \leftarrow \mathsf{Trap}(\mathsf{pk}, \mathsf{sk}, \mathbb{P})$. The trapdoor generation algorithm Trap is run by the data receiver. The algorithm takes as input a public key pk, a secret key sk, and a keyword policy structure $\mathbb{P}$. It outputs a trapdoor td.
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathbb{W})$. The encryption algorithm Enc is run by the data sender. The algorithm takes as input a public key pk, and a set of keywords $\mathbb{W}$. It outputs a ciphertext ct.
- $1/0 \leftarrow \mathsf{Search}(\mathsf{ct}, \mathsf{td})$. The search algorithm Search is run by the cloud server. The algorithm takes as input a keyword ciphertext ct and a trapdoor td. It outputs a bit 1 if the search is successful, or a bit 0 if the search is failed.

In addition to the keyword ciphertext described in our syntax, the data owner also encrypts documents using a public-key encryption scheme. However, it is essential to highlight that our focus in this paper is solely on the encryption of keywords.

It is easy to see that an A-KP-ABE scheme shares a similar syntax as an expressive ASE scheme. In A-KP-ABE, a ciphertext can only be decrypted with a secret key if the attributes in the ciphertext satisfy the policy in the key. In expressive ASE, keywords can only be searched if the keywords in the ciphertext satisfy the policy associated with the trapdoor. Besides, the semantic security of ASE (IND-CKA) (defined in Sec. 5.2) aligns with the same level of security as the anonymity in A-KP-ABE (defined in Sec. 5.1). Therefore, we can convert our A-KP-ABE to FEASE by using the following generic transformation:

Generic transformation from A-KP-ABE to expressive ASE. An ASE scheme $\mathsf{ASE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Trap}, \mathsf{Search})$ can be constructed from an A-KP-ABE scheme $\mathsf{A\text{-}KP\text{-}ABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ by the following steps:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{ASE.KeyGen}(1^\lambda)$. On input of a security parameter 1^λ, this algorithm executes as follows: (1) Run $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{A\text{-}KP\text{-}ABE.Setup}(1^\lambda)$. (2) Set $\mathsf{sk} \leftarrow \mathsf{msk}$ and output $(\mathsf{pk}, \mathsf{sk})$.

- $\mathsf{td} \leftarrow \mathsf{ASE.Trap}(\mathsf{pk}, \mathsf{sk}, \mathbb{P})$. On input of pk, sk and a keyword policy $\mathbb{P}$, this algorithm executes as follows: (1) Run $\mathsf{sk} \leftarrow \mathsf{A\text{-}KP\text{-}ABE.KeyGen}(\mathsf{pk}, \mathsf{sk}, \mathbb{P})$. (2) Set $\mathsf{td} \leftarrow \mathsf{sk}$ and output td.

- $\mathsf{ct} \leftarrow \mathsf{ASE.Enc}(\mathsf{pk}, \mathbb{W})$. On input of pk and a set of keywords $\mathbb{W}$, this algorithm executes as follows: (1) Set a message $\mathsf{msg} = 1$. (2) Run $\mathsf{ct} \leftarrow \mathsf{A\text{-}KP\text{-}ABE.Enc}(\mathsf{pk}, \mathbb{W}, \mathsf{msg})$ and output ct.

- $1/0 \leftarrow \mathsf{ASE.Search}(\mathsf{ct}, \mathsf{td})$. On input of td and ct, this algorithm executes as follows: (1) Set $\mathsf{sk} \leftarrow \mathsf{td}$. Run $\mathsf{msg}' \leftarrow \mathsf{A\text{-}KP\text{-}ABE.Dec}(\mathsf{ct}, \mathsf{sk})$. (2) If $\mathsf{msg}' = 1$, the algorithm outputs 1, else 0.

Based on this transformation, the resulting construction of FEASE is shown in Fig. 4. We highlight the differences between FEASE and our A-KP-ABE scheme. Note that $n_{\pi(i)}$ and $v_{\pi(i)}$ represent the keyword name and value in the keyword policy $\mathbb{P}$ respectively. n_i and v_i are the keyword name and value in the keyword set $\mathbb{W}$ respectively. We can see that FEASE's construction mirrors our A-KP-ABE scheme by treating attributes as keywords and setting the message as a known value.

4.3 Fast and Expressive PAEKS scheme

After obtaining FEASE, we extend it into the first expressive PAEKS scheme that has security against Keyword Guessing Attack (KGA). Traditional ASE schemes cannot resist KGA because 1) the data sender encrypts the keyword with only a data receiver's public key. The cloud server can generate ciphertext and exhaustively test the keywords within an existing trapdoor, and 2) the trapdoor does guarantee keyword privacy: Given two keyword policies, a cloud server can discern a trapdoor is generated from which policy. Thus, to defend against KGA, the following security requirements should be satisfied:

1. The cloud server is not capable of matching an existing trapdoor by generating a ciphertext.

2. Ciphertext Indistinguishability (CI): A ciphertext should not reveal the keyword values in the keyword set.

3. Trapdoor Indistinguishability (TI): A trapdoor should not reveal the keyword values in the keyword policies.

$\underline{(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)}$.

Run $\mathsf{GroupGen}(1^\lambda)$ to obtain the group parameters $\mathsf{par} := (p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2)$. Pick $\alpha, b_1, b_2 \xleftarrow{\$} \mathbb{Z}_p$ and a hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$. Compute the public key and secret key as

$$\mathsf{pk} = (\mathsf{par}, \mathsf{H}, g_2^{b_1}, g_2^{b_2}, e(g_1, g_2)^\alpha), \mathsf{sk} = (\alpha, b_1, b_2)$$

$\underline{\mathsf{td} \leftarrow \mathsf{Trap}(\mathsf{pk}, \mathsf{sk}, \mathbb{P} = (\mathrm{M}, \pi, \{\pi(i)\}_{i\in[\ell]}))}$.

Remind that $\{\pi(i)\}_{i\in[\ell]} = \{n_{\pi(i)}, v_{\pi(i)}\}_{i\in[\ell]}$. Pick $r \xleftarrow{\$} \mathbb{Z}_p$, $\mathbf{v} \xleftarrow{\$} \mathbb{Z}_p^{n-1}$. Compute $\mathsf{td}_1 = g_2^r$

$$\mathsf{td}_{2,i} = (g_1^{\mathrm{M}_i(\alpha\|\mathbf{v})^\top} \cdot \mathsf{H}(\pi(i))^r)^{\frac{1}{b_1}}, \mathsf{td}_{3,i} = (g_1^{\mathrm{M}_i(\alpha\|\mathbf{v})^\top} \cdot \mathsf{H}(\pi(i))^r)^{\frac{1}{b_2}}$$

Output $\mathsf{td} = ((\mathrm{M}, \pi, \{n_{\pi(i)}\}_{i\in[\ell]}), \mathsf{td}_1, \{\mathsf{td}_{2,i}, \mathsf{td}_{3,i}\}_{i\in[\ell]})$.

$\underline{\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathbb{W} = \{w_i\}_{i\in[m]} = \{n_i, v_i\}_{i\in[m]})}$.

Pick $s_1, s_2 \xleftarrow{\$} \mathbb{Z}_p$, let $s = s_1 + s_2$. Compute

$$\mathsf{ct}_{1,i} = \mathsf{H}(w_i)^s, \mathsf{ct}_2 = g_2^{b_1 s_1}, \mathsf{ct}_3 = g_2^{b_2 s_2}, \mathsf{ct}_4 = e(g_1, g_2)^{\alpha s}$$

Output $\mathsf{ct} = (\{n_i\}_{i\in[m]}, \{\mathsf{ct}_{1,i}\}_{i\in[m]}, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{ct}_4)$.

$\underline{1/0 \leftarrow \mathsf{Search}(\mathsf{ct}, \mathsf{td})}$.

Tests if there is any subset I that matches the keyword names $\{n_i\}_{i\in[m]}$ in ct with $(\mathrm{M}, \pi, \{n_{\pi(i)}\}_{i\in[\ell]})$ in td. If not, return 0. Otherwise, it finds constants $\{\gamma_i\}_{i\in I}$ s.t. $\sum_{i\in I}\gamma_i \mathrm{M}_i = (1, 0, ..., 0)$ and computes:

$$\mathsf{ct}_4 = \frac{e\left(\prod_{i\in I}(\mathsf{td}_{2,i})^{\gamma_i}, \mathsf{ct}_2\right) \cdot e\left(\prod_{i\in I}(\mathsf{td}_{3,i})^{\gamma_i}, \mathsf{ct}_3\right)}{e\left(\prod_{i\in I}(\mathsf{ct}_{1,\pi(i)})^{\gamma_i}, \mathsf{td}_1\right)}.$$

If the equation holds, return 1. Otherwise, the cloud server continues to find another subset of I and repeats the checking. If the above equation does not hold for all subsets, return 0.

Figure 4: Our FEASE scheme.

The formal definitions of CI and TI are defined in the Sec. 5.3. In a PAEKS scheme, the data sender is allowed to have a public key and a secret key. The keywords are encrypted by using the data sender's secret key and the data receiver's public key. The trapdoor is generated by using data receiver's secret key and the data sender's public key. Since the cloud server does not hold any of the secret keys, it cannot generate a ciphertext to match with an existing trapdoor. Borrowing the semantic security of FEASE, it is feasible to preserve CI in the PAEKS. Thus, our main target is to develop TI in our PAEKS design.

The syntax of an expressive PAEKS scheme is defined with the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$. The Setup algorithm is run by a trusted party. The algorithm takes as input a security parameter 1^λ. It outputs the global public parameter pp.

- $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_s(1^\lambda)$. The KeyGen_s algorithm is run by a data sender. This algorithm takes as input a security parameter 1^λ. It outputs the sender's public key pk_s and secret key sk_s.

- $(\mathsf{pk}_r, \mathsf{sk}_r) \leftarrow \mathsf{KeyGen}_r(1^\lambda)$. The KeyGen_r algorithm is run by a data receiver. This algorithm takes as input a security parameter 1^λ. It outputs the receiver's public key pk_r and secret key sk_r.

- $\mathsf{td} \leftarrow \mathsf{Trap}(\mathsf{pp}, \mathsf{pk}_s, \mathsf{sk}_r, \mathbb{P})$. The trapdoor generation algorithm Trap is run by the data receiver. The algorithm takes as input the public parameter pp, the sender public key pk_s, the receiver secret key sk_r, and a keyword policy structure $\mathbb{P}$. It outputs a trapdoor td.

- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}_r, \mathsf{sk}_s, \mathbb{W})$. The encryption algorithm Enc is run by the data sender. The algorithm takes as input the public parameter pp, the receiver public key pk_r, the sender secret key sk_s, and a set of keywords $\mathbb{W}$. It outputs a keyword ciphertext ct.

- $1/0 \leftarrow \mathsf{Search}(\mathsf{ct}, \mathsf{td})$. The search algorithm Search is run by the cloud server. The algorithm takes as input a keyword ciphertext and a trapdoor td. It outputs a bit 1 if the search is successful, or a bit 0 if the search is failed.

Data sender authentication. We first analyze why the trapdoor construction of FEASE does not guarantee keyword privacy. Considering the TI security model defined in the Sec. 5.3: Given two keyword policies that contain keywords $\pi(i)_0$ and $\pi(i)_1$ separately, and a trapdoor $(\mathsf{td}_1, \mathsf{td}_{2,\pi(i)_b}, \mathsf{td}_{3,\pi(i)_b})$ where $b \in \{0, 1\}$, a cloud server can test the keywords $\pi(i)_0$ and $\pi(i)_1$ separately into the equation $e(\prod_{i\in I}(\mathsf{td}_{2,i})^{\gamma_i}, g_2^{b_1}) = e(g_1, g_2)^\alpha \cdot e(\prod_{i\in I}\mathsf{H}(\pi(i)_b)^{\gamma_i}, \mathsf{td}_1)$ where the γ_i can be easily calculated since there is no policy for a single keyword. Even to distinguish two sets of keywords, the server can try different policies together with γ_i and hash values. Since the number of keywords and policies is polynomially bounded, the server can discern the keywords hidden in the trapdoor.

The concrete construction of our PAEKS scheme is shown in Fig. 5. We highlight the difference between the PAEKS and FEASE. Our idea to extend from FEASE to PAEKS is to embed a data sender's secret key $\frac{1}{c} \in \mathbb{Z}_p$ in the keywords term to be distinguished in the ciphertext: $\mathsf{H}(w_i)^{\frac{s}{c}}$, and eliminate $\frac{1}{c}$ by the corresponding pairing element g_2^{cr} in trapdoor, where g_2^c is the data sender's public key. The reasons are listed as follows:

- Multiplying a random number $\frac{1}{c}$ to s does not affect the construction for the ciphertext, thus the CI security is inherited from the IND-CKA security of FEASE.

$\underline{\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)}$. Run $\mathsf{GroupGen}(1^\lambda)$ to obtain group parameters $\mathsf{par} := (p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2)$. Pick a hash function $\mathsf{H}: \{0,1\}^* \to \mathbb{G}_1$. The global public parameter is

$$\mathsf{pp} = (\mathsf{par}, \mathsf{H}).$$

$\underline{(\mathsf{pk}_r, \mathsf{sk}_r) \leftarrow \mathsf{KeyGen}_r(1^\lambda)}$. Pick $\alpha, b_1, b_2 \leftarrow \mathbb{Z}_p$. Compute

$$\mathsf{pk}_r = (g_2^{b_1}, g_2^{b_2}, e(g_1, g_2)^\alpha), \mathsf{sk}_r = (\alpha, b_1, b_2).$$

$\underline{(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_s(1^\lambda)}$. Pick $c \leftarrow \mathbb{Z}_p$. Compute

$$\mathsf{pk}_s = g_2^c, \mathsf{sk}_s = c.$$

$\underline{\mathsf{td} \leftarrow \mathsf{Trap}(\mathsf{pp}, \mathsf{pk}_s, \mathsf{sk}_r, \mathbb{P} = (\mathrm{M}, \pi, \{\pi(i)\}_{i\in[\ell]}))}$.

Remind that $\{\pi(i)\}_{i\in[\ell]} = \{n_{\pi(i)}, v_{\pi(i)}\}_{i\in[\ell]}$. Pick $r \xleftarrow{\$} \mathbb{Z}_p$, $\mathbf{v} \xleftarrow{\$} \mathbb{Z}_p^{n-1}$. Compute $\mathsf{td}_1 = g_2^{cr}$

$$\mathsf{td}_{2,i} = (g_1^{\mathrm{M}_i(\alpha\|\mathbf{v})^\top} \cdot \mathsf{H}(\pi(i))^r)^{\frac{1}{b_1}}, \mathsf{td}_{3,i} = (g_1^{\mathrm{M}_i(\alpha\|\mathbf{v})^\top} \cdot \mathsf{H}(\pi(i))^r)^{\frac{1}{b_2}}.$$

Output $\mathsf{td} = ((\mathrm{M}, \pi, \{n_{\pi(i)}\}_{i\in[\ell]}), \mathsf{td}_1, \{\mathsf{td}_{2,i}, \mathsf{td}_{3,i}\}_{i\in[\ell]})$

$\underline{\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}_r, \mathsf{sk}_s, \mathbb{W} = \{w_i\}_{i\in[m]} = \{n_i, v_i\}_{i\in[m]})}$.

Pick $s_1, s_2 \xleftarrow{\$} \mathbb{Z}_p$, let $s = s_1 + s_2$. Compute

$$\mathsf{ct}_{1,i} = \mathsf{H}(w_i)^{\frac{s}{c}}, \mathsf{ct}_2 = g_2^{b_1 s_1}, \mathsf{ct}_3 = g_2^{b_2 s_2}, \mathsf{ct}_4 = e(g_1, g_2)^{\alpha s}$$

Output $\mathsf{ct} = (\{n_i\}_{i\in[m]}, \{\mathsf{ct}_{1,i}\}_{i\in[m]}, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{ct}_4)$.

$\underline{1/0 \leftarrow \mathsf{Search}(\mathsf{ct}, \mathsf{td})}$.

Tests if there is any subset I that matches the keyword names $\{n_i\}_{i\in[m]}$ in ct with $(\mathrm{M}, \pi, \{n_{\pi(i)}\}_{i\in[\ell]})$ in td. If not, return 0. Otherwise, it finds constants $\{\gamma_i\}_{i\in I}$ s.t. $\sum_{i\in I}\gamma_i \mathrm{M}_i = (1, 0, ..., 0)$ and computes:

$$\mathsf{ct}_4 = \frac{e\left(\prod_{i\in I}(\mathsf{td}_{2,i})^{\gamma_i}, \mathsf{ct}_2\right) \cdot e\left(\prod_{i\in I}(\mathsf{td}_{3,i})^{\gamma_i}, \mathsf{ct}_3\right)}{e\left(\prod_{i\in I}(\mathsf{ct}_{1,\pi(i)})^{\gamma_i}, \mathsf{td}_1\right)}.$$

If the equation holds, return 1. Otherwise, the cloud server continues to find another subset of I and repeats the checking. If the above equation does not hold for all subsets, return 0.

Figure 5: Our PAEKS scheme.

- g_2^{cr} is a single element that is not related to the number of keywords, thus the efficiency of trapdoor generation remains almost the same as FEASE.

- After changing the term from g_2^r to g_2^{cr}, the cloud server is not capable of attacking the trapdoor since it only has the knowledge of g_2^c and g_2^{cr} instead of g_2^r.

5 Security definitions

In this section, we introduce the formal security definitions and models of expressive A-KP-ABE, ASE, and PAEKS schemes.

5.1 Security definitions of A-KP-ABE

The security model for an A-KP-ABE scheme with a partially hidden structure addresses the property that a ciphertext does not reveal any information about the encrypted message, which we call "Indistinguishability against Chosen Plaintext Attack (IND-CPA)" security and that a ciphertext does not reveal any information about the encrypted attribute set, which we call "Anonymity (Anon)".

IND-CPA Security. We model the adaptive IND-CPA security in a game $\prod$ that is running between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows:

- **Setup.** $\mathcal{C}$ runs $\mathsf{Setup}(1^\lambda)$ to obtain a public key pk and a master secret key msk. It sends pk to the adversary and keeps msk secret.
- **Phase 1.** $\mathcal{A}$ issues queries to a key generation oracle for polynomial many times:
  - Key generation oracle: Given an access structure $\mathbb{A}$, the oracle generates a secret key $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{pk}, \mathsf{msk}, \mathbb{A})$ and returns sk to $\mathcal{A}$.
- **Challenge.** $\mathcal{A}$ outputs a challenge attribute set $\mathbb{S}^*$ and two equal-length messages msg_0^*, msg_1^* with the restriction that $\mathbb{S}^*$ cannot satisfy any access structure $\mathbb{A}$ that has been queried in Phase 1. Then $\mathcal{C}$ selects a random bit $b \in \{0, 1\}$, runs the algorithm $\mathsf{ct}_b^* \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathbb{S}^*, \mathsf{msg}_b^*)$ and returns the challenge ciphertext ct_b^* to $\mathcal{A}$.
- **Phase 2.** Same as Phase 1 with the restriction that any input access structure $\mathbb{A}$ cannot be satisfied by $\mathbb{S}^*$.
- **Guess.** $\mathcal{A}$ outputs $b' \in \{0, 1\}$ and wins the game if $b' = b$.

An A-KP-ABE scheme is adaptively IND-CPA secure if the advantage function refers to the security game $\prod$

$$\mathsf{Adv}_{\Pi,\mathcal{A}}^{\mathsf{CPA}}(\lambda) = \left| \Pr[b' = b] - \frac{1}{2} \right|$$

is negligible in security parameter λ for any PPT adversary $\mathcal{A}$.

Anonymity. Then we model the anonymity property in a game $\prod$ that is running between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows:

- **Setup.** $\mathcal{C}$ runs $\mathsf{Setup}(1^\lambda)$ to obtain a public key pk and a master secret key msk. It gives pk to adversary $\mathcal{A}$ and keeps msk secret.
- **Phase 1.** The adversary $\mathcal{A}$ issues queries to a key generation oracle for polynomial many times:
  - Key generation oracle: Given an access structure $\mathbb{A}$, the oracle generates a secret key $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{pk}, \mathsf{msk}, \mathbb{A})$ and returns sk to $\mathcal{A}$.
- **Challenge.** $\mathcal{A}$ outputs a message msg^* and two equal-size attribute sets $\mathbb{S}_0^* = \{n_i, v_{i0}\}_{i\in[m]}$, $\mathbb{S}_1^* = \{n_i, v_{i1}\}_{i\in[m]}$ with the restriction that $\mathbb{S}_0^*$, $\mathbb{S}_1^*$ have the same attribute names $\{n_i\}_{i\in[m]}$ and neither of them satisfies any access structure $\mathbb{A}$ that has been queried in Phase 1. $\mathcal{C}$ selects a random bit $b \in \{0, 1\}$, runs the algorithm $\mathsf{ct}_b^* \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathbb{S}_b^*, \mathsf{msg}^*)$ and returns the challenge ciphertext ct_b^* to $\mathcal{A}$.
- **Phase 2.** Same as Phase 1 with the restriction that any input access structure $\mathbb{A}$ cannot be satisfied by $\mathbb{S}_0^*$ and $\mathbb{S}_1^*$.
- **Guess.** $\mathcal{A}$ outputs $b' \in \{0, 1\}$ and wins the game if $b' = b$.

An A-KP-ABE scheme is anonymous if the advantage function refers to the security game $\prod$

$$\mathsf{Adv}_{\Pi,\mathcal{A}}^{\mathsf{Anon}}(\lambda) = \left| \Pr[b' = b] - \frac{1}{2} \right|$$

is negligible in security parameter λ for any PPT adversary $\mathcal{A}$.

5.2 Security definitions of expressive ASE

The security model for an expressive ASE scheme with a partially hidden structure addresses the property that a ciphertext does not reveal any information about the keyword values, which we call Indistinguishability against Chosen Keyword Attacks (IND-CKA) defined here for keyword sets.

IND-CKA security. We model the adaptive IND-CKA security in a game $\prod$ that is running between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows:

- **Setup.** The challenger $\mathcal{C}$ runs $\mathsf{KeyGen}(1^\lambda)$ to obtain a public key pk and the a secret key sk. It gives the public key pk to adversary $\mathcal{A}$ and keeps sk to itself.
- **Phase 1.** The adversary $\mathcal{A}$ adaptively issues queries to a trapdoor oracle for polynomial many times:
  - Trapdoor oracle: Given a keyword policy structure $\mathbb{P}$, the oracle generates a trapdoor $\mathsf{td} \leftarrow \mathsf{Trap}(\mathsf{pk}, \mathsf{sk}, \mathbb{P})$ and returns td to $\mathcal{A}$.
- **Challenge.** $\mathcal{A}$ outputs two equal-size keyword sets $\mathbb{W}_0^* = \{n_i, v_{i0}\}_{i\in[m]}$, $\mathbb{W}_1^* = \{n_i, v_{i1}\}_{i\in[m]}$ with the restriction that $\mathbb{W}_0^*$, $\mathbb{W}_1^*$ have the same keyword names $\{n_i\}_{i\in[m]}$, and neither of them satisfies any trapdoor that has been queried in Phase 1. $\mathcal{C}$ selects a random bit $b \in \{0, 1\}$, runs the algorithm $\mathsf{ct}_b^* \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathbb{W}_b^*)$ and returns the challenge ciphertext ct_b^* to the $\mathcal{A}$.
- **Phase 2.** $\mathcal{A}$ continues to issue queries to the trapdoor oracle for polynomial times with the restriction that any $\mathbb{P}$ input by $\mathcal{A}$ cannot be satisfied by $\mathbb{W}_0^*$ and $\mathbb{W}_1^*$.

- **Guess.** $\mathcal{A}$ outputs its guess $b' \in \{0, 1\}$ and wins the game if $b' = b$.

An expressive ASE scheme is adaptively IND-CKA secure if the advantage function refers to the security game $\prod$

$$\mathrm{Adv}^{\mathsf{CKA}}_{\prod,\mathcal{A}}(\lambda) = \left| \Pr[b' = b] - \frac{1}{2} \right|$$

is negligible in security parameter λ for any PPT adversary $\mathcal{A}$.

5.3 Security definitions of expressive PAEKS

The security model for an expressive PAEKS scheme with a partially hidden structure addresses the property that a ciphertext does not reveal any information about the keyword values, which we call "Ciphertext Indistinguishability (CI)" and the property that a trapdoor does not reveal any information about the keyword policy values, which we call "Trapdoor Indistinguishability (TI)". CI and TI are defined in two separate games that are running between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. We refer to the state-of-the-art PAEKS security model proposed in [20], in which it addresses both the CI and TI in a multi-user, multi-challenge, and fully chosen setting. Intuitively, these terms indicate the following conditions in the CI/TI game:

1. Multi-user CI/TI: In the CI/ TI game, $\mathcal{A}$ can not only input a keyword set/keyword policy but also input a data receiver/data sender's public key [10] to the ciphertext/trapdoor oracle respectively.

2. Multi-challenge CI/TI: In the CI/TI game, $\mathcal{A}$ can choose two sets of keywords for challenge keyword sets/policies rather than only two single keywords respectively [11].

3. Fully-chosen CI/TI: In the CI/TI game, $\mathcal{A}$ can query ciphertext/trapdoor for the challenge keyword sets/keyword policies from the ciphertext/trapdoor oracle respectively.

Game 1: Ciphertext Indistinguishability

1. **Setup.** Given a security parameter λ, the challenger $\mathcal{C}$ generates the global system parameter pp. Then $\mathcal{C}$ generates a pair of sender's key $(\mathsf{pk}_s, \mathsf{sk}_s)$ and a pair of receiver's key $(\mathsf{pk}_r, \mathsf{sk}_r)$. It gives $(\mathsf{pp}, \mathsf{pk}_s, \mathsf{pk}_r)$ to the adversary $\mathcal{A}$.

2. **Phase 1.** $\mathcal{A}$ is allowed to adaptively issue queries to the following oracles for polynomial many times:

   - Trapdoor Oracle $O_T(\mathbb{P}, \mathsf{pk})$: Given a keyword policy structure $\mathbb{P}$ and a public key pk (not necessarily the sender's pk_s), the oracle computes a trapdoor $\mathsf{td} \leftarrow \mathsf{Trap}(\mathsf{sk}_r, \mathsf{pk}, \mathbb{P})$ and returns td to $\mathcal{A}$.
   - Ciphertext Oracle $O_C(\mathbb{W}, \mathsf{pk})$: Given a set of keywords $\mathbb{W}$ and a public key pk (not necessarily the receiver's pk_r), the oracle computes a ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}_s, \mathsf{pk}, \mathbb{W})$ and returns ct to $\mathcal{A}$.

3. **Challenge.** After Phase 1, $\mathcal{A}$ outputs two equal-size keyword sets $\mathbb{W}^*_0 = \{n_i, v_{i0}\}_{i\in[m]}$, $\mathbb{W}^*_1 = \{n_i, v_{i1}\}_{i\in[m]}$ with the restriction that $\mathbb{W}^*_0$, $\mathbb{W}^*_1$ have the same keyword names $\{n_i\}_{i\in[m]}$ and neither of them satisfies any trapdoor that has been queried for $O_T(\cdot, \mathsf{pk}_s)$ in Phase 1, and submits them to $\mathcal{C}$. $\mathcal{C}$ randomly chooses a bit $b \in \{0, 1\}$, computes $\mathsf{ct}^*_b \leftarrow \mathsf{Enc}(\mathsf{pk}_r, \mathsf{sk}_s, \mathbb{W}^*_b)$ and returns ct^*_b to $\mathcal{A}$.

4. **Phase 2.** $\mathcal{A}$ continues to issue queries to O_T and O_C as above, with the restriction that any trapdoor that is queried for $O_T(\cdot, \mathsf{pk}_s)$ should not be satisfied by $\mathbb{W}^*_0$ and $\mathbb{W}^*_1$.

5. **Guess.** $\mathcal{A}$ outputs $b' \in \{0, 1\}$ and wins the game if $b' = b$.

We define $\mathcal{A}$'s advantage of successfully distinguishing the ciphertext of PAEKS as

$$\mathrm{Adv}^{\mathrm{CI}}_{\mathcal{A}}(\lambda) = \left| \Pr[b' = b] - \frac{1}{2} \right|.$$

Definition 2. *A PAEKS scheme is fully CI secure if for any PPT adversary $\mathcal{A}$, $\mathrm{Adv}^{\mathrm{CI}}_{\mathcal{A}}(\lambda)$ is negligible for security parameter λ.*

Game 2: Trapdoor Indistinguishability

1. **Setup.** The challenger $\mathcal{C}$ generates pp, $(\mathsf{pk}_s, \mathsf{sk}_s)$ and $(\mathsf{pk}_r, \mathsf{sk}_r)$ as in Game 1. It then gives $(\mathsf{pp}, \mathsf{pk}_s, \mathsf{pk}_r)$ to the adversary $\mathcal{A}$.

2. **Phase 1.** $\mathcal{A}$ issues queries to oracles $O_T(\mathbb{P}, \mathsf{pk})$ and $O_C(\mathbb{W}, \mathsf{pk})$ as in Game 1.

3. **Challenge.** After Phase 1, $\mathcal{A}$ chooses two equal size keyword policies $\mathbb{P}^*_0 = (\mathrm{M}^*, \pi^*, \{n_{\pi^*(i)}, v_{\pi(i)0}\}_{i\in[\ell]})$, $\mathbb{P}^*_1 = (\mathrm{M}^*, \pi^*, \{n_{\pi^*(i)}, v_{\pi(i)1}\}_{i\in[\ell]})$ with the restriction that $\mathbb{P}^*_0, \mathbb{P}^*_1$ have the same $(\mathrm{M}^*, \pi^*, \{n_{\pi^*(i)}\}_{i\in[\ell]})$ and neither of them are satisfied by any ciphertext that has been queried to $O_C(\cdot, \mathsf{pk}_r)$ in Phase 1, and submits them to $\mathcal{C}$ as the challenge keywords. $\mathcal{C}$ randomly chooses a bit $b \in \{0, 1\}$, computes $\mathsf{td}^*_b \leftarrow \mathsf{Trap}(\mathsf{pk}_s, \mathsf{sk}_r, \mathbb{P}^*_b)$ and returns td^*_b to $\mathcal{A}$.

4. **Phase 2.** $\mathcal{A}$ continues to issue queries to O_T and O_C as above, with the restriction that any ciphertext that is queried for $O_C(\cdot, \mathsf{pk}_r)$ should not be satisfied by $\mathbb{P}^*_0$ and $\mathbb{P}^*_1$.

5. **Guess.** $\mathcal{A}$ outputs $b' \in \{0, 1\}$ and wins the game if $b' = b$.

We define $\mathcal{A}$'s advantage of successfully distinguishing the trapdoors of PAEKS as

$$\mathrm{Adv}^{\mathrm{TI}}_{\mathcal{A}}(\lambda) = \left| \Pr[b' = b] - \frac{1}{2} \right|.$$

Definition 3. *A PAEKS scheme is fully TI secure if for any PPT adversary $\mathcal{A}$, $\mathrm{Adv}^{\mathrm{TI}}_{\mathcal{A}}(\lambda)$ is negligible for security parameter λ.*

[10] The public key is not necessary to be the challenged data sender or receiver, it could be anyone's public key including $\mathcal{A}$.

[11] This setting is not naturally preserved in a PAEKS scheme that only supports equality queries. But for an expressive PAEKS, this is a default setting.

6 Security Analysis of our schemes

In this section, we prove the security of our A-KP-ABE, FEASE, and PAEKS schemes under the Generic Group Model (GGM) and random oracle model. The reasons we use the generic group model can be summarized into the following aspects:

- The GGM is widely utilized in practical applications due to its sufficient security, supported by a deep understanding of pairing curves and widespread adoption. In practice, there is no significant difference observed between standard assumptions like SXDH and GGM security [50]. Real-world systems are more vulnerable to issues such as side-channel attacks or poor security practices, which are beyond the scope of GGM.
- The most efficient ABE schemes, such as FABEO [50] and BSW [10], are proven under GGM. Our proposed schemes, aimed at efficiency, utilize the same proof technique as [50], making GGM the natural choice.
- GGM serves as the base technique for proving the security of many well-known static assumptions like Diffie-Hellman (DH), Bilinear Diffie-Hellman (BDH), and their variants [11, 12, 54]. This underscores GGM's role in security proofs within various cryptographic contexts.

Theorem 1. *Our A-KP-ABE scheme is adaptively IND-CPA secure under the generic group model by modeling the hash function* H *as a random oracle.*

The proof of Theorem 1 is shown in our full version [43].

Theorem 2. *Our A-KP-ABE scheme is anonymous under the generic group model by modeling the hash function* H *as a random oracle.*

The proof of Theorem 2 is shown in our full version [43].

Theorem 3. *FEASE is adaptively IND-CKA secure under the generic group model by modeling the hash function* H *as a random oracle.*

The proof of Theorem 3 is shown in our full version [43].

Theorem 4. *The proposed PAEKS scheme is fully CI secure under the generic group model by modeling the hash function* H *as a random oracle.*

The proof of Theorem 4 is shown in our full version [43].

Theorem 5. *The proposed PAEKS scheme is fully TI secure under the generic group model by modeling the hash function* H *as a random oracle.*

The proof of Theorem 5 is shown in our full version [43].

7 Implementation and performance

We first introduce the roadmap of our experiments. Then we analyze the performance of the expressive ASE and KP-ABE schemes. Finally, we conduct an additional set of experiments for larger datasets.

7.1 Implementation roadmap

We implement our FEASE, PAEKS, A-KP-ABE schemes and the most efficient expressive ASE and ABE schemes in Python 3.9.16 using the Charm 0.5 framework [4] and the MNT224 curve for pairings because it is the best Type-III curve in PBC, the default pairing library in Charm. All running times below were measured on a PC with a 3.59 GHz AMD Ryzen 5 3600 6-Core Processor and 16GB RAM. The implementation code is available on GitHub [1].

In particular, we compare our FEASE and PAEKS scheme with the most efficient expressive ASE schemes CWDWL16 [21] and MZNLHS17 [44] and compare our A-KP-ABE scheme with the most efficient KP-ABE schemes FAME [3] and FABEO [50] [12]. Among these schemes, only CWDWL16 is constructed on a symmetric setting. In order to compare their efficiency on the same level, we transfer the construction of CWDWL16 to the asymmetric setting (presented in Appendix A). We choose not to compare FEASE with former expressive ASE schemes [34, 37, 42] since they are all based on the inefficient composite order groups. According to the analysis in [25, 29], in terms of the pairing-friendly elliptic curves, prime order groups have a clear advantage in both parameter size and computational efficiency over composite order groups.

For the expressive ASE schemes, we first choose random words from the English vocabulary to form keyword names and randomly assign a positive integer between 1 - 100 as a keyword value to each keyword name. Thus, every keyword is in the format of "Department: 2", "Professional: 6", "Hospital: 7", etc. The keyword values are the input of the trapdoor and encryption algorithm and the keyword names are exposed. We ensure that the keyword names in every trapdoor are included in the ciphertext, i.e., the keyword names can always match regardless of the policy, but the keyword values are chosen randomly for the trapdoor and ciphertext side. i.e., the keyword values only have little probability to match. In this way, we can simulate the worst case that the search has to traverse every subset of the matched keyword names to maximize the search time. On the ciphertext side, we test the encryption for 10, 20, ..., 100 keywords. On the trapdoor side, we choose 10, 15, ..., 50 keywords and assign "AND" and "OR" gates between the keywords to form policies.

For the KP-ABE schemes, We separate it into two cases: for our A-KP-ABE scheme, we create the attribute set and access policies in the way as the keyword set and keyword policies as stated above. For FAME and FABEO KP-ABE

[12] The reasons are separately shown in Sec. 2.1 and our full version [43].

Groups	Choose	Multiply	Exp.	Hash	Pairing
$\mathbb{G}_1$	0.58	0.02	0.57	0.04	3.68
$\mathbb{G}_2$	4.32	0.28	4.37	10.85	
$\mathbb{G}_T$	-	0.05	0.96	-	

Table 2: Average time (in milliseconds) for various operations on MNT224 curve.

Scheme	Setup	KeyGen_s	KeyGen_r	KeyGen_c
CWDWL16 [21]	31.56	-	-	4.1
MZNLHS17 [44]	25.7	-	-	1.2
FEASE (Fig. 4)	19.44	-	-	-
PAEKS (Fig. 5)	9.33	4.2	11.34	-
FAME [3]	21.69	-	-	-
FABEO [50]	9.8	-	-	-
A-KP-ABE (Fig. 3)	19.1	-	-	-

Table 3: Setup time and sender/receiver/server key generation time for various schemes (in milliseconds).

schemes, the partially hidden structure is not needed since they do not aim to protect attribute privacy. Thus, their attributes are straightforward and their policies are to use AND gate between any attribute because all the attributes are then required for decryption. We test these schemes against policies and attribute sets of size 10, 20, ..., 100 since large policy sizes are quite likely in typical use cases [28].

For both the keyword policies and access policies, we first convert the policies into a Boolean formula and then to an MSP using Lewko-Waters' method [39] (see Sec. 3.2 for a detailed discussion) so that the matrix M has only entries in $\{0, 1, -1\}$ and the reconstruction coefficients $\{\gamma_i\}$ are always 0 or 1, which reduces the number of exponentiations.

We present the setup times for the expressive ASE schemes and KP-ABE schemes in Table 3. Then we show the running times for the expressive ASE schemes in Fig. 6 and for the KP-ABE schemes in Fig. 7. These results are supported by our theoretical overview in Table 4 and Table 5 which lists the number of multiplications and exponentiations for each group as well as the number of hashing and pairing operations for each scheme. Additionally, we provide the number of group elements of trapdoor/secret key and ciphertext in Table 6. A more detailed explanation of running times and sizes is shown below.

7.2 Basic operation and initializations

Table 2 lists the average time taken by various operations on MNT224 in milliseconds. We can see that operations on group $\mathbb{G}_2$ are much more expensive than on $\mathbb{G}_1$, in which it has 8 times for choosing an element and exponentiation, 14 times for multiplication, and 271 times for hashing. Pairing is also a relatively expensive operation that is close to the cost of exponentiation on $\mathbb{G}_2$. It is also important to note that the size of an element in $\mathbb{G}_2$ is 3 times that of $\mathbb{G}_1$.

Setup time. In Table 3, we show the time of setup, data sender/data receiver key generation (only used in our PAEKS scheme), and cloud server key generation (used in CWDWL16 and MZNLHS17) of the schemes listed in our evaluation. Since all schemes support large universes of keywords/attributes, all of these schemes have a constant setup time and user/server key generation time and are almost equally fast. In specific, the setup time of FEASE/PAEKS is a bit faster than CWDWL16 and MZNLHS17, and the setup time for our A-KP-ABE lies between FAME and FABEO.

7.3 Expressive ASE schemes

The running times for expressive ASE schemes are shown in Fig. 6. For encryption, MZNLHS17 and our FEASE/PAEKS have a very close time around $0.06 \sim 0.07$s for encrypting 100 keywords, in which CWDWL16 is nearly 7 times slower. This result can be supported by Table 4: Although FEASE/PAEKS has two more exponentiations in $\mathbb{G}_2$, MZNLHS17 has 6 more exponentiations and $m+2$ multiplications in $\mathbb{G}_1$ and CWDWL16 has much more of them.

For the trapdoor generation algorithm, FEASE/PAEKS has the fastest 0.03s for generating a trapdoor with 50 keywords, followed by 3.76s from CWDWL16, and the MZNLHS17 runs a very inefficient 53.75s for 50 keywords. In terms of Table 4, almost all the multiplications and exponentiations of CWDWL16 and MZNLHS17 are calculated on $\mathbb{G}_2$. Instead, FEASE/PAEKS has less number of them and they happened in $\mathbb{G}_1$. Besides, MZNLHS17 has a quadratic increasing time regarding keyword numbers in a trapdoor, in which the trade-off is brought from the target of constant-size ciphertext.

For the search algorithm, we can see from Fig. 6 (c) that, all the schemes are linear to the number of matched keyword names subset. For the special case that we implement (each subset contains only one keyword), FEASE/PAEKS runs the fastest 0.1s for 10 subsets which is 2 times faster than CWDWL16 and MZNLHS17. Then Fig. 6 (d) shows that FEASE/PAEKS is the only one that remains a constant time around 0.011s for searching no matter how many keywords are included in a subset, while CWDWL16 and MZNLHS has a linear increase. We can see this in Table 5. First of all, FEASE/PAEKS has the least number of multiplications in $\mathbb{G}_T$ and pairings. Second, the pairing number of CWDWL16 is related to x_2 - the total number of keywords needed for search, which is why the search time is both related to x_1 - the matched subset number, and the number of keywords in each subset. Instead, the pairing number of MZNLHS17 and FEASE/PAEKS is only related to x_1. However, MZNLHS17 has extra calculations to multiply 4 components for each trapdoor element in each subset, before carrying out the pairing operations. This calculation is on the costly $\mathbb{G}_2$ group and is linear to x_2, which incurs a linear increase for their search time. Note that FEASE/PAEKS also has a linear increase regarding x_2 for multiplications in $\mathbb{G}_1$, but as shown in Table 2, it only takes 0.02ms for each hence it does not impact the search time for a limited number of keywords.

Schemes	Trapdoor/Secret Key						Encryption					
	$\mathbb{G}_1$			$\mathbb{G}_2$			$\mathbb{G}_1$			$\mathbb{G}_2$		
	Mul	Exp	Hash	Mul	Exp	Hash	Mul	Exp	Hash	Mul	Exp	Hash
CWDWL16 [21]	-	1	-	3ℓ	$8\ell+1$	-	$2m$	$6m+2$	-	-	-	-
MZNLHS17 [44]	-	1	-	3ℓ	$4\ell^2+4\ell+1$	-	$m+2$	$m+6$	-	-	-	-
FEASE (Fig. 4)	2ℓ	4ℓ	ℓ	-	1	-	-	m	m	-	2	-
PAEKS (Fig. 5)	2ℓ	4ℓ	ℓ	-	1	-	-	m	m	-	2	-
FAME [3]	$9\ell n+3n$	$9\ell+3n$	$6\ell+6n$	-	3	-	$3m$	$6m$	$6m$	-	-	-
FABEO [50]	ℓ	2ℓ	ℓ	-	1	-	-	m	m	-	1	-
A-KP-ABE (Fig. 3)	2ℓ	4ℓ	ℓ	-	1	-	-	m	m	-	2	-

Table 4: Computational overhead for trapdoor/key generation and encryption between ASE (top)/KP-ABE (bottom) schemes. m denotes the number of keywords/attributes in the ciphertext, ℓ, and n are the number of rows and columns of the MSP matrix.

Schemes	Search/Decryption				
	$\mathbb{G}_1$	$\mathbb{G}_2$		$\mathbb{G}_T$	
	Mul	Mul	Exp	Mul	Pairing
CWDWL16 [21]	-	1	-	$5x_2$	$6x_2+1$
MZNLHS17 [44]	-	$7x_2-x_1+1$	-	5	$6x_1+1$
FEASE (Fig. 4)	$3x_2$	-	-	2	$3x_1$
PAEKS (Fig. 5)	$3x_2$	-	-	2	$3x_1$
FAME [3]	$6x_2$	-	-	6	6
FABEO [50]	$2x_2$	-	-	2	2
A-KP-ABE (Fig. 3)	$3x_2$	-	-	3	$3x_1$

Table 5: Computational overhead for search/decryption between ASE (top)/KP-ABE (bottom) schemes. x_1 denotes the total number of matched keyword/attribute names subset. x_2 denotes the total number of keywords/attributes under every matched names subset.

Schemes	Trapdoor/Secret key		Ciphertext	
	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_1$	$\mathbb{G}_2$
CWDWL16 [21]	1	$6\ell+1$	$5m+1$	-
MZNLHS17 [44]	1	$4\ell^2+2\ell+1$	6	-
FEASE (Fig. 4)	2ℓ	1	m	2
PAEKS (Fig. 5)	2ℓ	1	m	2
FAME [3]	3ℓ	3	$3m$	3
FABEO [50]	ℓ	1	m	1
A-KP-ABE (Fig. 3)	2ℓ	1	m	2

Table 6: Comparison of communication overhead between ASE (top)/KP-ABE (bottom) schemes. m denotes the number of keywords/attributes in the ciphertext, and ℓ is the number of rows of the MSP matrix.

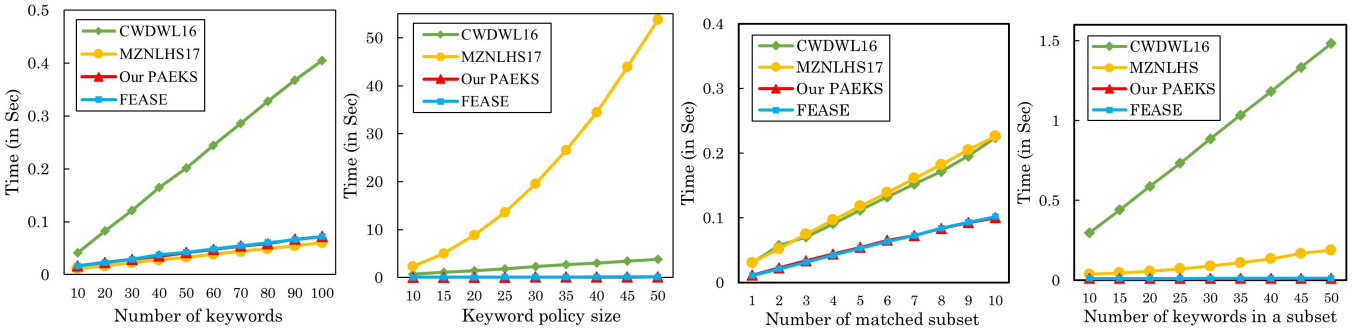
Then we can see the comparison of the communication overhead in Table 6, which shows the number of group elements of trapdoor and ciphertext. Note that in general, elements in $\mathbb{G}_2$ are about 2 to 3 times the size of elements in $\mathbb{G}_1$. For ciphertext, MZNLHS17 has a constant of 6 ciphertext elements in $\mathbb{G}_1$. FEASE/PAEKS has m element in $\mathbb{G}_1$ and 2 elements in $\mathbb{G}_2$, which is less than CWDWL16 that has $5m+1$ elements on $\mathbb{G}_1$. For trapdoor, FEASE/PAEKS has 2ℓ elements in $\mathbb{G}_1$ and 1 element in $\mathbb{G}_2$ while CWDWL16 has $6\ell+1$ elements in $\mathbb{G}_2$ group and 1 element in $\mathbb{G}_1$. The worst one is MZNLHS17 that they have quadratic $4\ell^2+2\ell+1$ trapdoor elements in $\mathbb{G}_2$, which is a trade-off of their constant-size ciphertext.

7.4 KP-ABE schemes

We can see the running times for KP-ABE schemes in Fig. 7. For encryption, our A-KP-ABE scheme runs a very fast 0.07s for the encryption of 100 keywords, with only 0.01s slower than FABEO! FAME needs 0.38s in the same case, which is 5 times slower than our A-KP-ABE. Referring to Table 4, this is because our A-KP-ABE has the same m exponentiations and hashes in $\mathbb{G}_1$ as same as FABEO while FAME has $6m$. The only difference between our A-KP-ABE and FABEO is that we have one more exponentiation in $\mathbb{G}_2$.

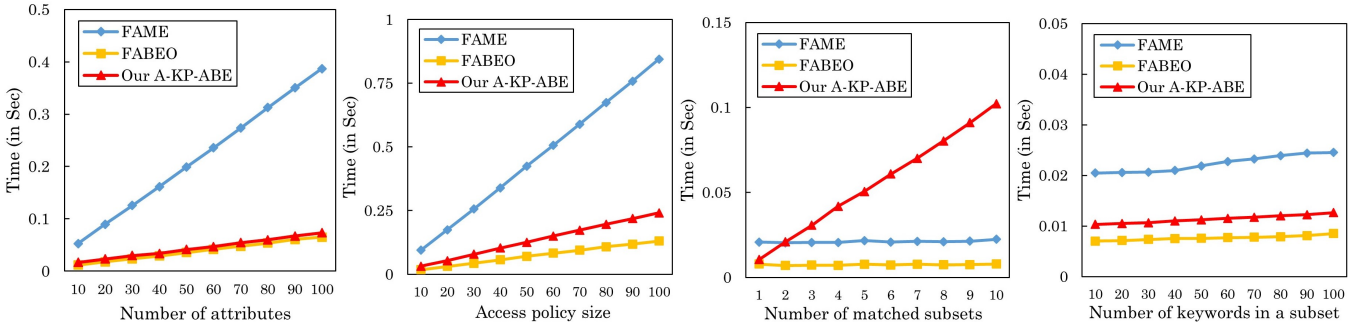
For key generation, FABEO has the fastest 0.13s for a secret key with 100 attributes. Our A-KP-ABE doubles the time with 0.24s but it is 3.5 times faster than FAME. As Table 4 shows, all schemes mainly build elements in $\mathbb{G}_1$. FABEO has ℓ hashes and 2ℓ exponentiations in $\mathbb{G}_1$, our A-KP-ABE has double size calculations for them in order to build the D-LIN type construction. Nevertheless, it is more efficient than FAME since the number of exponentiations, hashes, and multiplications of FAME all depend on ℓ and n.

For decryption, the Fig. 7 (c) shows that FAME and FABEO all have a constant decryption time of 0.02s and 0.007s respectively while our A-KP-ABE has a linear increase with the number of matched attribute names subset x_1. Even in this case, our A-KP-ABE is only 0.08s slower than FAME and 0.093s slower than FABEO when $x_1 = 10$. We can see from Fig. 7 (d) that when $x_1 = 1$, our A-KP-ABE has a constant 0.012s no matter how many attributes are included in the subset, which is very close to FABEO and even two times faster than FAME. In other words, when $x_1 \leq 2$, our A-KP-ABE can decrypt at least as fast as FAME. Table 5 shows the reason that, when $x_1 = 1$, our A-KP-ABE has a constant 3 pairings that lie between FABEO (2 pairings) and FAME (6 pairings). The linear relation to x_1 is the main bottleneck between our A-KP-ABE and non-anonymous FAME and FABEO since the former protects anonymity by using the partially hidden structure while the latter does not. However, it is common sense that stronger security requirement leads to the degradation of efficiency. Compared to former A-KP-ABE schemes, our scheme already achieves the smallest gap between the anonymous ABE and non-anonymous ABE field, in which our A-KP-ABE scheme can even have comparable efficiency to the fastest non-anonymous KP-ABE schemes!



(a) ASE Encryption (b) ASE Trapdoor generation (c) ASE Search regarding matched subset number (assume only one keyword in each subset) (d) ASE Search regarding keyword number in a matched subset (assume there is only one matched subset)

Figure 6: Running times for ASE schemes.



(a) KP-ABE Encryption (b) KP-ABE Key generation (c) KP-ABE Decryption regarding matched subset number (assume only one attribute in each subset) (d) KP-ABE Decryption regarding keyword number in a matched subset (assume there is only one matched subset)

Figure 7: Running times for KP-ABE schemes.

To compare the communication overhead, we can go through Table 6. For ciphertexts, FABEO has only m elements in $\mathbb{G}_1$ and one element in $\mathbb{G}_2$, FAME is three times heavier than FABEO in every parameter. Our A-KP-ABE only adds one more element in $\mathbb{G}_2$ so it has a very similar ciphertext size as FABEO. For the secret key, FABEO has ℓ size elements in $\mathbb{G}_1$ and 1 element in $\mathbb{G}_2$, while FAME has thrice more again. Our A-KP-ABE doubles the size of elements in $\mathbb{G}_1$ of FABEO, so the communication overhead lies between FABEO and FAME.

7.5 Experiments with larger datasets

In the previous sections, we compared our schemes with the state-of-the-art expressive ASE and ABE schemes using 100 keywords. Despite being acknowledged as the fastest in the expressive ASE field, we recognized the importance of testing our schemes on a larger, more practical dataset size. Thus, we expand our dataset to 1000, 5000, and 10000 keywords for both our FEASE and PAEKS schemes. This decision was influenced by the fact that many state-of-the-art SSE schemes evaluate their efficiency on much larger datasets. This marks the first ASE work to explore datasets of such a substantial scale.

Schemes	Dataset	Setup	Trap	Enc	Search
FEASE	1000	0.018	2.17	0.62	0.068
	5000	0.018	14.13	3.28	1.5
	10000	0.018	35.64	6.61	6.76
PAEKS	1000	0.031	2.05	0.64	0.076
	5000	0.031	12.81	3.05	1.75
	10000	0.031	33.4	6.31	6.55

Table 7: The running time (in seconds) for our FEASE and PAEKS in larger datasets with 1000, 5000, and 10000 keywords for both the keyword sets and keyword policies.

Table 7 presents the setup, trapdoor generation, encryption, and search algorithms for both our FEASE and PAEKS schemes. These algorithms were tested with 1000, 5000, and 10000 keywords, considering "AND" gates between all keywords in the policy for simplicity. First, the setup time for both schemes remains constant and unaffected by the number of keywords [13]. In the case of FEASE, with a dataset containing

[13]The same considerations apply to the data sender key and data receiver

1000 keywords, trapdoor generation, and encryption take 2.17s and 0.62s, respectively. The search algorithm operates at 0.068s. As the dataset expands to 5000 keywords, these times increase to 14.13s, 3.28s, and 1.5s, respectively. For a dataset of 10,000 keywords, these times further rise to 35.64s, 6.61s, and 6.76s. Besides, the results from the PAEKS scheme mirror those of FEASE closely, given their similar constructions.

The diagrams in Figure 6 show a strict linear correlation between the running time of encryption and the number of keywords in the dataset. However, there are deviations in the running time of trapdoor generation and the search algorithm, exceeding the linear relation with the keyword count. This discrepancy arises because when the number of keywords surpasses 1000, the built-in recursive functions "evalStack" and "requiredAttributes" utilized in the trapdoor and search algorithm hit the maximum recursion depth of the program. Consequently, the program requires additional memory to execute, leading to increased running times. Addressing this issue and optimizing the program remain areas for our future work.

Based on the results of this experiment, it is evident that while the efficiency of ASE has not yet reached the same level as SSE, our FEASE demonstrates state-of-the-art efficiency in the expressive ASE field. In summary, the asymptotic performance of the expressive ASE field is developing from the fully hidden scheme using IPE [34] to partially hidden schemes in composite order groups [37, 42], and further to partially hidden schemes in prime order group [21, 44]. Furthermore, the foundation KP-ABE scheme is chosen from [38] for [37, 42], to [51] for [21, 44], and finally to the most efficient FABEO [50] for our FEASE and PAEKS. The progress in group settings, partially hidden structure, and KP-ABE schemes together form the efficiency enhancement for expressive ASE schemes. We anticipate that future research efforts will continue to explore novel techniques to further enhance ASE efficiency.

8 Conclusion

In this paper, we have proposed a fast and expressive asymmetric searchable encryption (FEASE) scheme and applied similar techniques to create two other fast and expressive applications: a public key authenticated encryption with keyword search (PAEKS) scheme and an anonymous key-policy attribute-based encryption (A-KP-ABE) scheme. The performance of these three schemes reaches the highest efficiency level in these three primitives, and it is comparable to the state-of-the-art non-anonymous ABE schemes FAME and FABEO. Compared to SSE, the lack of capabilities supporting dynamic updates is still a shortcoming of ASE schemes. In the future, we will carry on our research for the dynamism of the ASE field.

key generation algorithms in the PAEKS scheme. Therefore, we have omitted the running time for these two algorithms in our evaluation.

Acknowledgments

This work is supported by the EU's research and innovation program: 952697 (ASSURED), 101019645 (SECANT), 101069688 (CONNECT), 101070627 (REWIRE), and 101095634 (ENTRUST), which are funded by the UK government Horizon Europe guarantee and administered by UKRI.

References

[1] FEASE: Fast and Expressive Asymmetric Searchable Encryption. https://github.com/Usenix2024/FEASE, 2023.

[2] Michel Abdalla et al. Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions. In CRYPTO, 2005.

[3] Shashank Agrawal and Melissa Chase. Fame: Fast Aattribute-Based Message Encryption. In CCS, 2017.

[4] Joseph A Akinyele et al. Charm: a Framework for Rapidly Prototyping Cryptosystems. JCE, 2013.

[5] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Public Key Encryption with Keyword Search Revisited. In ICCSA, 2008.

[6] Amos Beimel. Secure Schemes for Secret Sharing and Key Distribution. PhD thesis, 1996.

[7] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and Efficiently Searchable Encryption. In CRYPTO, 2007.

[8] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In CCS, 1993.

[9] Mihir Bellare and Phillip Rogaway. Optimal Asymmetric Encryption. In EUROCRYPT, 1995.

[10] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-Policy Attribute-Based Encryption. In IEEE S&P, pages 321–334, 2007.

[11] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical Identity Based Encryption with Constant Size Ciphertext. In EUROCRYPT, 2005.

[12] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short Group Signatures. In CRYPTO, 2004.

[13] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public Key Encryption with Keyword Search. In EUROCRYPT, 2004.

[14] Dan Boneh and Brent Waters. Conjunctive, Subset, and Range Queries on Encrypted Data. In TCC 2007, 2007.

[15] Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter. A survey of provably secure searchable encryption. ACM Computing Surveys, 2014.

[16] Elie Bouscatié, Guilhem Castagnos, and Olivier Sanders. Public Key Encryption with Flexible Pattern Matching. In ASIACRYPT, 2021.

[17] Élie Bouscatié, Guilhem Castagnos, and Olivier Sanders. Pattern Matching in Encrypted Stream from Inner Product Encryption. In PKC, 2023.

[18] Jin Wook Byun et al. Off-Line Keyword Guessing Attacks on Recent Keyword Search Schemes over Encrypted Data. In SDM, 2006.

[19] Rongmao Chen et al. Dual-Server Public-Key Encryption with Keyword Search for Secure Cloud Storage. IEEE TIFS, 2015.

[20] Leixiao Cheng and Fei Meng. Public Key Authenticated Encryption with Keyword Search from LWE. In ESORIC, 2022.

[21] Hui Cui, Zhiguo Wan, Robert H Deng, Guilin Wang, and Yingjiu Li. Efficient and Expressive Keyword Search over Encrypted Data in Cloud. IEEE TDSC, 2016.

[22] Nicolas Desmoulins, Pierre-Alain Fouque, Cristina Onete, and Olivier Sanders. Pattern Matching on Encrypted Streams. In ASIACRYPT, 2018.

[23] Giovanni Di Crescenzo and Vishal Saraswat. Public Key Encryption with Searchable Keywords Based on Jacobi Symbols. In INDOCRYPT, 2007.

[24] Keita Emura. Generic Construction of Public-key Authenticated Encryption with Keyword Search Revisited: Stronger Security and Efficient Construction. In APKC, 2022.

[25] David Mandell Freeman. Converting Pairing-Based Cryptosystems from Composite-Order Groups to Prime-Order Groups. In EUROCRYPT, 2010.

[26] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, 2003.

[27] Philippe Golle, Jessica Staddon, and Brent Waters. Secure Conjunctive Keyword Search over Encrypted Data. In ACNS, 2004.

[28] Matthew Green et al. Outsourcing the Decryption of ABE Ciphertexts. In USENIX, 2011.

[29] Aurore Guillevic. Comparing the Pairing Efficiency over Composite-Order and Prime-Order Elliptic Curves. In ACNS, 2013.

[30] Susan Hohenberger and Brent Waters. Attribute-based encryption with fast decryption. In PKC, 2013.

[31] Qiong Huang et al. An Efficient Public-Key Searchable Encryption Scheme Secure against Inside Keyword Guessing Attacks. Information Sciences, 2017.

[32] Yong Ho Hwang and Pil Joong Lee. Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-User System. In Pairing, 2007.

[33] Seny Kamara, Charalampos P., and Tom R. Dynamic Searchable Symmetric Encryption. In CCS, 2012.

[34] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. In EUROCRYPT, 2008.

[35] Dalia Khader. Public Key Encryption with Keyword Ssearch Based on K-Resilient IBE. In ICCSA, 2006.

[36] Junzuo Lai, Robert H Deng, and Yingjiu Li. Expressive CP-ABE with Partially Hidden Access Structures. In AsiaCCS, 2012.

[37] Junzuo Lai, Xuhua Zhou, Robert Huijie Deng, Yingjiu Li, and Kefei Chen. Expressive Search on Encrypted Data. In AsiaCCS, 2013.

[38] Allison Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki T., and Brent W. Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption. In EUROCRYPT, 2010.

[39] Allison Lewko and Brent Waters. Unbounded HIBE and Attribute-Based Encryption. In EUROCRYPT, 2011.

[40] Hongbo Li et al. Public-key Authenticated Encryption With Keyword Search Supporting Constant Trapdoor Generation and Fast Search. IEEE TIFS, 2022.

[41] Zi-Yuan Liu et al. Public-Key Authenticated Encryption with Keyword Search: Cryptanalysis, Enhanced Security, and Quantum-Resistant Instantiation. In AsiaCCS, 2022.

[42] Zhiquan Lv, Cheng Hong, Min Zhang, and Dengguo Feng. Expressive and Secure Searchable Encryption in the Public Key Setting. In ISC, 2014.

[43] Long Meng, Liqun Chen, Yangguang Tian, Mark Manulis, and Suhui Liu. FEASE: Fast and Expressive Asymmetric Searchable Encryption. Accepted in USENIX Security Symposium, 2024. https://eprint.iacr.org/2024/054.

[44] Ru Meng et al. An Efficient Key-Policy Attribute-Based Searchable Encryption in Prime-Order Groups. In ProvSec, 2017.

[45] Mahnaz Noroozi and Ziba Eslami. Public Key Authenticated Encryption with Keyword Search: Revisited. IET Information Security, 2019.

[46] Dong Jin Park, Kihyun Kim, and Pil Joong Lee. Public Key Encryption with Conjunctive Field Keyword Search. In ISA, 2005.

[47] Baodong Qin et al. Public-key Authenticated Encryption with Keyword Search Revisited: Security Model and Constructions. Information Sciences, 2020.

[48] Baodong Qin et al. Improved Security Model for Public-Key Authenticated Encryption with Keyword Search. In ProvSec, 2021.

[49] Hyun Sook Rhee, Jong Hwan Park, Willy Susilo, and Dong Hoon Lee. Improved Searchable Public Key Kncryption with Designated Tester. In AsiaCCS, 2009.

[50] Doreen Riepel and Hoeteck Wee. Fabeo: Fast Attribute-Based Encryption with Optimal Security. In CCS, 2022.

[51] Yannis Rouselakis and Brent Waters. Practical Constructions and New Proof Methods for Large Universe Attribute-Based Encryption. In CCS, 2013.

[52] Dhruti Sharma. Searchable encryption: A survey. Information Security Journal, 2023.

[53] Chen Shen et al. Expressive Public-Key Encryption with Keyword Search: Generic Construction from KP-ABE and an Efficient Scheme over Prime-Order Groups. IEEE Access, 2019.

[54] Victor Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In EUROCRYPT, 1997.

[55] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical Techniques for Searches on Encrypted Data. In IEEE S&P, 2000.

[56] Qiang Tang and Liqun Chen. Public-Key Encryption with Registered Keyword Search. In EuroPKI, 2010.

[57] Yi-Fan Tseng, Chun-I Fan, and Zi-Cheng Liu. Fast Keyword Search over Encrypted Data with Short Ciphertext in Clouds. JISA, 2022.

[58] Peng Xu et al. Public-key Encryption with Fuzzy Keyword Search: A Provably Secure Scheme Under Keyword Guessing Attack. IEEE ToC, 2012.

[59] Bo Zhang and Fangguo Zhang. An Efficient Public Key Encryption with Conjunctive - Subset Keywords Search. JNCA, 2011.

[60] Dong Zhang, Qing F., Hongyi Q., and Min L. A Public-Key Encryption with Multi-keyword Search Scheme for Cloud-based Smart Grids. In IEEE DSC, 2021.

[61] Qingji Zheng, Shouhuai Xu, and Giuseppe Ateniese. Vabks: Verifiable attribute-based keyword search over outsourced encrypted data. In IEEE INFOCOM, 2014.

A CWDWL16 scheme in the Type-III setting

Fig. 8 shows [21] that we transformed into the Type-III setting.

$(\mathsf{pp}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. Run $\mathsf{GroupGen}(1^\lambda)$ to obtain $\mathsf{par} := (p, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2)$. Pick $u_1, h_1, \delta_1 \xleftarrow{\$} \mathbb{G}_1$, $u_2, h_2, \delta_2 \xleftarrow{\$} \mathbb{G}_2$, $\alpha, d_1, d_2, d_3, d_4 \xleftarrow{\$} \mathbb{Z}_p$, and a hash function $\mathsf{H}: \mathbb{G}_T \to \mathbb{G}_2$. Compute the public key and secret key as

$$\mathsf{pp} = (\mathsf{par}, \mathsf{H}, g_1, u_1, h_1, \delta_1, u_2, g_1^{d_1}, g_1^{d_2}, g_1^{d_3}, g_1^{d_4}, e(g_1, g_2)^\alpha)$$

$$\mathsf{sk} = (\alpha, g_2, h_2, \delta_2, d_1, d_2, d_3, d_4)$$

$(\mathsf{pk}_c, \mathsf{sk}_c) \leftarrow \mathsf{KeyGen}_c(\mathsf{pp})$. Pick $\beta \xleftarrow{\$} \mathbb{Z}_p$ and compute

$$\mathsf{pk}_c = g_1^\beta, \mathsf{sk}_c = \beta$$

$\mathsf{td} \leftarrow \mathsf{Trap}(\mathsf{pp}, \mathsf{pk}_c, \mathsf{sk}, \mathbb{P} = (\mathrm{M}, \pi, \mathbb{P} = (\mathrm{M}, \pi, \{\pi(i)\}_{i\in[\ell]}))$.

Pick $r, r', t_{1,1}, t_{1,2}, ..., t_{\ell,1}, t_{\ell,2} \xleftarrow{\$} \mathbb{Z}_p$, $\mathbf{v} \xleftarrow{\$} \mathbb{Z}_p^{n-1}$. Compute

$$\begin{aligned}
&\mathsf{td}_{1,i} = g_2^{\mathrm{M}_i(\alpha\|\mathbf{v})^\top} \cdot \delta_2^{d_1 d_2 t_{i,1} + d_3 d_4 t_{i,2}},\\
&\mathsf{td}_{2,i} = \mathsf{H}(e(\mathsf{pk}_c, \mathsf{td}_8)^r) \cdot g_2^{d_1 d_2 t_{i,1} + d_3 d_4 t_{i,2}},\\
&\mathsf{td}_{3,i} = (u_2^{\pi(i)} h_2)^{-d_2 t_{i,1}}, \mathsf{td}_{4,i} = (u_2^{\pi(i)} h_2)^{-d_1 t_{i,1}},\\
&\mathsf{td}_{5,i} = (u_2^{\pi(i)} h_2)^{-d_4 t_{i,2}}, \mathsf{td}_{6,i} = (u_2^{\pi(i)} h_2)^{-d_3 t_{i,2}},\\
&\mathsf{td}_7 = g_1^r, \mathsf{td}_8 = g_2^{r'}.
\end{aligned}$$

Output $\mathsf{td} = ((\mathrm{M}, \pi, \{n_{\pi(i)}\}_{i\in[\ell]}), \{\mathsf{td}_{1,i}, \mathsf{td}_{2,i}, \mathsf{td}_{3,i}, \mathsf{td}_{4,i}, \mathsf{td}_{5,i}, \mathsf{td}_{6,i}\}_{i\in[\ell]}, \mathsf{td}_7, \mathsf{td}_8)$.

$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathbb{W} = \{w_i\}_{i\in[m]} = \{n_i, v_i\}_{i\in[m]})$.

Pick $\mu, s_{1,1}, s_{1,2}, ..., s_{m,1}, s_{m,2}, z_1, ..., z_m \xleftarrow{\$} \mathbb{Z}_p$ and compute

$$\begin{aligned}
&\mathsf{ct}_1 = g_1^\mu, \mathsf{ct}_{2,i} = \delta_1^{-\mu}(u_1^{w_i} h_1)^{z_i}, \mathsf{ct}_{3,i} = g_1^{d_1(z_i - s_{i,1})}, \mathsf{ct}_{4,i} = g_1^{d_2 s_{i,1}},\\
&\mathsf{ct}_{5,i} = g_1^{d_3(z_i - s_{i,2})}, \mathsf{ct}_{6,i} = g_1^{d_4 s_{i,2}}, \mathsf{ct}_7 = e(g_1, g_2)^{\alpha\mu}
\end{aligned}$$

Output $\mathsf{ct} = (\{n_i\}_{i\in[m]}, \mathsf{ct}_1, \{\mathsf{ct}_{2,i}, \mathsf{ct}_{3,i}, \mathsf{ct}_{4,i}, \mathsf{ct}_{5,i}, \mathsf{ct}_{6,i}\}_{i\in[m]}, \mathsf{ct}_7)$.

$1/0 \leftarrow \mathsf{Search}(\mathsf{pp}, \mathsf{sk}_c, \mathsf{ct}, \mathsf{td})$. Tests if there is any subset I that matches the keyword names $\{n_i\}_{i\in[m]}$ in ct with $(\mathrm{M}, \pi, \{n_{\pi(i)}\}_{i\in[\ell]})$ in td. If not, return 0. Otherwise, it finds constants $\{\gamma_i\}_{i\in I}$ s.t. $\sum_{i\in I}\gamma_i \mathrm{M}_i = (1, 0, ..., 0)$ and computes:

$$\mathsf{ct}_7 = \prod_{i\in I}(e(\mathsf{ct}_1, \mathsf{td}_{1,i}) e(\mathsf{ct}_{2,i}, \frac{\mathsf{td}_{2,i}}{\mathsf{H}(e(\mathsf{td}_7, \mathsf{td}_8)^\beta)}) e(\mathsf{ct}_{3,i}, \mathsf{td}_{3,i})$$
$$e(\mathsf{ct}_{4,i}, \mathsf{td}_{4,i}) e(\mathsf{ct}_{5,i}, \mathsf{td}_{5,i}) e(\mathsf{ct}_{6,i}, \mathsf{td}_{6,i}))^{\gamma_i},$$

If the equation holds, return 1. Otherwise, the cloud server continues to find another subset of I and repeats the checking. If the above equation does not hold for all subsets, return 0.

Figure 8: The construction of Cui et al. [21] ASE scheme.