ACKNOWLEDGEMENT

ABSTRACT

KEYWORDS

TABLE OF CONTENTS

1.INTRODUCTION

Cloud security refers to the set of procedures, technologies, and policies designed to protect data, applications, and infrastructure in cloud computing environments. As organizations increasingly adopt cloud services for their computing needs, ensuring the security of data and resources stored and processed in the cloud becomes paramount. Cloud computing offers the possibility of providing suitable access within a network for a set of resources. Many users use different services for outsourcing their data within the cloud, saving and mitigating the local storage and other resources involved. As security concerns rise worldwide, so does investment in new technologies and services to protect data from falling into the wrong hands. And if we can't entirely stop hackers from accessing information or even internal leaks, then preventing the ability to read the data is an essential part of protecting privacy. That brings us to the all-too-familiar need for encryption and cryptographic storage.

Many cryptographic methods have been devised to address the issue of confidentiality and privacy of owner's data in cloud. Searchable encryption is one such technique that helps us deal with these security issues and provides a solution.

Searchable encryption is a cryptographic technique that enables searching over encrypted data without revealing the contents of the data or the search query to the server or any third-party observer. It allows users to perform keyword searches on encrypted data stored in the cloud or other remote servers while maintaining privacy and confidentiality.In general, searchable encryption schemes aims to provide confidentiality and integrity, while retaining main benefits of cloud storage - availability, reliability, data sharing, and ensuring requirements through cryptographic guarantees rather than administrative controls.

2. LITERATURE SURVEY

2.1 HISTORY:

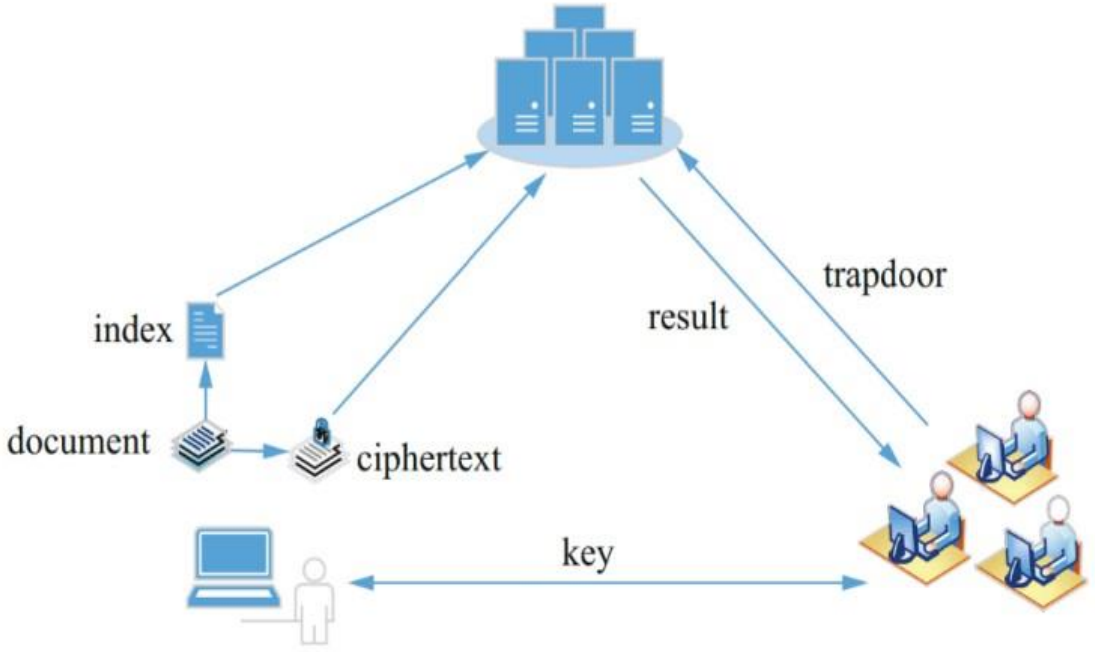
The problem of searching on encrypted data was considered by Song, Wagner and Perrig[1] though previous work on Oblivious RAM by Goldreich and Ostrovsky[4] could be used in theory to address the problem. This work[1] proposed an SSE scheme with a search algorithm that runs in time , where . Goh[5] and Chang and Mitzenmacher[6] gave new SSE constructions with search algorithms that run in time , where is the number of documents. Curtmola, Garay, Kamara and Ostrovsky[2] later proposed two static constructions with search time, where is the number of documents that contain , which is optimal. This work also proposed a semi-dynamic construction with search time, where is the number of updates. An optimal dynamic SSE construction was later proposed by Kamara, Papamanthou and Roeder.[7] Goh[5] and Chang and Mitzenmacher[6] proposed security definitions for SSE. These were strengthened and extended by Curtmola, Garay, Kamara and Ostrovsky[2] who proposed the notion of adaptive security for SSE. This work also was the first to observe leakage in SSE and to formally capture it as part of the security definition. Leakage was further formalized and generalized by Chase and Kamara.[8] Islam, Kuzu and Kantarcioglu described the first leakage attack.[9] All the previously mentioned constructions support single Searchable Encryption: When storing data remotely, secrecy is essential. Encrypting the informa- tion stored remotely has been standard from the start. The use of remote storage servers, also called Cloud Servers, is increasing rapidly for personal use and businesses. The standard way of storing information remotely has been to encrypt during transmission and then let the server decrypt and then re-encrypt the data before storing it safely. Allowing the server to use its own encryption makes sense and enables many valuable features for the users. However, this comes at the cost of letting the server view the user's data. In today's world, it is safer to assume that corporations are willing to misuse their

user's information if they can get away with it. As users, this is practically impossible to stop. Therefore it is better to avoid the situation altogether. One solution to this problem is a type of encryption called Searchable Encryption (SE) which allows for searching on encrypted data. The two main branches of SE are searchable symmetric encryption and searchable asymmetric encryption [44].

2.2 TYPES OF SEARCHABLE ENCRYPTION:

SEARCHABLE SYMMETRIC ENCRYPTION:

Searchable Symmetric Encryption (SSE) is a type of encryption that makes it possible to make hidden searches on encrypted data. Suppose a user, Alice wants to store a set of documents on a remote server. As Alice does not want the server to be able to view the contents of her documents, she encrypts her files before uploading them to the server. Whenever Alice wants to retrieve any documents containing a specific keyword, she can generate a special token(later referred to as a trapdoor) and ask the server to search for the token in her encrypted documents. The server never learns anything about the keyword or the documents.



The general structure of SSE schemes consists of four main functions: keygen, buildIndex, trapdoor, and search

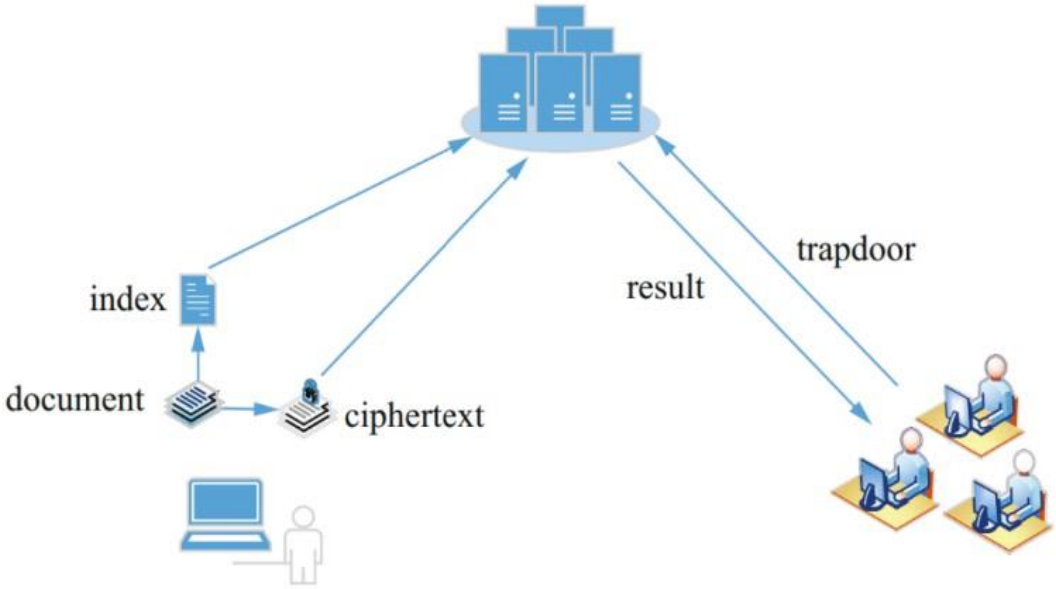
- Keygen: Generates a private key. Takes a security parameter as input- (usually the keysize of the system) and outputs the private key.
- BuildIndex: Generates an index for a file or a set of files. Takes the private key and file(s) as input and outputs the index.
- Trapdoor: Generates a trapdoor for a keyword. A trapdoor is the encoded (hidden) version of a keyword. Takes a keyword and the private key as input and returns the trapdoor for the given keyword.
- Search: Finds all documents that contain the provided trapdoor. Takes an index or a set of indices and a trapdoor as input and returns a set of documents (or document references/identifiers).

During the initialization of a new user, the keygen algorithm is called by the data owner (can be the same person as the 'user'), and the private key is stored locally or distributed lo- cally to the desired users. When the user wants to store a file, they generate its index(locally), encrypt the file with a normal symmetric encryption algorithm, and upload both the index and the file to the server. Depending on the scheme, there are either one index per document or one index for all document. When the user wants to search for a keyword, the user generates a trapdoor locally with the private key and sends the trapdoor to the server. The server calls the search function and returns any matching documents. The user can then decrypt the files with their private key.

SEARCHABLE ASYMMETRIC ENCRYPTION :

Searchable asymmetric encryption, also called Public-key encryption with keyword search (PEKS) works similarly to SSE but in a public-key setting. Suppose Alice uses several different devices to read her emails. Alice wants any emails containing the word 'urgent' to be sent directly to her phone, and the rest should be sent to her laptop. Like in the SSE example, Alice can create a token

and send this to the server. In this example, Alice uses her private key to create a token with the keyword 'urgent' and sends this to the server with the instruction to send emails with a matching token to be sent to her phone. Now if another user wants to send an email to Alice, they encrypt the email with a relevant keyword(for example 'urgent') and with Alice's public key and send it to her. The server receives the email and can check if it has a matching token and route it accordingly. Alice then uses her private key to decrypt the email.



The general structure of PEKS scheme :

- Keygen: Generates a private/public key pair. Takes a security parameter as input and outputs the private/public key pair.
- PEKS: Encrypts a document while preserving searchability for a specific user. Takes the recipient's public key and a keyword as input, and returns an encrypted document with searchability for that keyword.
- Trapdoor: Generates a trapdoor for a keyword. Takes a user's private key and a keyword and returns a token/ trapdoor for the given keyword.

- Test: Tests if an encrypted document was encrypted with a specific token or not. Takes a user's private key, the encrypted document, and a token as input, and returns 'true' if the document and the token were encrypted with the same keyword, otherwise 'false'.

2.3 DEVELOPMENT OF SEARCHABLE SYMMETRIC ENCRYPTION

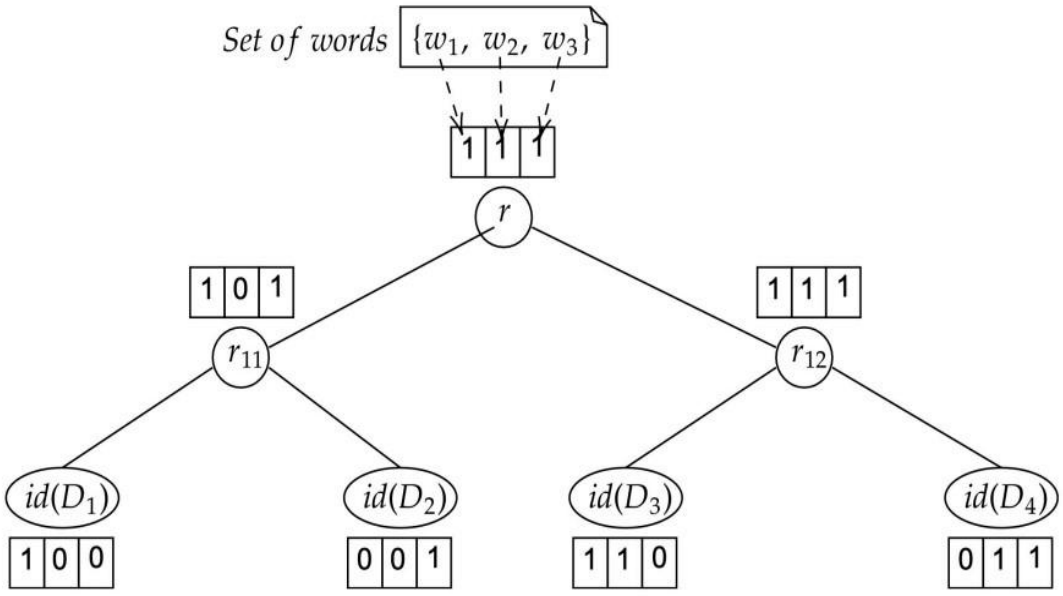
Initially, the only way to hide the contents of data being stored remotely was to encrypt the data before uploading. This approach worked but was highly ineffective as the user would have to download all the data, decrypt it and search locally every time they wanted to search for something specific in their documents. To solve this problem, Song et al. [41] proposed a scheme that made it possible to partially decrypt files on the server without revealing the file's contents. In this scheme, the user keeps a special private key which is used for both encrypting files and creating trapdoors(called search tokens in the paper). The trapdoors disclose no information about the search contents and can only be created by anyone with the correct private key. [39] In the field of SSE, there are several different research branches focusing on different techniques of searching on encrypted data. The main branches include searching with a single keyword, fuzzy keyword search, conjunctive keyword search, ranked keyword search, and verifiable keyword search.

1. SINGLE KEYWORD SEARCH :

In the branch of single keyword search, there have been proposed several ways of searching, mainly how to structure the index table used to perform the lookup of the trapdoor. As mentioned, the first scheme to use SSE. This scheme does not use an index but performs the search directly on the encrypted text. The encryption is performed in a way that makes it possible for the server, when provided with a trapdoor to partially decrypt and check if the trapdoor is present in the encrypted text. Because of its clever design, only the user with the private key can generate a trapdoor from a search word. Neither the trapdoor nor the

partially decrypted ciphertext reveal anything about its contents to the server. As the search is performed directly on the encrypted text and the server has to iterate over the entire document, the search time scales linearly with the length of the document.

The most recent addition to the development of single keyword search is dynamic SSE schemes. Van Liesdonk et al. [33] were the first to tackle the problem of making updates more efficient. In their paper, they propose two variants of their scheme with different qualities regarding search time and storage requirements. The first variant is interactive and the second is not interactive. Kamara and Papamanthou [28] proposed a new scheme that uses a three-structure as the index. The main difference between this method and the previous methods is that instead of maintaining a map of [trapdoor, document] pairs, the user constructs a search tree where the nodes in the tree are used to perform the search



2. FUZZY KEYWORD SEARCH:

With the single keyword search technique, the search word has to match the stored keywords exactly. Any typo or small inconsistency in the search word will result in the search failing. Fuzzy keyword search tackles this problem by being

able to handle minor differences in the search word and the stored keyword. Li et al. [32] design and utilize a 'wildcard-based' technique related to the concept of edit distance to construct a fuzzy keyword set. They also propose a fuzzy search scheme utilizing the construction of the fuzzy keyword set. Adjedj et al. [13] use their fuzzy keyword scheme to perform fast and secure biometric identification. Like Kuzu et al. [30], they use locality-sensitive hashing (LSH), which will with a very high probability output the same value for two inputs with a small matching score(based on Hamming distance).

3. CONJUCTIVE KEYWORD SEARCH:

In both single keyword search and fuzzy keyword search, the user provides a trapdoor to the server and receives the documents containing the key- word represented by that trapdoor. If a user wants to search for documents containing multiple keywords, they either have to search with all keywords separately and then locally separate the desired documents, or somehow em- bed several keywords into each trapdoor when generating the index. Both options are far from ideal. The first option leaks a substantial amount of information to the server, and the second option would make the size of each index scale exponentially. To solve this problem, Golle et al. [26] proposed two schemes with con- junctive keyword search. The first scheme has communication costs linear to the number of documents but the work can be done offline before the request is sent to the server. The second scheme's search cost is on the order of the number of keywords and does not require any offline work. Cash et al. [18] solve the problem of searching for multiple keywords with boolean queries. Their scheme is not the fastest or most secure but provides a realistic and practical trade-off between security and efficiency. Faber et al. [24] propose an extension to the scheme proposed by Cash et al. [18], adding support for substring-, wildcard-, phase-, and range queries. The new query types do add some cost in performance

and storage, but the authors claim that the extension is still practical today, even for large databases.

4.RANKED KEYWORD SEARCH:

With a ranked keyword search, only the most relevant documents are returned during a query. This can be used to make systems more effective and reduce unnecessary network traffic. Zerr et al. [46] present a ranking model used to create a relevance score transformation function. This lets a server return the most relevant results for a user query without revealing any information about the indexed data. Cao et al. [17] propose the first multi- keyword scheme with ranked search. Xia et al. [45] proposedscheme with the same multi-keyword ranked search attribute, but also features efficient updates and deletions. The scheme uses a three-structure index and uses a ”Greedy Depth-first Search” algorithm for searching.

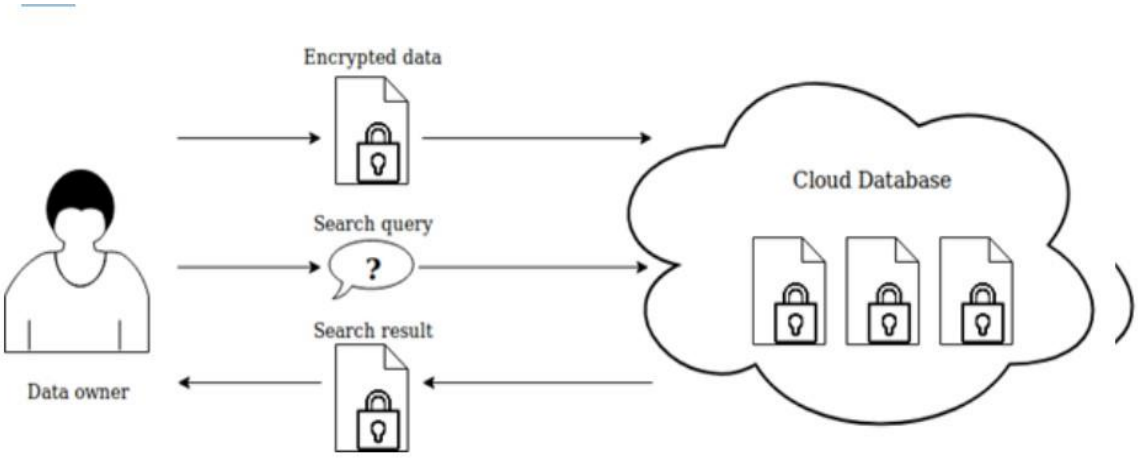
5. VERIFIABLE KEYWORD SEARCH:

With a verifiable keyword search, the recipient can check whether the result of a query is complete and correct. This attribute is widely used in general internet communication and helps avoid unnoticed hardware or software errors, as well as protecting the user from semi-honest servers trying to save computation resources. Chai and Gong [19] proposed the first verifiable SSE scheme. The scheme uses a trie-like (prefix tree) structure as the index. This index is used to search, and produce proof that the returned results are valid and complete. Li et al. [31] discuss an aspect of query authentication called query freshness that previously has not been explored. Query fresh- ness means being able to verify that the search result comes from the latest version of the database. Kurosawa and Ohtaki [29] propose a verifiable SSE scheme with a focus on security against

active adversaries, as opposed to the more common perspective of a passive (honest-but-curious server) adversary.

3. EXISITING APPROACH OR METHOD

The first Searchable Encryption scheme (and the first formal scientific definition of searchable encryption scheme) was proposed by Song, Wagner, and Perrig in 2000. The problem of search in encrypted data has since received much interest from governments, academic researchers, and the industrial sector such as Google's Encrypted BigQuery, Microsoft's SQL Server 2016, Azure SQL Database, Oracle Database Cloud, Amazon RDS, etc. A typical workflow for a searchable encryption scheme would look something like this:



Abstract searchable encryption scheme

3.1 BASIC METHODS OF SEARCHING OVER ENCRYPTED DATA:

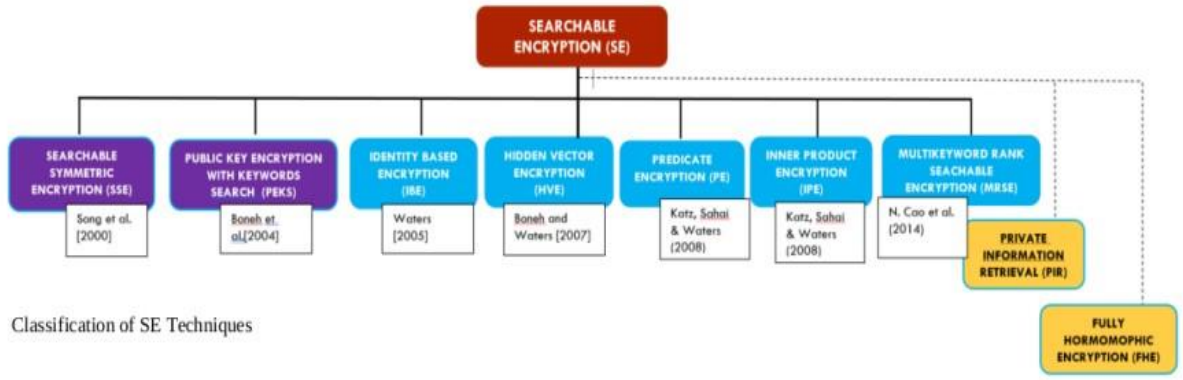
Decrypt, search, encrypt: The most basic approach to searching through encrypted data is to download the data to the client's computer, decrypt it locally, and then search for the desired results in the plaintext data. For many reasons, this approach is neither practical nor scalable due to the possible significant data size. The key management problems: How to handle key distribution properly when the client needs the decryption key and can consequently leak it and the access credentials to an encrypted storage. Also, the emergence of a plaintext copy of

the encrypted data in the "decrypt+search" scenario renders the encryption useless as the attacker can find a way to intercept and steal the plaintext copy.

Decrypt, run query, send results: Another method lets the application backend (server side) request encrypted data from the database, decrypt it, run a search query on the server side, and send only the results back to the user. Similarly to the scheme above, this introduces key management problems as the server now has both data and decryption key. In this case, security is compromised as all records protected by encryption will be revealed to every service and the attacker has access to the server environment.

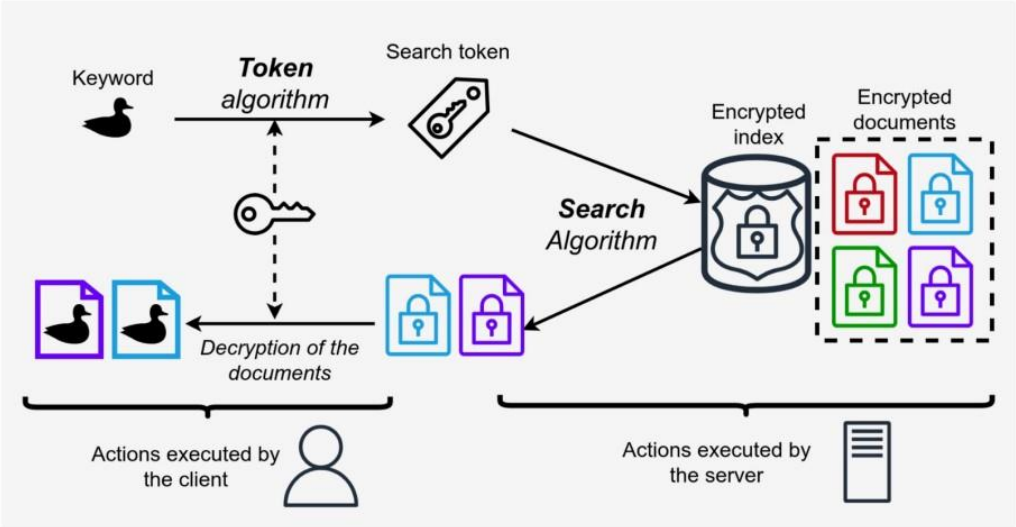
3.2 CRYPTOGRAPHIC SCHEMES FOR SEARCH ON ENCRYPTED DATA

There are two ways to implement searchable encryption (SE). It can be either done through encrypting data in a way that enables execution of queries' directly on the ciphertext or by introducing a searchable index stored together with the encrypted data. In a properly implemented secure search scheme, the database server and application backend (server-side) never ever sees the decrypted data.



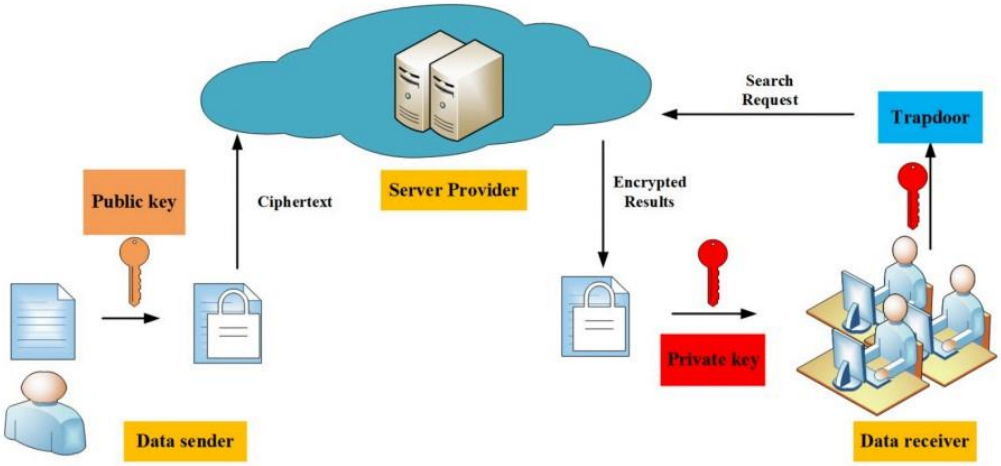
Classification of SE Techniques

The main cryptographic techniques for constructing searchable encryption schemes are symmetric searchable encryption (SSE) and public key encryption with keyword search (PEKS)



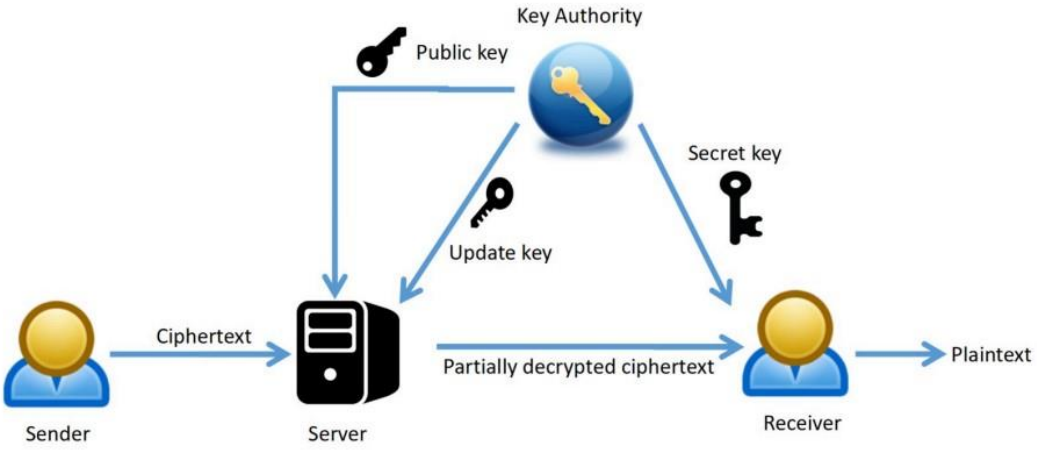
Searchable Symmetric Encryption

Searchable Symmetric Encryption (SSE) is a widely used approach in searchable encryption. SSE-based schemes use symmetric key primitives, allowing users to create ciphertexts and perform secure searches. The primary advantage of SSE is its computational efficiency, making it a practical choice for many applications. In an SSE scheme, the data owner first encrypts the data and then uploads it to the server. When the data owner wants to search for a particular keyword, they generate a search token using their secret key and the keyword. This search token is sent to the server, which then uses it to retrieve all the documents that contain the keyword. However, SSE has its limitations. One of the main drawbacks is that it only allows the secret key owners to perform search operations. This means that if the data owner wants to share the encrypted data with others, they would also have to share their secret key, which could potentially compromise the security of the data.



Public-Key Encryption with Keyword Search

Public-Key Encryption with Keyword Search (PEKS) is another approach to searchable encryption. Unlike SSE, PEKS schemes use asymmetric cryptography, which means they have separate keys for encryption and decryption. This allows the data owner to share the encrypted data with multiple users without having to share their secret key. In a PEKS scheme, the data owner first encrypts the data using their public key and then uploads it to the server. When a user wants to search for a particular keyword, they generate a search token using the data owner's public key and the keyword. This search token is sent to the server, which then uses it to retrieve all the documents that contain the keyword. While PEKS offers more flexibility than SSE, it is also more computationally expensive. This makes it less practical for applications that require fast search times.

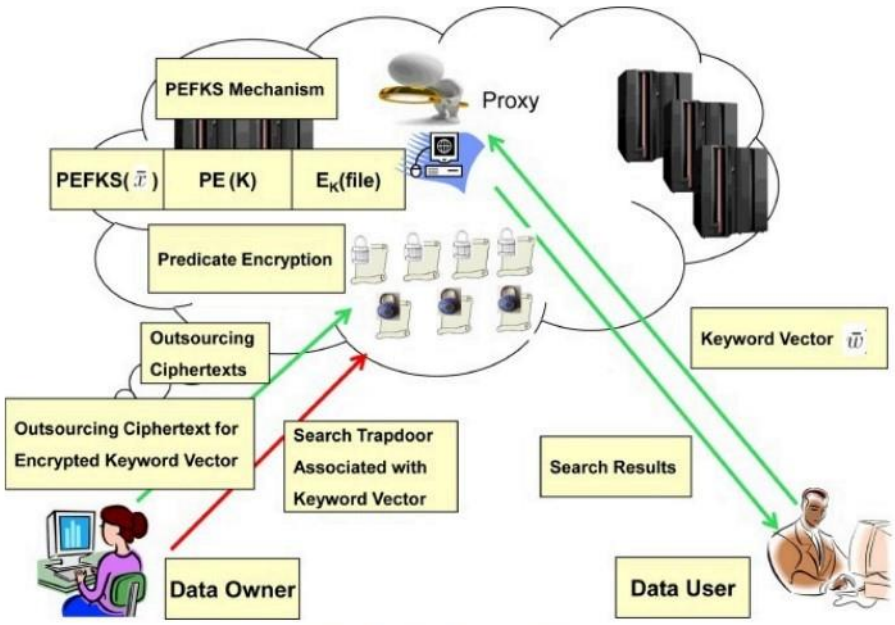


Identity-Based Encryption

Identity-Based Encryption (IBE) is a type of public-key encryption where the public key of a user is some unique information about the identity of the user (e.g., a user's email address). This means that a sender who has access to the public parameters of the system can encrypt a message using the receiver's name or email address as a key. The receiver obtains its decryption key from a central authority, which needs to be trusted as it generates secret keys for every user.

HIDDEN VECTOR ENCRYPTION(HVE)

Hidden Vector Encryption (HVE) is a special kind of predicate encryption. HVE supports the evaluation of conjunctive equality, comparison, and subset operations between attributes in ciphertexts and attributes in tokens. It allows the query keyword set to contain a wildcard keyword by using Bloom filter (BF). Using a hierarchical clustering algorithm, a clustering Bloom filter tree (CBF-Tree) is constructed, which improves the efficiency of wildcard search.

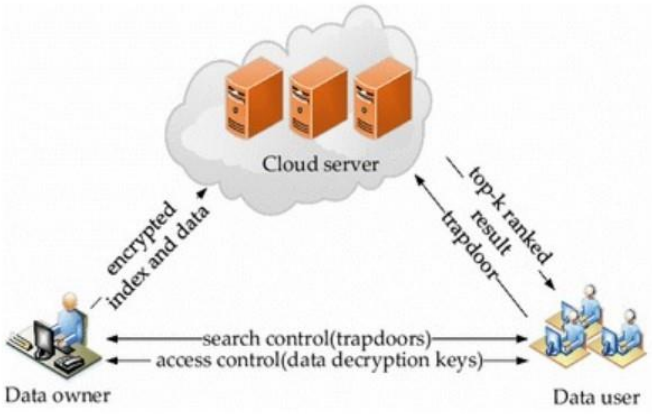


Predicate Encryption

Predicate Encryption (PE) is a new paradigm for public-key encryption that generalises identity-based encryption and more. In predicate encryption, secret keys correspond to predicates and ciphertexts are associated with attributes; the secret key SKf corresponding to a predicate f can be used to decrypt a ciphertext associated with attribute I if and only if f(I) = 1.

INNER PRODUCTION ENCRYPTION (IPE)

Inner Product Encryption (IPE) is a type of functional encryption where the decryption algorithm, given a ciphertext related to a vector x and a secret key related to a vector y, computes the inner product x · y. This means that the decryption will be correct if and only if the attribute vector and the predicate vector satisfy the inner product predicate, meaning that the inner product operation of x and y equals zero. Over the past decade, many IPE schemes have been proposed, such as those based on pairing and lattice. The security definition of an IPE scheme can be naturally extended from the IND–CPA security of identity-based encryption.



Multi-Keyword Ranked Searchable Encryption

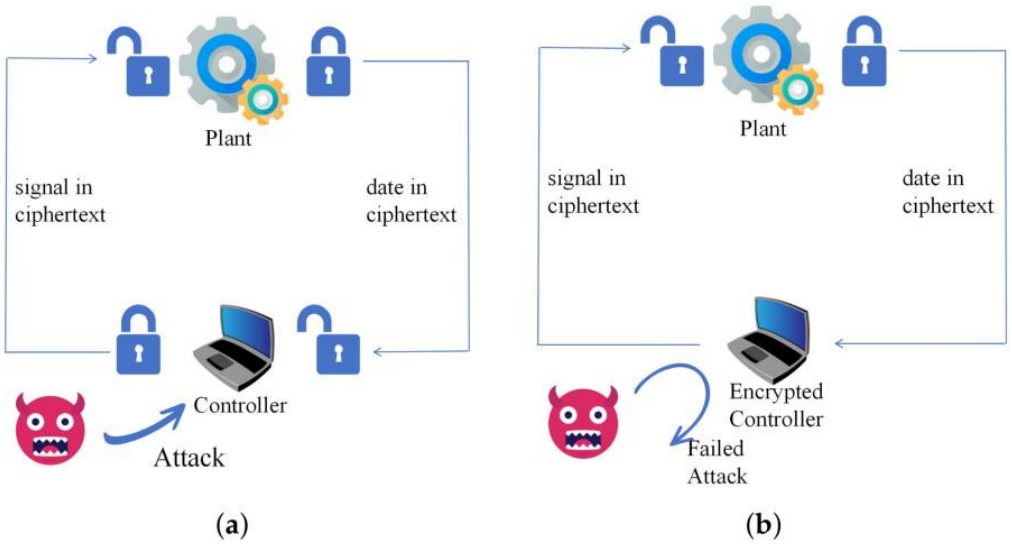
Multi-Keyword Ranked Searchable Encryption (MRSE) supports multi-keyword contained in one query and returns the top-k search results related to the query keyword set. It realised effective search on encrypted data. Most previous works about MRSE can only do the complete keyword search and rank on the server-side. However, with more practice, users may not be able to express some keywords completely when searching. Server-side ranking increases the possibilities of the server inferring some keywords queried, leading to the leakage of the user's sensitive information.

PRIVATE INFORMATION RETRIEVAL

In cryptography, a private information retrieval (PIR) protocol is a protocol that allows a user to retrieve an item from a server in possession of a database without revealing which item is retrieved. PIR is a weaker version of 1-out-of-n oblivious transfer, where it is also required that the user should not get information about other database items. One trivial, but very inefficient way to achieve PIR is for the server to send an entire copy of the database to the user. In fact, this is the only possible protocol (in the classical or the quantum setting) that gives the user information theoretic privacy for their query in a single-server setting.There are two ways to address this problem: make the server

computationally bounded or assume that there are multiple non-cooperating servers, each having a copy of the database.

FULLY HOMOMORPHIC ENCRYPTION:



(a) without encrypted controller (b) with encrypted controller

Fully Homomorphic Encryption (FHE) is a powerful cryptographic primitive which allows parties to compute with encrypted data, without the need of decrypting it first. It has applications in finance, health care systems, image and text classification, machine learning, electronic voting, and multiparty computation. FHE allows operations to be performed on encrypted data without ever decrypting it, offering an unprecedented level of security. This means that in the event of a data leak, the exposed information would be rendered meaningless without the encryption key, which stays securely in the hands of the owner. One of the main benefits of FHE is that it eliminates the tradeoff between data usability and data privacy. There is no need to mask or drop any features in order to preserve the privacy of data. All features may be used in an analysis, without compromising privacy. However, FHE is still emerging and struggles with poor performance, making it commercially infeasible for computationally-heavy applications.

SEARCH FUNCTIONALITIES :

Searchable encryption schemes support various search functionalities to meet users' diverse information requirements. Some of the main search functionalities include Boolean Search (BS), fuzzy search (FS), and Ranked Search (RS). Boolean Search allows users to search for documents that contain a specific set of keywords. This is useful for applications where the user knows exactly what they are looking for. Fuzzy Search allows users to search for documents that contain keywords that are similar to the search query. This is useful for applications where the user may not know the exact spelling of the keyword. Ranked Search allows users to search for documents that are most relevant to the search query. This is useful for applications where the user wants to see the most relevant documents first.

VULNERABILITIES AND ATTACKS :

Existing encryption schemes adopted by searchable encryption schemes have vulnerabilities that could lead to possible attacks. For instance, information leakages are possible from the encrypted data. This could potentially allow an adversary to learn sensitive information about the data, even though it is encrypted. There are several types of attacks that can be launched against searchable encryption schemes. These include known-ciphertext attacks, known-keyword attacks, and chosen-keyword attacks. In a known-ciphertext attack, the adversary has access to the ciphertexts but not the secret key. In a known-keyword attack, the adversary knows some of the keywords that are used in the search queries. In a chosen-keyword attack, the adversary can choose the keywords that are used in the search queries.

COUNTERMEASURES:

Various countermeasures have been proposed to prevent attacks on searchable encryption schemes. These include using stronger encryption algorithms, implementing access control mechanisms, and using secure multi-party computation techniques.

DATABASE SPECIFIC TOOLS FOR SEARCH IN ENCRYPTED DATA:

There are several existing tools designed specifically for searching in encrypted data, such as CryptDB, Mylar, CipherSweet, and Acra Searchable Encryption. These tools provide various features, such as full-text search, range queries, and join operations, making them suitable for a wide range of applications.

4. PROPOSED APPROACH OR METHOD:

4.1 DESIGN :

4.2IMPLEMENTATION:

1.MACHINE LEARNING APPROACH:

ALGORITHM :

Implementing searchable encryption using machine learning involves combining cryptographic techniques with machine learning models to enable searching over encrypted data without compromising privacy. Here's a simplified illustration of how you might implement such a system:

- Data Preprocessing:

  - Begin by preprocessing your data using machine learning techniques.

This step might involve feature extraction, dimensionality reduction, or other transformations depending on the type of data you're dealing with.

- For instance, if you're working with text data, you might tokenize the text, remove stop words, and apply techniques like TF-IDF or word embeddings to represent documents

- Machine Learning Model Training:

-Train a machine learning model on the preprocessed data. The choice of model depends on your specific application and the nature of the data.

-For example, if you're working with textual data, you might use algorithms like logistic regression, support vector machines, or neural networks for classification or regression tasks.

- Integration with Encryption:

- After training your machine learning model, integrate it with the encryption process. This integration should ensure that the model can operate on encrypted data without revealing sensitive information.

- Choose an appropriate encryption scheme that supports searchable encryption, such as Order-Preserving Encryption (OPE), Deterministic Encryption (DE), or Homomorphic Encryption (HE).

- Depending on the encryption scheme, you might need to adapt your machine learning model to operate within the constraints imposed by encryption, such as restricted operations or formats.

- Encrypting the Data:

- Encrypt your data using the chosen encryption scheme while preserving the necessary information for search operations. This ensures that the data remains secure even while stored or transmitted.

- Searchable Encryption Setup:

- Set up your searchable encryption system, which allows users to perform search operations on the encrypted data without revealing the underlying plaintext.

- Implement algorithms for encrypted search operations, such as keyword search, range queries, or similarity searches, depending on your application requirements.

- Searchable Encryption Operations:

- Users can now interact with the searchable encryption system to perform search queries over the encrypted data.

- When a user submits a search query, the query is encrypted using the same encryption scheme used to encrypt the data.

- The encrypted query is then processed by the server without decrypting the underlying data, and relevant results are returned to the user.

- Evaluation and Optimization:

- Continuously evaluate the performance and security of your searchable encryption system.

- Optimize your machine learning models, encryption schemes, and search algorithms to improve efficiency, accuracy, and security while minimizing computational overhead and potential vulnerabilities.

- Privacy and Security Considerations:

  - Pay close attention to privacy and security considerations throughout the implementation process.

  - Ensure that sensitive information is adequately protected against potential attacks or breaches, and that user privacy is maintained even during search operations.

By integrating machine learning with searchable encryption techniques, you can enable powerful search capabilities while ensuring the confidentiality and privacy of sensitive data. However, it's crucial to carefully design and implement the system to balance functionality, efficiency, and security.

PSEUDO-CODE:

```
# Import necessary libraries

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine_similarity

from cryptography.fernet import Fernet

# Training phase

def train_model(training_data):

    vectorizer = CountVectorizer()

    X = vectorizer.fit_transform(training_data)

    return vectorizer, X
```

```
# Encryption phase
def encrypt_data(data, key):
    f = Fernet(key)
    encrypted_data = f.encrypt(data.encode())
    return encrypted_data
# Search phase
def search(query, encrypted_documents, vectorizer):
    decrypted_documents = []
    for encrypted_doc in encrypted_documents:
        decrypted_doc = decrypt_data(encrypted_doc)
        decrypted_documents.append(decrypted_doc)
    # Transform query into vector representation
    query_vector = vectorizer.transform([query])
 # Compute cosine similarity between query and documents
    similarities = []
    for doc in decrypted_documents:
        doc_vector = vectorizer.transform([doc])
        similarity = cosine_similarity(query_vector, doc_vector)[0][0]
```

```
        similarities.append(similarity)
    # Return documents sorted by similarity
    return sorted(zip(decrypted_documents, similarities), key=lambda x: x[1], reverse=True)
# Decryption phase
def decrypt_data(encrypted_data):
    f = Fernet(key)
    decrypted_data = f.decrypt(encrypted_data).decode()
    return decrypted_data
# Example usage
if _name_ == "_main_":
    # Training data
    training_data = ["document 1 text", "document 2 text", "document 3 text"]
# Train machine learning model
    vectorizer, _ = train_model(training_data)
# Encrypt documents
    key = Fernet.generate_key()
    encrypted_documents = [encrypt_data(doc, key) for doc in training_data]
```

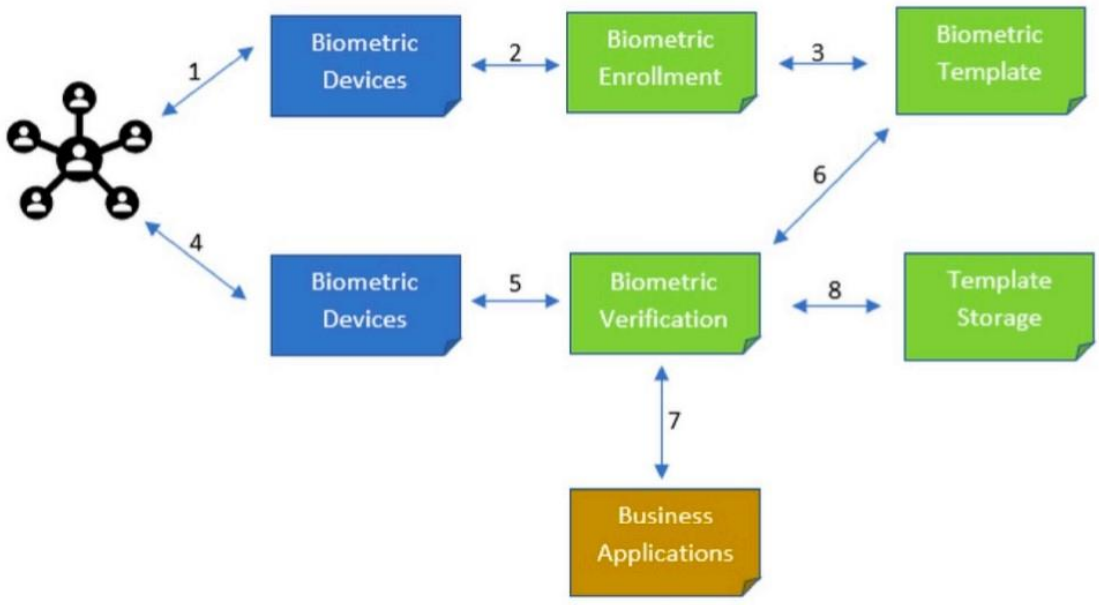
```
# Search query

query = "search query"

# Search

results = search(query, encrypted_documents, vectorizer)

# Print results

for doc, similarity in results:

    print(f"Similarity: {similarity}, Document: {doc}")
```

2.BIOMETRIC AUTHENTICATION:

The identification of users by utilizing user IDs, ID cards, passwords, tokens, or other elements of user identification has always represented an important challenge in designing a complex system. This kind of technique has been proven to be easily corrupted through different attacks. We have seen different attacks take place with success—the password being easily guessed due to user carelessness with regard to password security, ID cards that can be easily stolen, and so on Biometrics represents the process of being able to identify a person ’s identity based on their physiological and behavioral characteristics, managing the process of determining the difference between an authorized person and a fake or unauthorized one. Within our research, we started from the idea that the system should be able to identify a person based on their data, by querying through all the characteristics and checking it against those characteristics already stored in the database as hash values (note that there is nothing stored in clear in our database). Finding a person is accomplished using two directions: identification and verification. The enrolment of the users in the current system

is carried out in eight phases (see Figure 1), with respect to the idea proposed . These phases can be summarized as follows:

(1) capture the biometrics data,
(2) process, enroll and extract the biometric template,
(3) store the data in different tokens,
(4) conduct real-time scanning of the chosen biometric,
(5) process the biometric data and extract the biometric template,
(6) match the biometric with the one saved in the database,
(7) obtain the matching score, which is sent to the application that is found within the production, and
(8) record the audit training process concerning the entire authentication system.

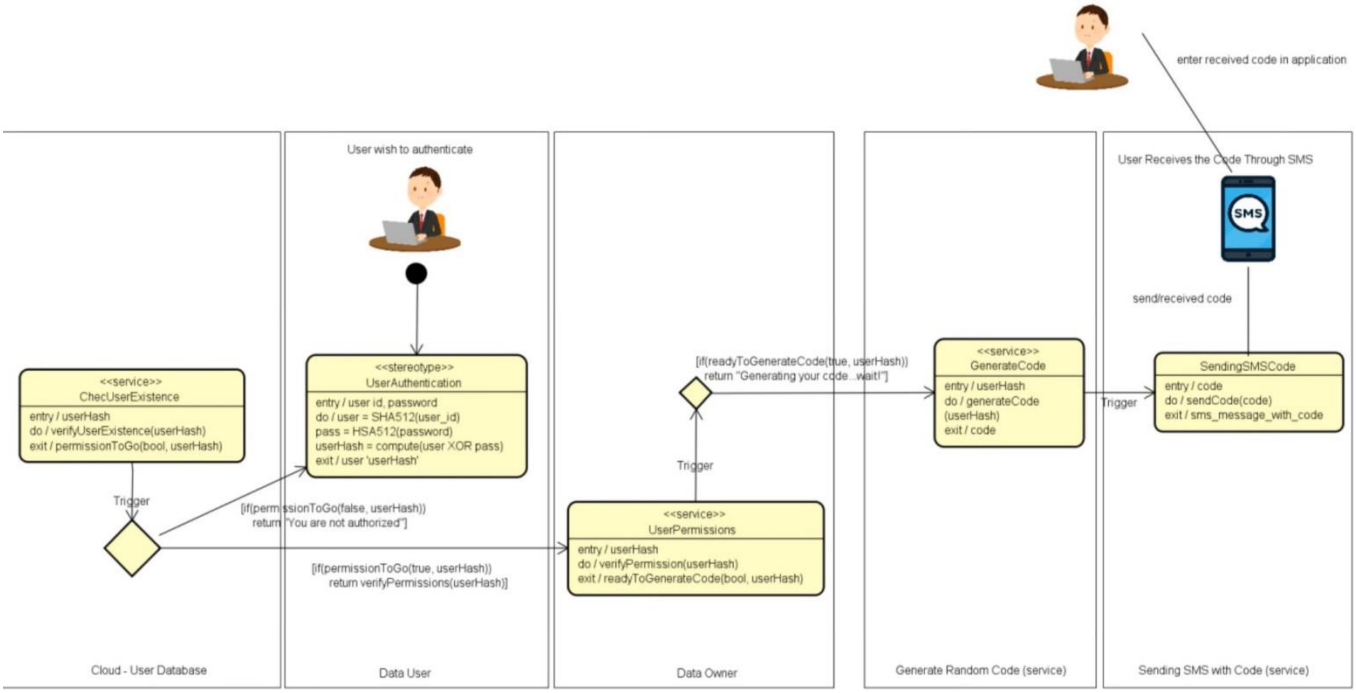


In addition to the enrolment process of known persons around us, who have served as users for testing purposes, we extended the support of biometric authentication for different types of databases, such as IEEE Biometric Databases

and Biometric Dataset Collections. This can be very helpful for offering to our system a complex database through which the queries are performed, and different metrics are measured, such as processing time.

Two-Factor Authentication and Multi-Factor Authentication :

Two-factor authentication (2FA) and multi-factor authentication (MFA) represent an authentication mechanism in which a user is asked to perform the second method of authentication once an initial login based on a username and password has been performed. In many of the applications (web software applications, mobile applications, or applicable applications for different fields), 2FA has become critical and vital in the fight against hacking. Many accounts are hacked every day and exposed on the Dark Web. The main idea of our 2FA approach is described . Our approach is based on something that the user knows (username and password), something that the user has (smartphone for the code received through an SMS on the smartphone), and something that the user is (biometric characteristic, e.g., fingerprint, facial recognition). If one looks at the BAM (Biometric Authentication Module) , in a quick search on the Internet one finds a wide variety of available tools. It is important to examine their features and to identify their advantages and disadvantages. Examining security flows represents another criterion that should be used when choosing such a tool (open source or not). Many guidelines and exploits/ vulnerabilities can be identified using this platform, and for some of them, many patches have been uploaded as a solution for fixing them.



Testing a 2FA scheme is not a trivial procedure. A 2FA scheme is characterized by two important components that cannot be controlled easily or commonly (already known by different tools or procedures), such as randomness algorithm/method and the device/ service/server (used to generate the codes/tokens) or an entire infrastruc- ture of servers with different services. The design process of the 2FA scheme has taken into consideration both, the randomness algorithm, and the infrastructure of servers, fulfilling what is necessary from the security point of view, as follows:

• Randomness: Through the generation of a unique code received on a user's smart- phone, we make the permission requirement hard to predict, it is a vital part of the testing process. We started from the idea of predicting the output in such a way as to be able to check our expectations with real output.

• The infrastructure of servers with/or services—If we are using external tools or services, such as the 2FA service from Google (Mountain View, CA, USA) or Microsoft (Redmond, WA, USA) or external devices, it means that we need to pay attention to security because we are dealing with something that is out of our

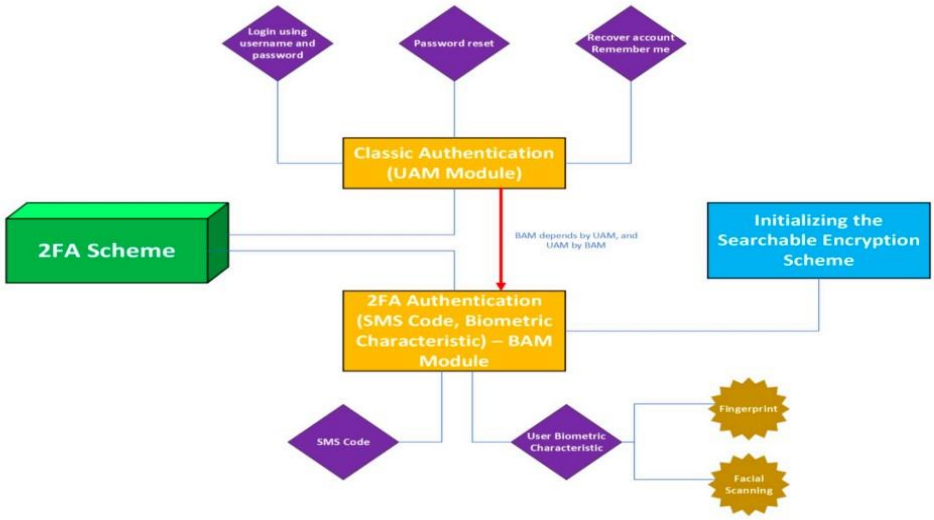
control. Performing tests against such infrastructure or devices can have high costs or testing may not be available within your environment, especially due to the continuous process that is represented by an SMS message or communication subscriber. The BAM model, which is composed of five modules, such as:

• Cloud—User database. The module is responsible for the interaction of the user with the database that holds information about the user and much more. The database is stored in cloud computing. For this scenario, we chose the Microsoft Azure platform (Redmond, WA, USA). In Microsoft Azure, we deployed the CheckUserExistence service, which runs continuously and checks each user who wishes to authenticate.

• Data user. The module is in charge of the authentication process and service for verifying the user identity. Data owner. This module verifies for each user all the permissions that are available for the user and decides terms of acceptance.

• Generate random code (service). Based on the user identity, the service computes a hash value of the user identity, and using the hash value, a code is generated and passed further to the Sending SMS with the code module.

• Sending SMS with code (service). This module sends a generated code to the user's smartphone to be used within the authentication process.



The research regarding authentication types provides very wellorganized surveys in which different approaches for biometric authentication are presented, as well as presenting the challenges raised by this authentication technology . Additional interesting papers that present authentication based on multiple factors. Cloud computing and big data opened the way for Internet-ofThings (IoT) devices. To secure the connection of systems within, often an IoT device receives a virtual identity, such that it is recognized and authenticated within the system to which it belongs. Such a type of authentication is presented, presents a study on using blockchain authentication in variable message format (VMF) as a military standard. Another method of authentication is based on blockchain technology, and it can be applied to authenticate users in smart grid infrastructures. Other interesting approaches for authentication are presented, in which the authentication is based on information about body composition analysis; presents a method of authentication based on eye color and facial recognition; presents an authentication method based on hand gestures captured by surface electromyography.

For reflecting on the progress of our current research, we took into consideration the recent advances that have been accomplished in attribute-based encryption with keyword search and attribute-based encryption with privacy protection and accountability. In attribute-based encryption with keyword search—using keypolicy attribute-based cryptography to generate different types of trapdoors that support different types of gates (e.g., AND, OR, and threshold)—represents one of the most novels approaches in designing searchable encryption with finedgrained access control.

In attribute-based encryption with privacy protection and accountability, most of the contri- butions are accomplished in the direction of ciphertext-policy attribute-based encryption (CP-ABE) mechanisms that provide the possibility of enabling fine-grained access control for encryption within data related to Internet-of-Things (IoT) data and the cloud. CP- ABE is one of the most challenging and shows great potential in providing flexible and fine-grained access control. This mechanism is much more suited for securing cloud applications, illustrating the complex authentication mechanisms, such as the one proposed in the current manuscript. Regarding searchable encryption, recent studies have a focus on dynamic schemes that allow operations of insert, update, or delete to be carried out directly on the encrypted data. Such studies in this direction. In addition, searchable encryption is used for biometric authentication—namely, the operations within the authentication process that are made directly over encrypted biometric data, without a need for decryption while applying the computations. Examples of such studies are presented. Additionally, searchable encryption can be applied in IoT or edge cloud environments or blockchain.

THE PROPOSED SCHEME:

The first draft of the current idea behind the proposed scheme was designed using a distributed infrastructure, and once the results were positive and we saw

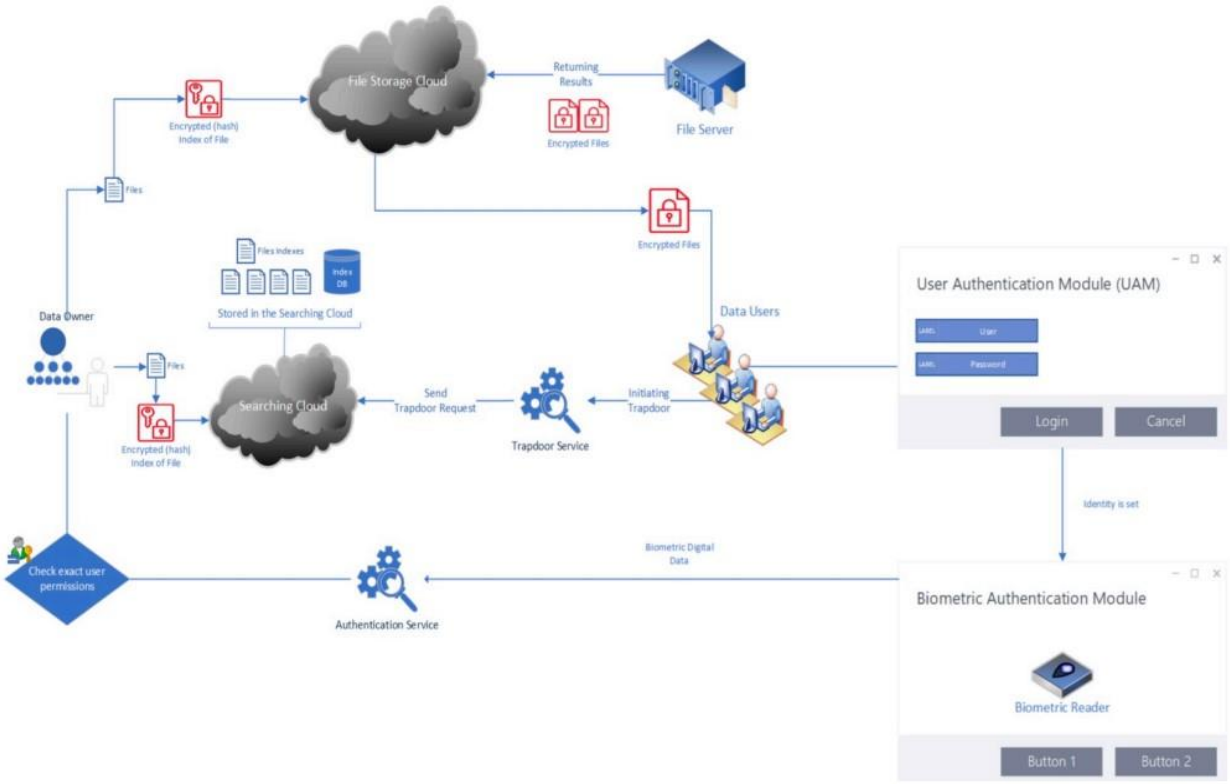
that its applicabil- ity could be extended properly to complex architectures, we started to adjust it to cloud computing and big data environments Components (module) 1 and 2 can be seen as a two-factor authentication scheme.

The proposed scheme has three important components (or modules) that are connected and cannot function if one is not fulfilled properly. The components are:

1. Searchable Encryption Scheme (SES). Based on UAM and BAM, the searchable encryp- tion is initialized and ready to perform the searching process for the keywords entered by the users. Based on those keywords, a set of documents is returned to the user.

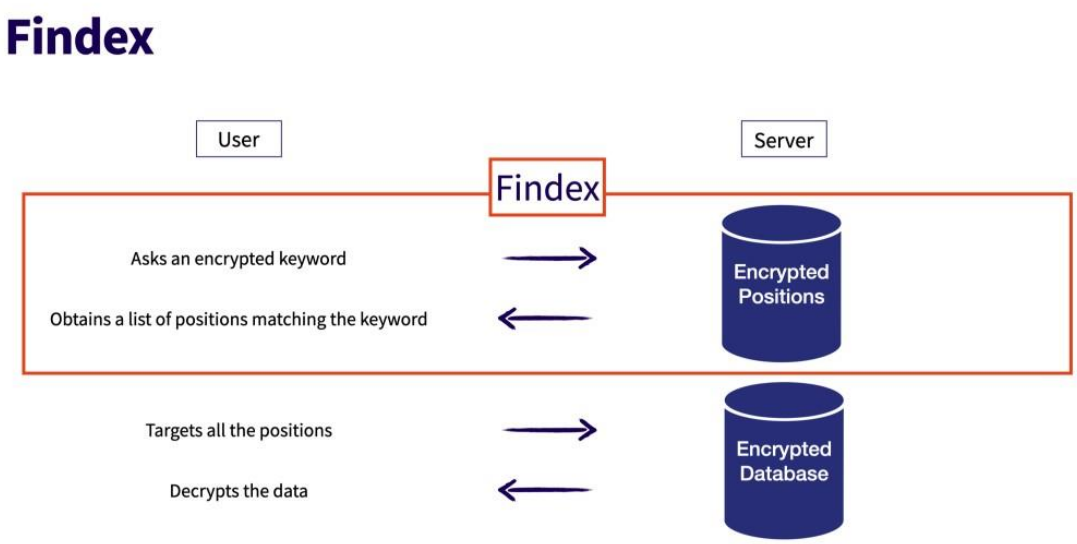
2. User Authentication Module (UAM). Its main purpose is to identify the user that wants to authenticate with the system in a cloud environment.

3. Biometric Authentication Module (BAM). This module represents an extra security measure, and it intends to certify based on one or more biometric characteristics that verify a user's proper identity.



3. FINDEX ENCRYPTEDSEARCH API:

Cosmian Findex is a Searchable Encryption scheme that allows the building of encrypted indexes. One can efficiently search these encrypted indexes using encrypted queries and receive encrypted responses. Since the environment cannot learn anything about the content of the index, the queries, or the responses, one can use Zero-Trust environments, such as the public cloud, to store the indexes.



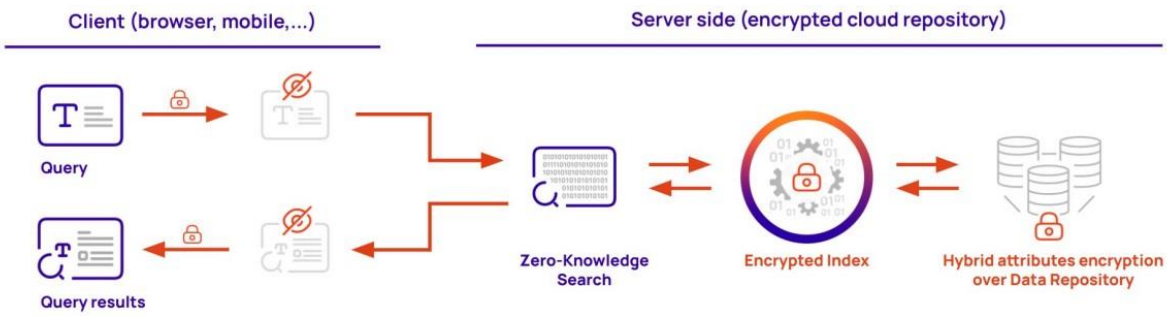
Cloudproof Encryption is the combination of two cryptographic schemes:

- Covercrypt that provides application level encryption
- Findex that provides encrypted search In addition to encrypting the data given access policies, Cosmian Cloudproof Encryption libraries offer the ability to create encrypted indexes and securely search for encrypted data. These indexes match an encrypted keyword to an encrypted location (a database UID, an URL…). The encrypted search can handle exact match, starts-with, ends-with and synonyms.

Encrypted indexes have the following characteristics:

- all data in the indexes are encrypted
- queries to the index are encrypted
- answers to queries are encrypted

Since the server never learns anything about the content, the queries or the answers, so the index can be safely stored in a zero trust environment, next to the indexed data. Typically, these indexes are stored in fast scalable key-value stores deployed in the cloud.



Searching the encrypted index:

The typical user search workflow is as follows:

1. Using a client library (java, python, js, ios, android, etc.), the user supplies a searched keyword and encrypts it using a secret key.
2. The client library issues two encrypted requests to the keyvalue store storing the index (the key-value store can be any store of your choice - see below)
3. The client library decrypts the response, typically the locations in the main DB where the keyword is found. The user will then query the main database for the given records, encrypted with application level encryption, and decrypt what their private key allows.

Updating and compacting the index:

In addition to the search functionality, Findex proposes an upsert (i.e. update or insert) function and an index compacting function. The upsert function allows concurrent updating of the index while in use. Update conflicts are resolved using a mechanism based on retries. See the callback to upsert Table Entry lines for details. The Sqlite and Redis implementations available in the various languages provide implementation examples. The index compacting function provides two essential functionalities:

- it reduces the index size by removing entries pointing to locations that no longer exist in the main database.
- it rotates ciphertexts, and thus stops potential statistical attacks by an attacker listening to encrypted queries on the encrypted indexes.

Cryptographic Systems:

Cloudproof Encryption is fundamentally the combination of 2 schemes: one for encryption using attributes (a hybrid KP-ABE + AES scheme) and one for quick, secure searches (an SSE scheme). Cosmian uses a hybrid encryption scheme (KEM DEM): symmetric encryption of the clear text with a randomly generated ephemeral symmetric key, itself encrypted using a public key encryption scheme.

Symmetric key encryption: AES-256 GCM:

When encrypting an arbitrary-sized clear text, a random 256bit ephemeral key is first generated using a Cryptographically Secure Pseudo-Random Number Generator (CS-PRNG). This key is used to encrypt the data using AES-256 GCM. Whenever possible, the library will attempt to use the AES native CPU instructions when they are available for speed. Cosmian uses the Rust implementation of AES-256 GCM. The result of this first encryption is a ciphertext starting with a 96bit Nonce, followed by the data encrypted in counter mode, which length is equal to that of the clear text and ending with a 128bit authentication tag.

5.CONCLUSION :

In the rapidly evolving digital landscape, the importance of data security cannot be overstated. As we increasingly rely on cloud services to store and manage our data, the need for robust security measures, such as searchable encryption, becomes paramount. Searchable encryption stands at the intersection of data accessibility and confidentiality, offering a promising solution to the paradox of maintaining data privacy while ensuring efficient retrieval. It is a critical component in the broader context of cloud security, providing a shield against potential data breaches and unauthorised access. However, the journey

towards perfecting searchable encryption is fraught with challenges. The current techniques, while effective, are not without their limitations. The trade-off between data confidentiality and performance is a significant hurdle that needs to be addressed. Despite these challenges, the future of searchable encryption is bright. With ongoing research and development, we can anticipate the emergence of more efficient and secure techniques. These advancements will undoubtedly bolster cloud security, paving the way for a safer and more secure digital future. As we embrace the digital age, the importance of searchable encryption in protecting our data becomes clearer. It goes beyond academic discussion to become a crucial part of our digital setup. Though the road ahead may be complex, the potential benefits of having secure and accessible data highlight the importance of this effort