Position Papers of the 19th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 17–24
DOI: 10.15439/2024F3018
ISSN 2300-5963 ACSIS, Vol. 40

# Transforming Attribute-Based Encryption schemes into Asymmetric Searchable Encryption schemes

Elisa Giurgea
0009-0004-3610-3666
Alexandru Ioan Cuza University,
Department of Computer Science,
Iasi, Romania
Email: elisa.giurgea@gmail.com

***Abstract*—Attribute-Based Encryption (ABE) and Asymmetric Searchable Encryption (ASE) are two highly useful Public-Key Encryption (PKE) technologies in today's cloud computing landscape.**

**By leveraging the idea that the attributes from ABE can serve as keywords for ASE, we propose an efficient technique to translate any ABE schemes into ASE schemes. We address both the case of Ciphertext-Policy Attribute-Based Searchable Encryption (CP-ABSE) and Key-Policy Attribute-Based Searchable Encryption (KP-ABSE) schemes.**

**Our main goal with these schemes is to maintain the security properties of ABE while introducing efficient search capabilities, thereby facilitating further advancements in ASE development. To validate our theoretical proposals, we have analyzed their practical applicability using existing ABE implementations.**

## I. Introduction

SEARCHABLE Encryption (SE) is a cryptographic technique that allows users to search over encrypted data without decrypting it, preserving data confidentiality while enabling search functionality. SE can be broadly categorized into two types: Symmetric Searchable Encryption (SSE) [1] and Asymmetric Searchable Encryption (ASE) [2].

Symmetric Searchable Encryption schemes utilize a single secret key that is shared between the data owner and the authorized users. This approach offers several advantages: efficiency in terms of computational overhead and search speed, making them suitable for scenarios where performance is critical; and simplicity, as the key management in SSE is straightforward since it involves only one key for both encryption and decryption. However, SSE also has notable drawbacks, such as limited access control with all the users with the secret key being able to access all the data, which may not be desirable in many applications; and scalability issues, as securely distributing and managing the single secret key becomes challenging with the number of system users increasing.

In contrast, Asymmetric Searchable Encryption employs a pair of cryptographic keys: a public key for encryption and a private key for decryption. This approach addresses some of the limitations of SSE by enabling more granular access control, allowing the data owner to specify which users can access which data. This is particularly useful in multi-user environments with varying access privileges. Moreover, ASE offers better scalability, each user having their own key pair, simplifying key distribution and management as the system scales up. Despite these advantages, ASE comes with its own set of challenges, as the use of public and private keys introduces additional complexity in terms of key management and the overall cryptographic operations. ASE schemes often incur higher computational costs and longer search times compared to SSE, which can be a major drawback in resource-constrained environments.

Given the trade-offs between SSE and ASE, there is a significant interest in developing more efficient ASE schemes that can leverage the strengths of existing cryptographic methods, as seen with the recently proposed [7], [3], [4], [5] and [6]. One promising approach is to derive ASE schemes from Attribute-Based Encryption (ABE). By treating attributes in ABE as keywords in ASE, it is possible to create systems that offer both efficient search capabilities and robust access control. Recently, the idea of treating attributes as keywords was also used by Long Meng, Liqun Chen and Yangguang Tian as a trivial assumption behind a new proposal of an efficient ASE scheme extended from an A-KP-ABE scheme [7].

This paper aims to propose a new class of efficient ASE schemes derived from ABE. Specifically, we present theoretical formalizations for Ciphertext-Policy Attribute-Based Searchable Encryption (CP-ABSE) and Key-Policy Attribute-Based Searchable Encryption (KP-ABSE), showing that any ABE scheme can become an ASE scheme. Our objective is to utilize the inherent advantages of ABE, such as fine-grained access control and scalability, to develop ASE schemes that offer efficient, secure, and flexible search functionality.

## II. General transformation of ABE in ASE

In ABE, attributes can be viewed as keywords. By treating these attributes as keywords in ASE, we can introduce search functionalities without compromising the existing access control mechanisms. This allows for the creation of searchable indexes based on attributes or access policies, enabling efficient retrieval of encrypted data based on specified search criteria.

Building on this concept of treating ABE attributes as keywords for ASE, having as basis the general schemes from [8], we will be further presenting the formalization for the



**Topical area:** Computer Science & Systems

extension of both ABE forms to ASE. The notations of general functions needed in the formalization are subsequently explained in "Table 1".

### A. CP-ABSE formalization

**Algorithm 1** CP-ABSE GlobalInit

$index \leftarrow [\,][\,]$; $H \leftarrow HashFunction()$;

**Algorithm 2** CP-ABSE Setup

**Require:** Security parameter $1^\lambda$;
**Ensure:** Public parameters $PK$, master key $msk$;
$(PK, msk) \leftarrow Setup(1^\lambda)$;
**return** $(PK, msk)$;

For Ciphertext Attribute-Based Encryption schemes, the setup algorithm of the system, $Setup(1^\lambda)$, takes as input a security parameter, $1^\lambda$ and outputs the public parameters, $PK$, and a master key, $msk$, which is known only to the private key generator $(PKG)$. Aside from the CP-ABE setup related initialisation, we are also adding to the system 2 new global parameters: $index$, which will store the encrypted documents/messages at a given index, and $H$, a hash function which will be further used in the index creation and query generation steps.

**Algorithm 3** CP-ABSE KeyGen

**Require:** Public parameters $PK$, master key $msk$, set of attributes $\gamma = \{a_1, a_2, \ldots, a_n\}$;
**Ensure:** Private key $D_\gamma$;
$D_\gamma \leftarrow KeyGen(PK, msk, \gamma)$;
**return** $D_\gamma$;

Next, onto the key generation algorithm, there are no changes from the regular behaviour expected of a CP-ABE scheme: $KeyGen(PK, msk, \gamma)$ algorithm takes as input the public parameters, $PK$, the master key, $msk$, both generated during the setup phase, and a set of attributes, $\gamma = \{a_1, a_2, \ldots, a_n\}$, for which the secret key is to be generated. It outputs the private key $D_\gamma$ which encapsulates $\gamma$.

**Algorithm 4** CP-ABSE Encrypt

**Require:** Message $m$, public parameters $PK$, access policy $A = BooleanExpr(\{a_1, a_2, \ldots, a_n\})$;
**Ensure:** Ciphertext $ct$;
$ct \leftarrow Encrypt(m, PK, A)$;
**return** $ct$;

The encryption algorithm also remains the same as with regular CP-ABE: $Enc(m, PK, A)$ takes as input parameters a message, $m$, an access policy, $A$ (which is or can be trivially transformed into a logical boolean expression $A = BooleanExpr(\{a_1, a_2, \ldots, a_n\})$), and the public parameters, $PK$. It outputs the ciphertext, $ct$.

**Algorithm 5** CP-ABSE CreateIndex

**Require:** Ciphertext $ct$, same access policy $A = BooleanExpr(\{a_1, a_2, \ldots, a_n\})$;
$A' \leftarrow A$;
$\gamma' \leftarrow ParsePolicy(A)$;
**for** $a'_i$ in $\gamma'$ **do**
  $A'.replace(a'_i, H(a'_i))$;
**end for**
$id \leftarrow ct$;
**if** $\exists index[A']$ in $index$ **then**
  $index[A'].append(id)$;
**else**
  $index[A'] \leftarrow [id]$;
**end if**

To ensure that the documents encrypted by the system are stored and searchable, we are executing $CreateIndex(ct, A)$ right after a document/message is encrypted based on a given policy. The algorithm is simply storing the ciphertexts at an index obtained from the same policy $A$ based on which the message was encrypted in the previous step.

To further hide the index values, the attributes from the given access policy $A$ (obtained via parsing the policy) will be further hashed with the system defined hash function $H$, and their clear-text values will be subsequently replaced in the policy with the hashed ones. The initial logical relations between the attributes in the policy will be kept for the newly obtained hashed policy, $A' = BooleanExpr(\{H(a_1), H(a_2), ..., H(a_n)\})$.

After the execution of the aforementioned operations, we will now append the document id for the ciphertext (in this demonstration case, the document id is the ciphertext itself) to the index of the hashed policy to store it.

These steps are to be repeated as needed for the multiple users which are to use the system and the multiple protected-access documents which need to be stored.

Now, how can a system user with a secret key, $D_\gamma$, get their list of available documents given the user's attached attributes?

**Algorithm 6** CP-ABSE GenerateQuery

**Require:** Private key $D_\gamma$;
**Ensure:** Query $Q = \{H(a_1), H(a_2), ..., H(a_n)\}$;
$\gamma \leftarrow ExtractAttributes(D_\gamma)$;
$Q \leftarrow \{\}$;
**for** $a_i$ in $\gamma$ **do**
  $Q.append(H(a_i))$;
**end for**
**return** $Q$;

First, the $GenerateQuery(D_\gamma)$ algorithm will be executed with the user's secret key as an input parameter. The generation of the query itself is based on the set of attributes, $\gamma$, encapsulated in the secret key $D_\gamma$.

The set of attributes, $\gamma$, is extracted from $D_\gamma$ (the extraction method itself is specific to the proposed CP-ABSE scheme)

TABLE I
FUNCTIONS USED IN THE FORMALIZATIONS AND THEIR MEANING

| Notation | First operation |
|---|---|
| $HashFunction()$ | Any trapdoor or one-way hash function desired to hide the values |
| $BooleanExpr(set\ of\ attributes)$ | An access control policy obtained as a boolean expression over a given set of attributes |
| $EvaluatePolicy(access\ policy,\ set\ of\ attributes)$ | Function to evaluate whether a given set of attributes satisfies a given access policy |
| $ParsePolicy(access\ policy)$ | Function to extract the attributes used in a private policy in a set of attributes |
| $ExtractAttributes(CP-ABE\ private\ key)$ | Function to extract the attributes encapsulated in a given CP-ABE specific private key |
| $ExtractPolicy(KP-ABE\ private\ key)$ | Function to extract the access policy encapsulated in a given KP-ABE specific private key |

and each of them are hashed with the same one-way hash function $H$ used for the attributes in the access policy during the index creation step to ensure unitary information. The hashed extracted attributes are next compacted into a set which will be returned as the search query, $Q = \{H(a_1), H(a_2), ..., H(a_n)\}$.

**Algorithm 7** CP-ABSE Search

**Require:** Query $Q$;
**Ensure:** List of document ids $results = \{id_1, id_2, ..., id_n\}$;
$results \leftarrow \{\}$
**for** $(A', ids)$ in $index$ **do**
  **if** $EvaluatePolicy(A', Q) = TRUE$ **then**
    $results.extend(ids)$;
  **end if**
**end for**
**return** $results$;

After successfully generating a query, $Q$, we can now search for the encrypted documents which are satisfying $Q$ by executing the $Search(Q)$ algorithm, which is to return the document ids (in this demonstration, the ciphertexts) from the $index$. The search itself is based on evaluating the hashed policies indices against query $Q$, which contains a list with the user's hashed attributes.

**How would this evaluation work?**

The evaluation step is quite trivial, as, having the policy, $A' = BooleanExpr(\{H(a_1), H(a_2), ..., H(a_n)\})$, in order for it to be satisfied the logical expression needs to be $TRUE$. And how do we check that having the set of hashed attributes, $Q = \{H(a_1), H(a_2), ..., H(a_n)\}$? For each hashed attribute in the set, we are replacing its appearances in the hashed policy with the $TRUE$ value. For the rest of the hashed attributes in the hashed policy which were not found in the set of hashed attributes given as an input parameter, we are replacing their appearances with the $FALSE$ value, as them not being in the set implies that the user does not have them attached to its secret key, $D_\gamma$. After obtaining the policy to be evaluated under a format such as $(TRUE\ and\ FALSE)\ or\ TRUE$, we are evaluating it using a built-in default boolean $eval$ function.

In case of the evaluation step for a certain indexed hashed policy returning $TRUE$, the document ids indexed under the aforementioned policy will be collected in a set of $results$. We will be repeating the step for all the indexes to retrieve all the document ids which would be viable results for the search query $Q$.

**Algorithm 8** CP-ABSE Decrypt

**Require:** Ciphertext $ct$, public parameters $PK$, private key $D_\gamma$;
**Ensure:** Message $m'$;
$m' \leftarrow Decrypt(ct, PK, D_\gamma)$;
**return** $m'$;

Now, after obtaining the set of $results$, for each $result$ (the document id being the actual ciphertext in this case) and we will be proceeding with the CP-ABE specific decryption algorithm, $Decrypt(result, PK, D_\gamma)$. The decryption algorithm takes as input the ciphertext, $result$, which was encrypted with an access policy, $A$, the public parameters, $PK$, and the private key of the user, $D_\gamma$. It outputs the initially encrypted document/message, if the set of attributes, $\gamma$, encapsulated in the private key, $D_\gamma$, satisfies the access policy $A$.

**Algorithm 9** CP-ABSE Usage Example

$GlobalInit()$;
$(PK, msk) \leftarrow Setup()$ {Global system setup}
$\gamma \leftarrow \{a_1, a_2, ..., a_n\}$;
... {Attribute initialization for system users}
$D_\gamma \leftarrow KeyGen(PK, msk, \gamma)$;
... {System registration / private key generation for system users}
Get messages / documents to be encrypted and stored;
Define necessary access policies;
$ct \leftarrow Encrypt(m, PK, A)$; $CreateIndex(ct, A)$;
$Q \leftarrow GenerateQuery(D_\gamma)$;
$results \leftarrow Search(Q)$;
**for** $result$ in $results$ **do**
  $m' \leftarrow Decrypt(result, PK, D_\gamma)$;
**end for**

### *B. KP-ABSE formalization*

**Algorithm 10** KP-ABSE GlobalInit

$index \leftarrow [\ ][\ ]$; $H \leftarrow HashFunction()$;

**Algorithm 11** KP-ABSE Setup

**Require:** Security parameter $1^\lambda$;
**Ensure:** Public parameters $PK$, master key $msk$;
$(PK, msk) \leftarrow Setup(1^\lambda)$;
**return** $(PK, msk)$;

For Key Policy Attribute-Based Encryption schemes, the setup algorithm of the system, $Setup(1^\lambda)$, is formalized similarly to the CP-ABE one, with it taking a security parameter, $1^\lambda$ and outputing the public parameters, $PK$, and a master key, $msk$, known only to the $PKG$. Likewise to previously formalized CP-ABSE scheme, we are adding the 2 global parameters: $index$ and $H$.

**Algorithm 12** KP-ABSE KeyGen

**Require:** Public parameters $PK$, master key $msk$, access policy $A = BooleanExpr(\{a_1, a_2, \ldots, a_n\})$;
**Ensure:** Private key $D_A$;
$D_A \leftarrow KeyGen(PK, msk, A)$;
**return** $D_A$;

Now, for the key generation algorithm KeyGen (PK, msk, A) we are keeping the same on as it was for KP-ABE schemes in general: by taking as input the previously generated $PK$ and $msk$ and an access policy, $A = BooleanExpr(\{a_1, a_2, \ldots, a_n\})$, the algorithm outputs the private key, $D_A$, which encapsulates the access policy based on which it was generated.

**Algorithm 13** KP-ABSE Encrypt

**Require:** Message $m$, public parameters $PK$, set of attributes $\gamma = \{a_1, a_2, \ldots, a_n\}$;
**Ensure:** Ciphertext $ct$;
$ct \leftarrow Encrypt(m, PK, \gamma)$;
**return** $ct$;

For KP-ABE based KP-ABSE, the encryption step also remains the same to the original, with $Enc(m, PK, \gamma)$ getting as input a document/message, $m$, a set of attributes, $\gamma = \{a_1, a_2, \ldots, a_n\}$, and the public parameters, $PK$ and returning the ciphertext, $ct$, generated based on the attributes given.

**Algorithm 14** KP-ABSE CreateIndex

**Require:** Ciphertext $ct$, same set of attributes $\gamma = \{a_1, a_2, \ldots, a_n\}$;
$\gamma' \leftarrow \{\}$;
**for** $a'_i$ in $\gamma'$ **do**
$\gamma'.add(H(a'_i))$;
**end for**
$id \leftarrow ct$;
**if** $\exists index[\gamma']$ in $index$ **then**
$index[\gamma'].append(id)$;
**else**
$index[\gamma'] \leftarrow [id]$;
**end if**

After the encryption step, we are indexing the ciphertext in the system by executing $CreateIndex(c, \gamma)$ algorithm, which is storing the ciphertext at an index generated based on the same set of attributes, $\gamma$, on which the ciphertext was encrypted. To hide the index values, the attributes are be hashed with the system defined hash function $H$ and compacted in a set. Next, we will append the document id for the ciphertext (again, the document id is the ciphertext itself) to the index of the hashed attributes set object.

The key generation is to be done for the needed number of users, and, for the documents/messages to be stored, the encryption and index creation algorithms are to be executed for whichever number of documents necessary.

When a search is wanted to be performed over the stored encrypted documents, the following algorithms will be executed:

**Algorithm 15** KP-ABSE GenerateQuery

**Require:** Private key $D_A$;
**Ensure:** Query $Q = BooleanExpr(\{H(a_1), H(a_2), ..., H(a_n)\})$, the hashed policy;
$A' \leftarrow ExtractPolicy(D_A)$;
$\gamma' \leftarrow ParsePolicy(A')$;
$Q \leftarrow A'$;
**for** $a'_i$ in $\gamma'$ **do**
$Q.replace(a'_i, H(a'_i))$;
**end for**
**return** $Q$;

First, we need to generate the search query based on the user's secret key, $GenerateQuery(D_A)$. The KP-ABE secret key encapsulates the access policy attributed to the user and, in order to generate the query, we will first be extracting the policy from the key. From the obtained policy, we will now be extracting the attributes. For each attribute's occurrences in the policy, we will be replacing its appearance with its hashed value, $H(a_i)$. Thus, we will be obtaining a hashed policy, which will be acting as our search query, $Q = BooleanExpr(\{H(a_1), H(a_2), ..., H(a_n)\})$.

**Algorithm 16** KP-ABSE Search

**Require:** Query $Q$;
**Ensure:** List of document ids $results = \{id_1, id_2, ..., id_n\}$;
$results \leftarrow \{\}$
**for** $(\gamma', ids)$ in $index$ **do**
  **if** $EvaluatePolicy(Q, \gamma') = TRUE$ **then**
    $results.extend(ids)$;
  **end if**
**end for**
**return** $results$;

Having the query, $Q$, the user can now perform a search for the secret documents which are satisfying $Q$ with the $Search(Q)$ algorithm, which will be returning a list with the accessible document ids/ciphertexts. The search is based on evaluating the hashed policy from the query against all the indices, each of them being a compacted set of hashed attributes.

When the evaluation for a certain indexed attribute set returns $TRUE$, the document ids indexed under it will be collected in a set of $results$.

**Algorithm 17** KP-ABSE Decrypt

**Require:** Ciphertext $ct$, public parameters $PK$, private key $D_\gamma$;
**Ensure:** Message $m'$;
$m' \leftarrow Decrypt(ct, PK, D_\gamma)$;
**return** $m'$;

For each $result$ (with the document id as the ciphertext) obtained in the $results$ set, the KP-ABE specific decryption algorithm, $Decrypt(result, PK, D_A)$ will be executed. Having as input parameters $result$, which was encrypted with a set of attributes, $\gamma$, the public parameters, $PK$, and the secret key of the user, $D_A$, the decryption algorithm outputs the initially encrypted document/message, if the set of attributes, $\gamma$, satisfies the access policy $A$, encapsulated in the private key, $D_A$.

**Algorithm 18** KP-ABSE Usage Example

$GlobalInit()$;
$(PK, msk) \leftarrow Setup()$ {Global system setup}
$A \leftarrow BooleanExpr(\{a_1, a_2, ..., a_n\})$;
... {Access policy initialization for system users}
$D_A \leftarrow KeyGen(PK, msk, A)$;
... {System registration / private key generation for system users}
Get messages / documents to be encrypted and stored;
Define attribute sets;
$ct \leftarrow Encrypt(m, PK, \gamma)$; $CreateIndex(ct, \gamma)$;
$Q \leftarrow GenerateQuery(D_A)$;
$results \leftarrow Search(Q)$;
**for** $result$ in $results$ **do**
  $m' \leftarrow Decrypt(result, PK, D_A)$;
**end for**

## III. Security

### A. CP-ABSE

The initial assumption is that the security of the CP-ABSE scheme is similar to the CP-ABE scheme on which it is based. We will further demonstrate that the additional operations introduced for searchability do not compromise the security guarantees provided by base CP-ABE. Specifically, we will sketch the arguments as to why the confidentiality of the encrypted documents and the privacy of the attributes from the access policies are preserved.

*1) CP-ABE Security Model:* The $Setup$ algorithm security for CP-ABE implies that the algorithm generates the public parameters, $PK$, and a master secret key, $msk$. The security requirement is that without $msk$, it is not feasible for adversary to generate valid private keys for any given attribute set.

The $KeyGen$ algorithm generates a private key $D_\gamma$ for a set of attributes $\gamma$. The security requirement is that an adversary with some private keys $D_{\gamma 1}$, $D_{\gamma 2}$, $D_{\gamma 3}$..., $D_{\gamma k}$ cannot generate a valid private key for a new attribute set $\gamma'$, unless $\gamma'$ is a subset of one of the sets $\gamma i$ where $1 \leq i \leq k$.

Encryption and decryption security wise, the $Encrypt$ algorithm ensures that a ciphertext $ct$ encrypted under an access policy $A$ can only be decrypted by a private key $D_\gamma$ if $\gamma$ satisfies $A$. The security requirement is that an adversary should not be able to decrypt $ct$ without possessing such a $D_\gamma$.

*2) CP-ABSE General Scheme Overview:* In addition to the CP-ABE setup, CP-ABSE introduces an $index$ structure and a hash function $H$ as system global parameters. Algorithms wise, it introduces $CreateIndex$, which is executed after encryption and it indexes the ciphertext using a hashed version of the access policy; $GenerateQuery$, which generates queries by hashing the attributes in the user's private key; and $Search$, which evaluates the hashed query against the hashed policies in the index and returns available results.

Having these added algorithms, we need to take the following security proof components into account:

- The confidentiality of the encrypted documents: Since the encryption and decryption processes in CP-ABSE are identical to CP-ABE, the confidentiality of the encrypted documents relies on the security of the underlying CP-ABE scheme. Any attack on the confidentiality of CP-ABSE ciphertexts can be reduced to an attack on CP-ABE ciphertexts; the $index$ stores only hashed versions of the access policies and does not reveal any additional information about the plaintext or the original attributes. The hash function $H$ should be an irreversible collision-free hash function, ensuring that the hashed values do not leak information about the original attributes.
- The privacy of the attributes in the access policies: The security of the hashed attributes in the index access policies and queries depends on the hash function $H$, which should be chosen to ensure that it is computationally infeasible to reverse-engineer the original attributes from their hashed values. The $GenerateQuery$ process uses

the same hash function $H$, ensuring that the attributes in the user's private key are protected in the same way as the attributes in the access policies.
- The search operation security: The search operation involves evaluating whether the hashed query satisfies the hashed policies in the index. This process does not reveal any additional information about the original attributes or the plaintext, as it only involves hashed values.

### B. KP-ABSE

Similarly to CP-ABSE, the initial assumption is that the security of the KP-ABSE scheme is the same as it is for the base KP-ABE scheme. We will demonstrate that the additional operations introduced to enable searchability do not compromise the security guarantees provided by the underlying KP-ABE scheme; this will be accomplished by showing that both the confidentiality of the encrypted documents and the privacy of the attributes are maintained.

*1) KP-ABE Security Model:* The security of the $Setup$ algorithm in KP-ABE ensures that it generates public parameters, $PK$, and a master secret key, $msk$. The security condition is that, without access to $msk$, it is infeasible for an adversary to create valid private keys for any access control policy.

For the $KeyGen$ algorithm, it produces a private key $D_A$ corresponding to a specific access policy $A$. The security condition here is that even if an adversary has access to private keys $D_{A1}$, $D_{A2}$, $D_{A3}$, ..., $D_{Ak}$, they cannot derive a valid private key for a new access policy $A'$, unless $A'$ is logically equivalent to one of the known policies $A_i$ where $1 \leq i \leq k$.

In terms of encryption and decryption security, the $Encrypt$ algorithm guarantees that a ciphertext $ct$ encrypted with a set of attributes $\gamma$ can only be decrypted by a private key $D_A$ if the access policy $A$ is satisfied by $\gamma$. The security requirement is that an adversary should not be able to decrypt $ct$ without having an appropriate $D_A$.

*2) KP-ABSE General Scheme Overview:* In addition to the standard KP-ABE setup, KP-ABSE includes an $index$ structure and a hash function $H$ as global system parameters. Algorithmically, it introduces several new components: $CreateIndex$, which is run post-encryption to index the ciphertext based on a hashed version of the attribute set used for encryption; $GenerateQuery$, which produces queries by hashing the attributes in the user's access policy contained in their private key; and $Search$, which matches the hashed query against the hashed attribute sets in the index to return relevant results.

Given the additional algorithms, we must consider the following security proof components:

- The confidentiality of the encrypted documents: Since the encryption and decryption processes in KP-ABSE are the same as in KP-ABE, the confidentiality of the encrypted documents depends on the security of the underlying KP-ABE scheme. Any attack on the confidentiality of KP-ABSE ciphertexts reduces to an attack on KP-ABE ciphertexts. The $index$ only stores hashed versions of the attribute sets, ensuring no extra information about the plaintext or original attributes is revealed. The hash function $H$ must be an irreversible, collision-free one-way function to prevent leakage of information about the original attributes.
- The privacy of the access policies: The security of the hashed attributes in the index and queries relies on the hash function $H$. This function must be chosen such that it is computationally infeasible to derive the original attributes from their hashed values. The $GenerateQuery$ algorithm also uses the hash function $H$, ensuring that the access policies in the user's private key are protected similarly to the attribute sets in the indices.
- The search operation security: The search operation involves checking whether the hashed query satisfies the hashed attribute sets in the index. This evaluation process only uses hashed values and does not reveal any additional information about the original attributes or the plaintext.

In our following work, we plan to formally prove that any adversary who can break the security of a CP-ABSE or KP-ABSE scheme with non-negligible probability can also break the security of the CP-ABE, respectively the KP-ABE on which it is based, which implies that the security of the ASE scheme built on top of the ABE scheme with the keywords acting as attributes idea is at least as strong as the ABE scheme.

## IV. Practical Implementations

The proposed CP-ABSE and KP-ABSE scheme structures are expected to be generally applicable for all existent ABE schemes with the addition of the specific index creation, query generation and search algorithms. In order to test the feasibility and applicability of the transformations proposed, several existent implementations for both CP-ABE and KP-ABE were extended with the aforementioned steps in order for them to become working ASE schemes.

The batch of initial implementations were taken from the open source toolbox library Charm [15], used for prototyping cryptosystems based on a series of provided schemes. Its implementation is done in Python by representatives of Johns Hopkins University as part of their Advanced Research in Cryptography laboratory [16].

From the ABENC [17] schemes package of the library, several schemes were successfully adapted by having the 3 algorithms added to them. The transitioning from ABE to ASE with the 3 steps has not impacted any of the functionality available for the implemented ABE schemes, including user attribute revocation, user access policy adjustments, attribute accountability hiding, policy accountability hiding and other bonus functionalities of the given ABE schemes.

For demonstrative implementations, *SHA256* was used was used as the one-way hash function, due to its efficiency and the fact that it is collision-free, but *SHA256* hashes do not include the salting element. That makes the hashes more susceptible to dictionary-based cyber attacks [18]. Thus, while *SHA256* is more suitable for applications that require frequent interaction,

*bcrypt* [18] is a better solution for safely storing the hashed policies at the expense of efficiency. The proposed extensions are not limiting to a certain hash function, policy structure or index secret document storage method and are meant to be a comprehensible base for the further development of ASE schemes, based on either CP-ABE, or KP-ABE schemes.

Taking that into consideration, the following schemes from the library were successfully adapted and tested:

- DAC-MACS [9], a multi-authority CP-ABE scheme with efficient decryption and authority revocation implementations, proposed by Kang Yang, Xiaohua Jia, K. Ren and B. Zhang;
- The next scheme proposed by Kang Yang, Xiaohua Jia, the Expressive, Efficient, and Revocable Data Access Control for Multi-Authority Cloud Storage CP-ABE scheme [10];
- The KP-ABE lightweight attribute-based encryption scheme for the Internet of things [11], proposed by Xuanxia Yao, Zhi Chen and Ye Tian;
- From Attribute Based Encryption with Privacy Protection and Accountability for CloudIoT [12], proposed by Jiguo Li, Yichen Zhang, Jianting Ning, Xinyi Huang, Geong Sen Poh and Debang Wang, the first proposed pairing-based scheme for policy-hiding CP-ABE;
- One of the first proposed CP-ABE schemes, in Ciphertext-Policy Attribute-Based Encryption [13], by John Bethencourt, Brent Waters; this scheme is a basic pairing-based CP-ABE scheme;
- The Rouselakis - Waters Efficient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption scheme [14], based on a bilinear pairing group of prime order;

The added algorithm which needed the most scheme specific implementation was the $GenerateQuery$ one, given the different formats of the private keys employed by each adapted scheme.

The development environment utilized for this project was PyCharm Community 2024 [19] with Python 3.12, running within a Parallels-managed virtual machine [20] on Ubuntu 22.04. The implementations were developed as prototypes, leveraging the open-source nature of the base ABE implementations designed for educational purposes in Python, which inherently did not prioritize maximum time efficiency.

Benchmark tests conducted post-adaptation indicated inconsistencies in execution times across multiple trials of the same code. Notably, there were instances where the more complex ASE implementations executed faster than expected, suggesting the influence of external factors related to the development environment. Consequently, these benchmark results should be interpreted with caution and are not wholly reliable.

## V. Acknowledgments

I would like to express my gratitude towards my MSc dissertation supervisor, Prof. Dr. Ferucio Laurențiu Țiplea, for the guidance and support throughout the course of this research. Prof. Dr. Ferucio Laurențiu Țiplea's advice and insightful feedback were instrumental in shaping the direction and outcome of this study.

## VI. Conclusion

The formalization of transforming Attribute-Based Encryption (ABE) schemes into Asymmetric Searchable Encryption (ASE) schemes is an important step forward in the field of cryptography. By treating attributes in ABE as keywords in ASE, efficient search capabilities are introduced, all while preserving the robust security properties of ABE. This theoretical framework for Ciphertext-Policy Attribute-Based Searchable Encryption (CP-ABSE) and Key-Policy Attribute-Based Searchable Encryption (KP-ABSE) demonstrates the feasibility and practicality of this approach, as evidenced by our successful adaptation of several existing ABE schemes.

The importance of this formalization lies in its ability to enable more secure and flexible search functionalities in encrypted databases, which is more and more necessary for multi-user environments with diverse access control requirements. Our analysis shows that the introduction of the index, query generation, and search algorithms does not compromise the confidentiality of encrypted documents or the privacy of access policies and does not introduce high performance overhead.

Moving forward, we aim to provide the formal security proof for both CP-ABSE and KP-ABSE schemes and further refine and optimize the three added steps: $CreateIndex$, $GenerateQuery$, and $Search$. Our goal is to enhance their efficiency, making them even more suitable for practical applications. Additionally, we plan to develop two specific algorithms based on this formalization: one for Key-Policy Attribute-Based Searchable Encryption (KP-ABSE) and one for Ciphertext-Policy Attribute-Based Searchable Encryption (CP-ABSE); and test performance benchmarks on relevant data. These algorithms will leverage the strengths of their respective ABE schemes, providing tailored solutions for different use cases and further advancing the capabilities of searchable encryption.

## References

[1] Dawn Xiaoding Song, D. Wagner and A. Perrig, *Practical techniques for searches on encrypted data*, in Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000, Berkeley, CA, USA, pp. 44-55, 2000, https://dx.doi.org/10.1109/SECPRI.2000.848445

[2] D. Boneh, G. Di Crescenzo, R. Ostrovsky and G. Persiano, *Public Key Encryption with Keyword Search*, in: Cachin, C., Camenisch, J.L. (eds) Advances in Cryptology - EUROCRYPT 2004. EUROCRYPT (Lecture Notes in Computer Science), vol 3027. Springer, Berlin, Heidelberg, 2004, https://doi.org/10.1007/978-3-540-24676-3_30

[3] B. G. Pillai and N. Dayanand Lal, *Blockchain-Based Searchable Asymmetric Encryption Scheme in Cloud Environment*, in 2023 International Conference on Applied Intelligence and Sustainable Computing (ICAISC), Dharwad, India, pp. 1-6, 2023 https://dx.doi.org/10.1109/ICAISC58445.2023.10201090

[4] M. Wang, L. Rui, S. Xu, Z. Gao, H. Liu and S. Guo, Shaoyong, *A multi-keyword searchable encryption sensitive data trusted sharing scheme in multi-user scenario*, in Elsevier North-Holland, Inc., USA, vol. 237, no. C, 2023 https://doi.org/10.1016/j.comnet.2023.110045

[5] M. Wang, L. Rui, S. Xu, Z. Gao, H. Liu and S. Guo, Shaoyong, *A multi-keyword searchable encryption sensitive data trusted sharing scheme in multi-user scenario*, in Elsevier North-Holland, Inc., USA, vol. 237, no. C, 2023 https://doi.org/10.1016/j.comnet.2023.110045

[6] R. Zhang, R. Xue, T. Yu and L. Liu, *PVSAE: A Public Verifiable Searchable Encryption Service Framework for Outsourced Encrypted Data*, in 2016 IEEE International Conference on Web Services (ICWS), San Francisco, CA, USA, pp. 428-435, 2016, https://dx.doi.org/10.1109/ICWS.2016.62

[7] L. Meng, L. Chen, Y. Tian, M. Manulis and S. Liu, *FEASE: Fast and Expressive Asymmetric Searchable Encryption*, in Cryptology ePrint Archive, Paper 2024/054, 2024, https://eprint.iacr.org/2024/054, in press

[8] T. Yarl, B.M. Goi, R. Komiya and S.Y. Tan, *A Study of Attribute-Based Encryption for Body Sensor Networks*, in Informatics Engineering and Information Science, vol. 251, pp. 238-247, 2011, https://doi.org/10.1007/978-3-642-25327-0_21

[9] K. Yang, X. Jia, K. Ren and B. Zhang, *DAC-MACS: Effective data access control for multi-authority cloud storage systems*, in 2013 Proceedings - IEEE INFOCOM, Turin, Italy, pp. 2895-2903, 2013, https://dx.doi.org/10.1109/INFCOM.2013.6567100

[10] K. Yang, X. Jia, *Expressive, Efficient, and Revocable Data Access Control for Multi-Authority Cloud Storage*, in 2014 IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 7, pp. 1735-1744, 2014, https://dx.doi.org/10.1109/TPDS.2013.253

[11] X. Yao, Z. Chen, Y. Tian, *A Lightweight Attribute-Based Encryption Scheme for the Internet of Things*, Future Generation Computer Systems, vol. 49, pp. 104-112, 2015, https://dx.doi.org/10.1016/j.future.2014.10.010

[12] J. Li, Y. Zhang, J. Ning, X. Huang, G. S. Poh, D. Wang, *Attribute Based Encryption with Privacy Protection and Accountability for CloudIoT*, in 2022 IEEE Transactions on Cloud Computing, vol. 10, no. 2, pp. 762-773, 2022, https://dx.doi.org/10.1109/TCC.2020.2975184

[13] J. Bethencourt, A. Sahai, B. Waters, *Ciphertext-Policy Attribute-Based Encryption*, in 2007 IEEE Symposium on Security and Privacy (SP), pp. 321-334, 2007, https://dx.doi.org/10.1109/SP.2007.11

[14] Y. Rouselakis and B. Waters, *Efficient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption*, in Financial Cryptography and Data Security (Lecture Notes in Computer Science), vol. 8975, Springer, Berlin, Heidelberg, Germany, 2015, https://dx.doi.org/10.1007/978-3-662-47854-7_19

[15] Charm: A Framework for Rapidly Prototyping Cryptosystems, https://github.com/JHUISI/charm/blob/dev/LICENSE.txt

[16] John Hopkins University: Advanced Research in Cryptography laboratory, https://arc.isi.jhu.edu/

[17] Charm ABENC schemes package, https://github.com/JHUISI/charm/tree/dev/charm/schemes/abenc

[18] M. Grigutytė, *What is bcrypt and how does it work?* , in NordVPN Blog, 2023, https://nordvpn.com/blog/what-is-bcrypt/

[19] PyCharm, https://www.jetbrains.com/pycharm/

[20] Parallels, https://www.parallels.com/