Computational Sciences and Engineering 4(1) (2024) 79-100

University of Guilan

Computational Sciences and Engineering

journal homepage: https://cse.guilan.ac.ir/

Validation of Returned Results in Asymmetric Searchable Encryption Schemes

Arian Arabnouri [a], Reza Ebrahimi Atani [a,*], Shiva Azizzadeh [a]

[a] Department of Computer Engineering, University of Guilan, Rasht, Iran

ARTICLE INFO

Article history:
Received 21 February 2025
Received in revised form 14 April 2025
Accepted 24 April 2025
Available online 26 April 2025

Keywords:
Searchable encryption
Confidentiality
Data integrity
Bilinear pairing
Unreliable server
dPEKS

ABSTRACT

As organizations increasingly outsource data to cloud storage, ensuring the security and integrity of this data becomes paramount. Searchable encryption (SE) offers a promising solution by enabling secure searches on encrypted data, thereby preserving privacy and confidentiality. However, existing SE schemes often overlook the issue of data integrity, particularly when the cloud server, an external and potentially untrusted entity, returns invalid or malicious results. This paper proposes a novel technique for validating the results returned by the cloud server in asymmetric searchable encryption schemes. The proposed method introduces minimal efficiency overhead and is easily applicable to existing schemes. By applying this technique to the dPEKS (designated Public Key Encryption with Keyword Search) scheme, we demonstrate a significant reduction in search time while enhancing the ability to validate results across multiple servers. Our approach ensures that the integrity of returned data is preserved, even in scenarios where the cloud server may act maliciously. The proposed technique is particularly effective in private scenarios, such as e-care and banking, where only authorized users can send and retrieve data. This work contributes to the ongoing effort to improve the security and reliability of searchable encryption in cloud environments.

1. Introduction

Secure and rapid access to data saved on cloud storage zones becomes increasingly essential in modern lives. Cloud services and infrastructures have caused a novel development in services provided within E-government and E-commerce, since they have high efficiency, low cost and rapid accessibility. Nevertheless, issues related to the security of data are among the most significant challenges that cloud zone confronts. Data security includes confidentiality, integrity, and availability. Utilizing cloud infrastructures improves data availability, since the data on cloud can

* Corresponding author.
E-mail addresses: rebrahimi@guilan.ac.ir (R. Ebrahimi Atani)

https://doi.org/10.22124/cse.2025.29912.1095

be accessed anytime and anywhere. However, considering unreliable characteristic of cloud, two other mentioned security features might be highly compromised. Cryptography is the basic mechanism used to provide confidentiality, although it has some costs. As an example, there is no search capability among encrypted context. Nevertheless, searching is one of the most essential functions being performed on cloud server, since there is a huge amount of data on this server. A practical method to preserve security of the data while searching among them is searchable encryption. Through this method, plain text is encrypted in such a manner that searching can be done among it while its security is preserved. Utilizing searchable encryption in an unsecure area such as cloud zone can improve security a lot while causing minimum decrement in efficiency. This is because it enables searching on encrypted text while preserving security of user. Providing data confidentiality, privacy preserving for users, high efficiency, proper security and ease of implementation, makes the searchable encryption method a leading one. Similarly to traditional cryptography, searchable encryption has two kinds which are symmetric and asymmetric ones. Searchable encryption is accomplished through four steps i.e. setup and key generation, encryption and generating indices, generating trapdoor, and testing. First of all, variables and functions needed within other scheme are produced, and key pairs of entities are generated as well. In the second step, keywords of intended document are extracted, the document is encrypted, and an index will be generated per each keyword. This index is sort of keyword encryption using receiver's public key (in asymmetric schemes) or private key (in symmetric schemes). Generally, a random output is produced through this step in asymmetric schemes. An authorized user who intends to search, attempts to produce a trapdoor using his intended keyword within the third step. Producing the trapdoor will be accomplished using receiver's private key. Trapdoor generator function receives the keyword and receiver's private key (in asymmetric schemes) or private key (in symmetric schemes) and produces an output related to the generated index of the same keyword. Finally, testing will be done after receiving the trapdoor by the server. The server verifies matching between received trapdoor and existing indices through this step, and returns documents which have indices matched with the trapdoor to the receiver. Returning the documents can be accomplished through one step (sending them), or two steps i.e. returning ID of the document to the receiver by the server, and sending intended IDs to the server by the receiver subsequently [1,2].

The other required security feature is data integrity. Users need to ensure the integrity of their data. Generally, two entities may be able to compromise data integrity i.e. cloud server and an external intruder. Injection attacks by which the external attacker is able to threaten data integrity are studied in [3, 27, 28], and a prevention method is proposed in the same study. The objective of this article is to ensure that cloud server does not violate the integrity of data. Hence, a scheme is proposed to validate results returned from the cloud server, which can be implemented within two scenarios. The first one is used for general applications in which everyone is able to send data to their intended destination. It is possible for server to cheat in these scenarios due to some existing constraints. Therefore, a second kind of application is proposed which can be used in private scenarios in which there are certain users. E-care [29] and bank scenarios can be named as instances of this kind of applications [30, 31, 32, 33]. The server is not able to cheat in these scenarios, although some of the documents matched with the trapdoor may not be returned by the server. The major objective of this article is to propose a scheme that validates the results returned from server side while preserving the data integrity against server. Accordingly, the rest of article is organized as follows: In Section 2, related works about proposed article are explained. Required preliminaries are reviewed in section 3. Architecture of our proposed scheme is explained in section 4. The proposed scheme is described

in section 5, and section 6 contains the analysis of it. Finally, the conclusion of the article is presented in section 7.

2. Related Works

Song et al. [4] proposed the first practical scheme for searchable encryption. This scheme belongs to the symmetric class of searchable encryption and it uses one private key during both encryption and decryption phases. In other words, this scheme is classified among single writer/reader schemes. No index was utilized within this scheme, and searching time for each document was related to the length of the document consequently. However, this scheme is not efficient when there is high amount of documents. Other efforts in this context have been made in [5-8] in which some concepts such as index, inverted index, conjunction search, and performing search via small computational capacity devices are introduced. Additionally, other efforts have been made in [9,10,] to verify returned results, and some others to improve performance and security [11,12,].

Boneh et al. [13] proposed the first asymmetric searchable encryption scheme, which was named public key encryption with keyword search scheme, and used identity-based encryption (IBE) in order to implement searchable encryption in an asymmetric manner [34]. Public key encryption with keyword search schemes are useful in scenarios in which multiple data owners attempt to send data for a certain receiver. Nevertheless, scheme proposed in [13] was vulnerable to keyword guessing attacks, since it was possible to produce the index and test the trapdoor using any generated index. The attacker generates intended index for any words in dictionary, and tests them against the monitored trapdoor. In the case of TRUE to be the result of test, the word used for generating trapdoor is the same word selected by the attacker. PERKS scheme [14] was proposed to solve this issue, in which the data owner sends his intended keyword for generating the index to the receiver. Receiving this word, the receiver first concatenate it with a private string and hash the new string. Subsequently, the hashed value will be sent to the data owner, and he use it for generating the trapdoor. This scheme does not allow everyone to produce indices. Nevertheless, there were fundamental problems with this scheme such as requiring receiver to be permanently online. Furthermore, in the case of using a shared private string for all of the data owners, it was still possible for them to perform a keyword guessing attack.

dPEKS [15] scheme was proposed as another approach to solve this problem, in which only server is able to perform testing. This scheme prevents offline keyword guessing attack by limiting searching to the server. This is because the Test function can be operated only by the server, and the server, consequently, is the only entity which can verify matching between trapdoor and index. Hence, it is not possible for an outer intruder to match the eavesdropped trapdoor with the index produced based on the keyword. The underlying idea of this scheme is to encrypt indices and trapdoors using server's public key in a way that server's private key becomes indispensable for operating the Test function and comparing trapdoor with index. The algorithm for the dPEKS scheme is as follows:

Initial agreement and key generation: The first step is picking two multiplicative cyclic groups of degree P, named G_1 and G_2. Following this, a generator named g is picked among G_1, along with two random members u_1 and u_2. Furthermore, $H_1: \{0,1\}^* \rightarrow G_1$, $H_2: \{0,1\}^* \rightarrow G_1$, and $H_3: G_2 \rightarrow \{0,1\}^{\lambda}$ are chosen as hash functions, and three random numbers are picked among Z_p, named a, b,

and c. Finally, private keys of the server and receiver is calculated as $PRIV_{SERV} = a$ and $PRIV_{REC} = b$, and public ones will be $PUB_{REC} = (g^b , u_2^{\frac{1}{b}})$ and $PUB_{SERV} = (g^a , u_1^{\frac{1}{a}})$ respectively.

Index generation: A random number r is picked among Z_p per each word w_i among the message M, and the index will be generated as $I = (I_1 , I_2) = ((PUB_{REC})^r , H_3\big(e(PUB_{SERV} , H_2(w_i)^r)\big))$. Afterwards, all of the encrypted indices will be sent to the server along with the encrypted document.

Trapdoor generation: A random number r' is picked among Z_p, and the trapdoor will be generated as $T_w = [T_1 , T_2] = [g^{r'} , H_2(w)^{\frac{1}{PRIV_{REC}}} . H_1({PUB_{SERV}}^{r'})]$ and will be sent to the server subsequently.

Test: The value of $t = \frac{\mathrm{T_2}}{\mathrm{H_1({T_1}^{PRIV_{SERV}})}}$ is calculated at first. Thereupon, if the equation $I_2 == \mathrm{H_2}(e(I_1 , t^{PRIV_{SERV}}))$ was true, it can be noticed that the searched word is the same as the i-th word, which the server will return it to the user consequently.

As mentioned above, this scheme is secure against offline keyword guessing attacks but it is still vulnerable to online ones. In 2013, an attack was performed against this scheme, which was named online keyword guessing attack, since the attacker had interaction with the server [16]. In 2015, Chen [17] proposed a scheme in order to secure dPEKS scheme against online attacks, although it was still vulnerable against cloud server [32]. Whereas, some schemes were proposed to prevent this attack [18-21]. Proposed plans in [18,19] are some instances that are applicable to dPEKS, and despite their constraints, they can protect against internal keyword guessing attack.

Another method was proposed in [20] based on using data owner's key such as work done in [19]. The SAE-I scheme which is shown in ***Figure 1*** is an authenticated searchable encryption scheme, in which the data owner's private key is utilized besides the receiver's public one in order to generate index. Moreover, generating trapdoor requires the receiver's private key besides the data owner's public one. Hence, before generating trapdoor, receiver has to determine the user from whom received documents must be searched among. This is because the trapdoor can be matched only with indices which comprise the intended word and are generated by the user whose public key is utilized in generating the trapdoor. Additionally, the keyword guessing attack is not possible to perform, since the adversary is not able to generate an index for his intended word in a way that it matches eavesdropped trapdoor generated by someone else. The algorithm for the SAE-I scheme is as follows:

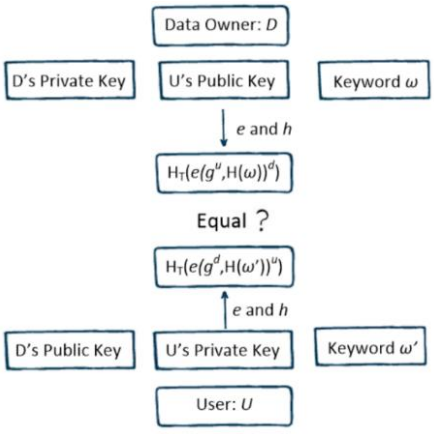

Figure 1: The SAE-I scheme.

Initial agreement and key generation: The first step is picking two multiplicative cyclic groups of degree P, named G_1 and G_2, along with a bilinear map function between these two groups as $e: G_1 \times G_1 \rightarrow G_2$. Following this, a generator named g is picked among G_1, along with two hash functions as $H_1: G_2 \rightarrow \{0,1\}^*$ and $H_2: \{0,1\}^* \rightarrow G_1$. Generating each user's key, a random number is picked among Z_p, named a. Subsequently, utilizing these random numbers, the key pair for user (u) is calculated as $KeyPair_u = (PRIV_u, PUB_u) = (a, g^a)$. Finally, each user's private key ($PRIV_u$) will be sent exclusively to himself, by which he will be able to generate index and trapdoor besides decrypting his documents. Moreover, user's public key (PUB_u) will be sent to all other users, by which they will be able to encrypt documents which they intend to send to the user (u), besides generating their indices. This public key will also be used in order to generate trapdoor for performing search among received documents from the user (u).

Index generation: In this scheme, the index will be generated as $I_w = H_1(e(PUB_{REC}, H_2(w))^{PRIV_{DO}})$.

Trapdoor generation: In this scheme, the trapdoor will be generated as $T_w = H_1(e(PUB_{DO}, H_2(w))^{PRIV_{REC}})$.

Test: Whenever an index and a trapdoor are both generated for a single word, and both data owner and receiver are same users, the index and the trapdoor will have their same values. Hence, verifying the matching between trapdoor and index only requires verifying the equation$I_w == T_w$. Moreover, achieving high efficiency in performing this function only requires storing the indices in a hash table and performing a search for the trapdoor among this hash table. These processes can be operated in *O(1)* order of time, which makes this function a very high efficiency one.

It is important to mention that in SAE-I scheme it is required to build a trapdoor for each data owner, and this issue is counted as a constraint for mentioned scheme. However, it is still efficient due to its proper structures. To the best of the authors' knowledge there is not an asymmetric searchable encryption scheme with verification capability. Nevertheless, the scheme proposed in [22] adds verification capability to the KP-ABE scheme that is an attribute-based asymmetric searchable encryption scheme. Schemes proposed in [23-26] were presented to increase capabilities of asymmetric searchable encryption as well.

3. Preliminaries

In this section, required background is briefly reviewed.

3.1. Discrete Logarithmic Problem and Diffie-Hellman Assumption

Discrete logarithm problem is the solution x for the equation $h = g^x$ over a finite cyclic group. Within this problem, g and h are elements of a multiplicative cyclic group of degree p and x is an element of $Z_p{}^*$ group. There is no efficient algorithm to solve the discrete logarithm problem as for now. Therefore, this problem gets a vast attention in asymmetric cryptography.

Diffie-Hellman's assumption is a mostly used assumption in asymmetric cryptography that is proposed based on the same problem. It expresses that it is not possible to calculate g^{ab} during a polynomial time, knowing only $< g, g^a, g^b >$ without a or b. This assumption is widely used in asymmetric cryptography. Key exchange, digital signature, searchable encryption, IBE and IBS can

be named as a number of its applications. Several assumptions are proposed based on this assumption and one of them is represented as follows:

DBDH assumption: Given $G_1$and G_2 to be two groups with prime order q, g to be the generator of G_1, and e: $G_1 \times G_1 \rightarrow G_2$ to be a mapping, and a , b and c to be three random numbers, there is no efficient algorithm to calculate $e(g , g)^{abc}$ having (g , g^a , g^b , g^c) . Diffie-Hellman assumption is used within dPEKS scheme in order to secure the trapdoor and to keep randomness of it against the external attacker. In our proposed scheme, this assumption is used in order to exchange private key between cloud storage and authentication server. Moreover, the generalized version of DBDH assumption is used to exchange keys between several servers.

3.2. Bilinear Pairings

Bilinear pairings is another function that is mostly used in asymmetric cryptography. These functions play a vital role in many Diffie-Hellman based schemes and attacks against them as well. Symmetric and asymmetric are two types of bilinear pairing. The symmetric bilinear pairing is used within the proposed scheme, which is defined as follows:

The e: $G_1 \times G_1 \rightarrow G_2$ mapping is a bilinear pairing while $(G_1,.)$ and $(G_2,.)$ are cyclic groups with prime order q. This mapping has the following characteristics:

1) Bilinearity: $\forall\, a , b \in Z_q$, $\forall\, g_1 , g_2 \in G_1$: e($g_1{}^a , g_2{}^b$) = $e(g_1 , g_2)^{a.b}$

2) Non-degeneracy: $\forall\, g \in$ Generator of (G_1), e(g , g) $\in$ Generator of (G_2).

3) Computability: There is an algorithm to effectively compute e(g_1 , g_2) .

4. Architecture Of Proposed Scheme

An architecture including four components is used within our proposed scheme. These components are the data owner, the receiver, the cloud server, and the authentication server. Building a trapdoor, the receiver will be able to perform a search within documents and obtain his desired ones. Storing the data and delivering it to the receiver are the duties assigned to the storage server. Furthermore, the storage server is able to utilize a trapdoor received from the receiver in order to perform a search, without learning about content of the document or the word that trapdoor is created with. This server can be provided by a third party, and it is assumed not to be honest. Hence, returned documents from this server may be malicious and cannot be trusted. The authentication server is obliged to verify the documents returned from the storage server. There are two modes for the proposed scheme. The first one is general mode in which everyone is able to send data to others. Contrary to the first mode, it is not possible for everyone to send data to others within the second mode, and only specific users are able to perform this action. ***Figure 2*** and ***Figure 3*** demonstrates the first and the second modes respectively.

The data owner creates the index using public key of the server and receiver's one, and sends the document and indices set to the server. In the case of feeling necessity (tendency) for restriction, the data owner signs produced index using his private key. Subsequently, he sends the indices set along with their signatures, encrypted context and his ID to the server, and the server will store these data.

Now the receiver generates the trapdoor using his private key and the public key of the server, then he sends the produced trapdoor to the server.

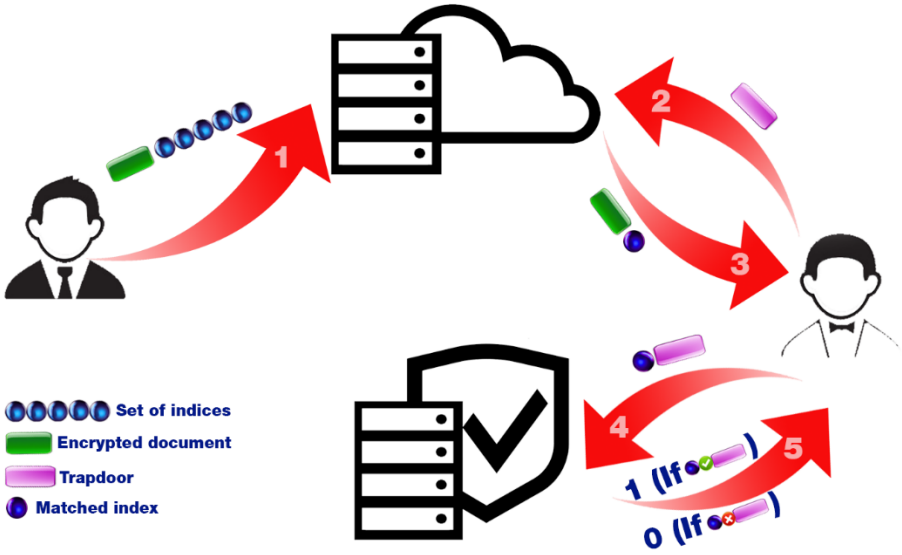

Figure 2. Proposed architecture for general scenarios

The server gets the trapdoor and performs a test to determine whether the trapdoor matches the index or not. In the case they match, the server sends the document to the receiver along with user's ID, matched index, and its signature in case of existence. The receiver sends his trapdoor along with the returned index and its signature to the authentication server in order to ensure validity of the test. The authentication server validates the index using the signature, and in the case of validity it checks whether the index and the trapdoor match. It returns number 1 to the receiver if they matched, otherwise it returns number 0. The GM scheme can be utilized for privacy preserving, and a digital signature can be used to perform the validation and provide validity.

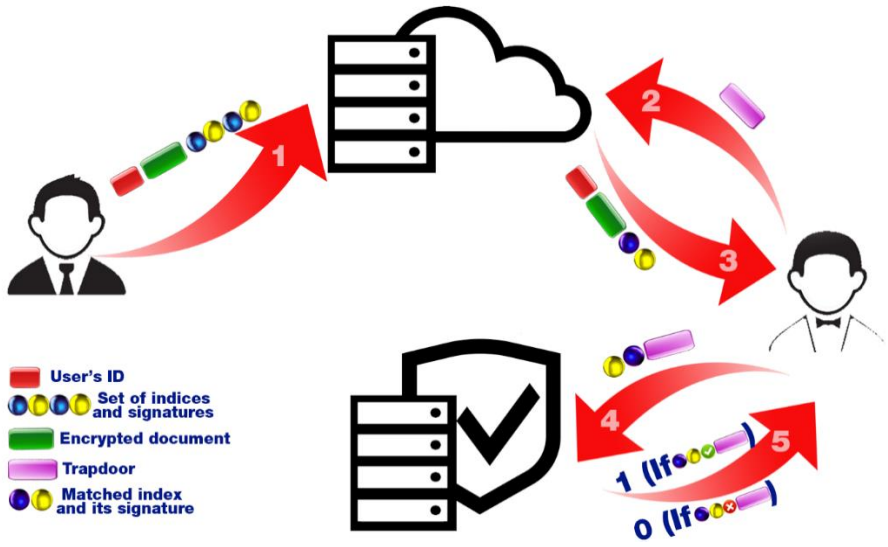

Figure 3. Proposed architecture for private scenarios

5. The Proposed Schemes

The cloud server is mostly considered to be honest but curious within studies done among searchable encryption field. Nevertheless, some scenarios may occur in real world that violate the honesty of

server. Hence, a mechanism is required to validate the honesty of server. Providing such a mechanism is the main objective of our proposed scheme. As mentioned previously, two situations may occur within this scheme. When system has certain limited users, validation will be more effective, since server can utilize the indices generated by other users for its document. The same situation occurs within private schemes such as [19] as well.

The underlying idea of the proposed scheme is granting test capability to the trusted third party in dPEKS scheme. In this scheme, only a single cloud server has the test capability. Therefore, server can send a document to receiver, even if the trapdoor does not match any of the indices of the document. This restriction is performed by adding a key pair to the cloud server in order to prevent online keyword guessing attack. Since the test function requires the private key of server, only the server is able to perform searching. Diffie-Hellman key exchange method is used within the proposed scheme in order to produce a shared private key between the cloud server and trusted third party (the authentication server). Subsequently, a public key can be generated for this shared key, and the current key pair can be used to perform testing. The cloud server and the authentication server both have the private key, so they both can perform testing. Hence, when a document does not contain any index matched with the trapdoor produced by the receiver, the server cannot return it to him. However, the server still can add an index matching the trapdoor to the indices set, and then return it. Preventing this issue, the index will be concated with the document ID and the result value will be signed subsequently within the proposed scheme. This signature cannot be forged by another entity. On the other hand, existence of a signature is required for each index. Therefore, the server cannot add its intended index to the indices set. According to existence of document ID, the produced signature is assigned to the document, and the signature of the index within document A cannot be used for document B.

Nevertheless, using digital signature in this scheme makes it impracticable in scenarios that data owner is not known formerly, since validating the signature requires the data owner's public key. Hence, the server has to interact with the data owner earlier and obtain his public key. The proposed scheme will be explained based on dPEKS and SAE-I schemes as follows.

5.1. Proposed scheme merged with dPEKS

Our proposed scheme is merged with public key encryption scheme with a designated tester within this section. It includes eight algorithms that will be explained as follows.

GlobalSetup(λ): Variables and functions needed within the steps ahead are generated in this step. Since the proposed scheme is an add-on for dPEKS scheme, this function calls the dPEKS.GlobalSetup(λ) method. Furthermore, the generator g_p will be chosen from Z_{p}. In the case of requiring users to be restricted, a MAC function or a digital signature will be picked.

$\mathbf{KeyGen_{REC}(gp)}$: This algorithm calculates the key pair for the receiver. Everyone will obtain the receiver's public key, and the data owner can encrypt the documents and produce the index using the receiver's public key. Furthermore, user needs the private key to perform searching on his documents and decrypt them as well. In the proposed scheme, the algorithm used for producing the receiver's key pair is similar to the one used in dPEKS scheme. ($PRIV_{REC} = dPEKS.KeyGen_{REC}(gp).PRIV_{REC}$ and $PUB_{REC} = dPEKS.KeyGen_{REC}(gp).PUB_{REC}$).

$\mathbf{KeyGen_{AUTH-SERV}(gp)}$: In the proposed scheme, a fourth entity, which must be trusted, is used in order to validate returned results from server. Since this entity does not exist in dPEKS scheme, there is no key generator algorithm for it as well. Therefore, this algorithm is included in our proposed scheme. A private key is required for the tester through testing phase. Consequently, the key pair produced for the authentication server will be used in order to exchange the key with the cloud server and to obtain this shared key in order to perform searching. Furthermore, the validation result of the returned result from the server will be checked by the authentication server, and the result will be signed using this key pair in order to prevent it from being forged.

A random number b is picked among Z_{p} and will be considered as the data owner's private key ($PRIV_{\mathrm{AUTH-SERV}} = a$). The public key will be calculated as $PUB_{\mathrm{AUTH-SERV}} = {g_p}^a$ subsequently.

$\mathbf{KeyGen_{SERV}(gp)}$: The key generator algorithm for the storage server in our proposed scheme is similar to the one included in dPEKS, except for some changes. A random number c is picked from Z_{p} and the shared key between the storage server and the receiver will be calculated as $PRIV_{SERV} = (PRIV_{SERV1}, PRIV_{SERV2}) = (\mathrm{c} = dPEKS.KeyGen_{SERV}(gp).PRIV_{SERV}, {PUB_{\mathrm{AUTH-SERV}}}^c)$. Then the public key of the server will be calculated as $PUB_{SERV} = (PUB_{SERV1}, PUB_{SERV2}) = ({g_p}^c, dPEKS.KeyGen_{SERV}(gp).PUB_{SERV})$.

$\mathbf{KeyGen_{DO}(gp)}$: In the case of requiring users to be restricted, a public/private key pair should be generated for the data owner as well. This key pair will be used within producing the signature for each index. This occurs when only certain authorized users must be able to produce the index. This digital signature can guarantee the server that the index is produced for the intended document. The public/private key pair is picked according to the used digital signature scheme.

$\mathbf{dPEKS(w}, \boldsymbol{ID_{doc}}, \mathbf{PUB_{SERV}}, \mathbf{PRIV_{DO}}, \mathbf{PUB_{REC}}, \mathbf{gp)}$: The DO produces an index for a keyword in order to enable searching capability for it. He accomplishes this action for every keyword in the document. Hence, the intended index is produced calling $I = dPEKS(gp, PUB_{REC}, PUB_{SERV1}, w)$ function. Then if it is required for users to be restricted, the produced index will be merged with the document ID and signed using the data owner's private key ($sign_{\boldsymbol{PRIV_{DO}}}(I \,||\, ID_{doc})$). A MAC function can also be used instead of the digital signature, since the signature will be only verified on the authentication server side ($F(K, I \,||\, ID_{doc})$). The value K used in this function is a shared key between the data owner and the server, which can be calculated through Diffie-Hellman assumption. In the case of restricting users (using digital signature or MAC function), it is required to send the data owner's ID to the cloud server along with the document and the indices set and the signature.

$\boldsymbol{dTrapdoor(gp, PUB_{SERV}, PRIV_{REC}, W)}$: A trapdoor should be produced in order to perform searching for the intended keyword among the document. Hence, $dTrapdoor(gp, PUB_{SERV1}, PRIV_{REC}, W)$ function will be called.

$\mathbf{dTest(gp, C, }\boldsymbol{PRIV_{SERV}}\mathbf{, }\boldsymbol{T_w}\mathbf{)}$: Calling $dTest(gp, C, PRIV_{SERV2}, T_w)$ function, the test will be performed on cloud server side.

$\mathbf{Check(gp, Index, Signed - Index, }\boldsymbol{PRIV_{SERV}}\mathbf{, ID_{DO}, }\boldsymbol{T_w}\mathbf{)}$: An additional function is used on the authentication server side in this scheme. This function is similar to the performed test on the storage server side and verifies the integrity of this test. The authentication server first obtains the data owner's public key using his ID, and compares the index with his signature subsequently. If

these two components match, it checks the index and the trapdoor. Hence, it calculates the shared key between the server and itself ($\boldsymbol{KEY_{AUTH-SERV}} = (\boldsymbol{PUB_{SERV1}})^{\boldsymbol{PRIV_{AUTH-SERV}}}$). Finally, $\boldsymbol{dTest}(\boldsymbol{gp}, \boldsymbol{C}, \boldsymbol{KEY_{AUTH-SERV}}, \boldsymbol{T_w})$ function will be called. In the case of TRUE to be the result, the index matched the trapdoor and the server accomplished the comparison correctly.

5.2. Proposed scheme based on SAE-I

This section merges our proposed scheme with public key encryption scheme with a designated tester. It includes eight algorithms which are explained as follows.

GlobalSetup(λ): Variables and functions needed within the steps ahead are produced in this step. Since the proposed scheme is an add-on for SAE-I scheme, this function calls the $SAE - I.GlobalSetup(\lambda)$method. Furthermore, the generator g_p will be chosen from Z_{p}. In the case of requiring users to be restricted, a MAC function or a digital signature will be picked.

$\mathbf{KeyGen_{REC}(gp)}$**:** Similar to the previous scheme, a public/private key pair for the receiver is required in order to produce index and trapdoor. The same algorithm used for producing the receiver's key pair in the SAE-I scheme is used in this scheme. ($PRIV_{REC} = \mathrm{SAE} - \mathrm{I}.KeyGen(REC).REC_{PRIV}$) and ($PUB_{REC} = \mathrm{SAE} - \mathrm{I}.KeyGen(REC).REC_{PUB}$).

$\mathbf{KeyGen_{AUTH-SERV}(gp)}$**:** Since the proposed scheme does not require a key to perform searching, the generated key pair will be only used to sign the result for the receiver

$\mathbf{KeyGen_{DO}(gp)}$**:** In this scheme the data owner's key pair is required to produce trapdoor and index, and the key pair is required for signing the index as well. consequently, this key pair is produced through ($PRIV_{DO} = \mathrm{SAE} - \mathrm{I}.KeyGen(DO).DO_{PRIV}$) and ($PUB_{DO} = \mathrm{SAE} - \mathrm{I}.KeyGen(DO).DO_{PUB}$).

$\mathbf{BuildIndex(w, ID_{doc}, PRIV_{DO}, PUB_{REC}, gp)}$: The trapdoor and its signature will be produced within this function through ($I = SAE - I.BuildIndex(PUB_{REC}, PRIV_{DO}, w)$) and ($sign_{\boldsymbol{PRIV_{DO}}}(I \,||\, ID_{doc})$). The MAC function ($F(K, I \,||\, ID_{doc})$) can be used as well.

$\boldsymbol{dTrapdoor}(\boldsymbol{gp}, \boldsymbol{PUB_{SERV}}, \boldsymbol{PRIV_{REC}}, \boldsymbol{W})$**:** A trapdoor is required in order to perform searching for the intended keyword among the document. Hence, $SAE - I.Trapdoor(PUB_{DO}, PRIV_{REC}, w)$ function will be called.

$\mathbf{dTest(gp, C, }\boldsymbol{PRIV_{SERV}}\mathbf{, }\boldsymbol{T_w}\mathbf{)}$**:** Calling $SAE - I.BuildIndex(I, T)$) function, the test will be performed on cloud server side.

$\mathbf{Check(gp, Index, Signed - Index, }\boldsymbol{PRIV_{SERV}}\mathbf{, ID_{DO}, }\boldsymbol{T_w}\mathbf{)}$**:** In this scheme, the authentication server checks matching between the trapdoor and the index calling $\boldsymbol{SAE - I.BuildIndex(I, T)}$), after ensuring the integrity of signature. In the case of TRUE to be the result, the index matched the trapdoor and the server accomplished the comparison correctly.

Proposed scheme merged with dPEKS

In this section, our proposed scheme is merged with public key encryption scheme with a designated tester. It includes eight algorithms which are explained as follows.

GlobalSetup(λ): Variables and functions needed within the steps ahead are produced in this step. Since the proposed scheme is an add-on for dPEKS scheme, this function calls the dPEKS.GlobalSetup(λ) method. Furthermore, the generator g_p will be picked among Z_{p}. In the case of requiring users to be restricted, a MAC function or a digital signature will be picked.

$\mathbf{KeyGen_{REC}(gp)}$: This algorithm calculates key pair for the receiver. Everyone will obtain the receiver's public key, and the data owner can encrypt documents and generate index using the receiver's public key. Furthermore, user needs the private key to perform searching on his documents and decrypt them as well. In the proposed scheme, algorithm used for producing the receiver's key pair is similar to the one used in the dPEKS scheme. ($PRIV_{REC} = dPEKS.KeyGen_{REC}(gp).PRIV_{REC}$ and $PUB_{REC} = dPEKS.KeyGen_{REC}(gp).PUB_{REC}$).

$\mathbf{KeyGen_{AUTH-SERV}(gp)}$: In the proposed scheme, a fourth entity, which must be trusted, is used in order to validate returned results from the server. Since this entity does not exist in dPEKS scheme, there is no key generator algorithm for it as well. Therefore, this algorithm is included in our proposed scheme. A private key is required for the tester through testing phase. Consequently, the key pair produced for the authentication server will be used in order to exchange key with the cloud server and to obtain this shared key in order to perform searching. Furthermore, the validation result of the search will be signed and sent to the receiver. A random number b is picked among Z_{p} and will be considered as the data owner's private key ($PRIV_{\mathrm{AUTH-SERV}} = a$). The public key will be calculated as $PUB_{\mathrm{AUTH-SERV}} = {g_p}^a$ subsequently.

$\mathbf{KeyGen_{SERV}(gp)}$: The key generator algorithm for the storage server in our proposed scheme is similar to the one included in dPEKS, except for some changes. A random number c is picked among Z_{p} and the shared key between the storage server and the receiver will be calculated through$PRIV_{SERV} = (PRIV_{SERV1}, PRIV_{SERV2}) =$ (c = $dPEKS.KeyGen_{SERV}(gp).PRIV_{SERV}, {PUB_{\mathrm{AUTH-SERV}}}^c$). Then the public key of server will be calculated as$PUB_{SERV} = (PUB_{SERV1}, PUB_{SERV2}) = ({g_p}^c, dPEKS.KeyGen_{SERV}(gp).PUB_{SERV})$.

$\mathbf{KeyGen_{DO}(gp)}$: In the case of requiring users to be restricted, a public/private key pair should be generated for the data owner as well. This key pair will be used within producing the signature for each index. This occurs when only certain authorized users must be able to produce the index. This digital signature can guarantee the server that the index is produced for the intended document. The public/private key pair is picked according to the used digital signature scheme.

***BuildTable*($gp, PUB_{SERV}, PRIV_{REC}, D$)**:

The receiver calls $dTrapdoor(gp, PUB_{SERV}, PRIV_{REC}, W)$ function in order to create a trapdoor for each word in the dictionary for the first time. Afterwards, he sends the created trapdoors for the authentication server. After receiving the trapdoors, the authentication server generates a table with two columns. The first entity of each row represents the trapdoor, and the second one will be initialized with 0.

dPEKS(w, ID_{doc}, $\mathbf{PUB_{SERV}}$, $\mathbf{PRIV_{DO}}$, $\mathbf{PUB_{REC}}$, gp): The DO produces an index for a keyword in order to enable searching capability for it. He accomplishes this action for every keyword in the document. Hence, two random members of ${Z_p}^*$ will be picked ($r, r' \in {Z_p}^*$) , then the intended index will be produced through $I = (I_1, I_2, I_3) = ((PUB_{REC})^r, g^{r'}, H_3\left(e(PUB_{SERV2}, H_2(w_i)^r)\right)^{r'})$

function. Afterwards, if it is required for users to be restricted, the produced index will be merged with the document ID and signed using the data owner's private key ($sign_{\boldsymbol{PRIV_{DO}}}(I \,||\, ID_{doc})$). The index will be sent to the server along with the signature subsequently. Finally, r' will be authenticated with the public key of the server ($Enc_{PUB_{\mathrm{AUTH-SERV}}}(r')$) and result value will be sent to the authentication server along with the index and its signature.

$\mathbf{UpdateTable(gp, C, PRIV_{SERV}, T_w)}$:

Receiving the index, the authentication server first determines to which trapdoor it belongs. In order to achieve this aim, the authentication server first calculates the shared key between server and itself through $KEY_{AUTH-SERV} = (PUB_{\mathrm{SERV1}})^{PRIV_{AUTH-SERV}}$. Following that, it performs testing through calling $dTest(gp, C, PRIV_{SERV}, T_w)$ and determines to which trapdoor the received index belongs. When the related trapdoor is recognized, the value of sum attribute associated with its entity will be added to the value of r' ($sum = sum + r'$).

$\boldsymbol{dTrapdoor(gp, PUB_{SERV}, PRIV_{REC}, W)}$: A trapdoor should be produced in order to perform searching for the intended keyword among the document. Hence, $dTrapdoor(gp, PUB_{SERV1}, PRIV_{REC}, W)$ function will be called.

$\mathbf{dTest(gp, C, PRIV_{SERV}, T_w)}$:

In order to perform testing, the server first calculates t = $\frac{\mathrm{T_2}}{\mathrm{H_1}(\mathrm{T_1}^{\mathrm{PRIV_{SERV2}}})}$ and in the case the following equation was true, the quested word is the **i**-th word and the server returns it to the user.

$$e(I_3, g) == e(H_2(e(I_1, t^{PRIV_{SERV2}}), I_2) \quad \textbf{(1)}$$

$\mathbf{Check(gp, Index, Signed-Index, PRIV_{SERV}, ID_{DO}, T_w)}$: After receiving results, user sends them along with his trapdoor to the authentication server in order to authenticate them. After receiving the receiver's request by the authentication server, it verifies signatures of indices. In the case of the signatures to be valid, it performs testing to determine the received trapdoor matches which trapdoor saved in the table. Afterwards, it calls the test function ($\boldsymbol{dTest(gp, I, KEY_{AUTH-SERV}, T_w)}$) with the shared key for each existing index. In the case of TRUE to be the result, the index matches the trapdoor and the server accomplished the comparison correctly. It verifies the $\boldsymbol{\prod I_2 == g^{sum}}$ equation to ensure all indices are returned by the server. All of them are sent correctly if the equation is true.

Proposed scheme based on SAE-I

This section merges our proposed scheme with public key encryption scheme with a designated tester. It includes eight algorithms which are explained as follows.

GlobalSetup(λ): Variables and functions needed within the steps ahead are produced in this step. Since the proposed scheme is an add-on for the SAE-I scheme, this function calls the $SAE-I.GlobalSetup(\lambda)$method. Furthermore, the generator g_p will be picked among Z_{p}. In the case of requiring users to be restricted, a MAC function or a digital signature will be picked.

$\mathbf{KeyGen_{REC}(gp)}$: Similar to the previous scheme, a public/private key pair for the receiver is required in order to produce index and trapdoor. The same algorithm used for producing the

receiver's key pair in SAE-I scheme is used in this scheme. ($PRIV_{REC} = \text{SAE} - \text{I}.KeyGen(REC).REC_{PRIV}$) and ($PUB_{REC} = \text{SAE} - \text{I}.KeyGen(REC).REC_{PUB}$).

$\text{KeyGen}_{\text{AUTH}-\text{SERV}}(\text{gp})$: Since the proposed scheme does not require a key to perform searching, the generated key pair will be used for encrypting and decrypting the data owner's messages (the value of r) and signing the result.

$\text{KeyGen}_{\text{DO}}(\text{gp})$: In this scheme the data owner's key pair is required to produce trapdoor and index, and the key pair is required for signing the index as well. consequently, this key pair will be generated through ($PRIV_{DO} = \text{SAE} - \text{I}.KeyGen(DO).DO_{PRIV}$) and ($PUB_{DO} = \text{SAE} - \text{I}.KeyGen(DO).DO_{PUB}$).

***BuildTable*(gp , PUB_{SERV}, $PRIV_{REC}$, D)**: The receiver calls $Trapdoor(gp\ , PUB_{SERV}, PRIV_{REC}\ , W)$ function in order to create a trapdoor for each word in the dictionary (**D**) for the first time. Afterwards, he sends the created trapdoors for the authentication server. After receiving the trapdoors, the authentication server generates a table with two columns. The first entity of each row represents the trapdoor, and the second one (**sum** attribute) will be initialized with 0.

BuildIndex(w, ID_{doc} , PRIV_{DO} , PUB_{REC}, gp):

A random number **r** is picked among $Z_p{}^*$ in order to create the index through $I = [\text{I}_1\ , \text{I}_2] = [g^r\ , (SAE - I.BuildIndex(PUB_{REC}\ ,\ PRIV_{DO}, w\ ))^r]$. Subsequently, the index will be concatenated with ID of the document ()$sign_{\boldsymbol{PRIV_{DO}}}(I\ ||\ ID_{doc})$() and r will be encrypted with the public key of the authentication server ($Enc_{PUB_{\text{AUTH}-\text{SERV}}}(r)$). Finally, the indices will be sent to the cloud server along with their signatures, and the encrypted r value will be sent to the authentication server along with the index and its signature.

UpdateTable(gp , C , $PRIV_{SERV}$, T_w):

Receiving the index, the authentication server first authenticates it using the signature, and then determines to which trapdoor it belongs. In order to achieve this aim, the authentication server performs testing through calling $Trapdoor(gp\ , KEY_{AUTH-SERV}, PRIV_{REC}\ , W)$ and determines to which trapdoor the received index belongs. Subsequently, the value of sum attribute associated with its entity will be added to the value of **r** ($sum = sum + r'$).

***dTrapdoor*(gp , PUB_{SERV}, $PRIV_{REC}$, W):** A trapdoor is required in order to perform searching for the intended keyword among the document. Hence, $SAE - I.Trapdoor(PUB_{DO}\ ,\ PRIV_{REC}, w\ )$ function will be called. The final trapdoor will be generated through

$$\text{T}_\text{w} = [\text{T}_1\ , \text{T}_2] = [g^{r'}\ , (SAE - I.Trapdoor(PUB_{DO}\ ,\quad PRIV_{REC}, w\ ))^{r'}] \tag{2}$$

in which r' is a random number among $Z_p{}^*$. Finally, the trapdoor will be sent to the server.

dTest(gp , C , T_w): Verifying $e(\text{I}_1\ , \text{T}_2) == e(\text{T}_1\ , \text{I}_2)$ equation, the test will be performed on cloud server side. In the case of TRUE to be the result, the index, its signature and the related document will be sent to the receiver as the result.

Check(gp , Index, Signed − Index , $PRIV_{SERV}$, ID_{DO} , T_w): After receiving results from the cloud server, user sends them along with his trapdoor to the authentication server in order to

authenticate them. After verifying the receiver's signature by the authentication server and in the case of the signature to be valid, it performs testing to determine which trapdoor saved in the table matched the received trapdoor. Afterwards, it calls the test function $\boldsymbol{Test}(\boldsymbol{gp}, \boldsymbol{I}, \boldsymbol{T_w})$ for each existing index in order to ensure all indices match the received trapdoors. It subsequently verifies the $\prod \boldsymbol{I_1} == \boldsymbol{g^{sum}}$ equation to ensure all indices are returned by the server. All of the documents are sent correctly if the equation is true.

6. Security and Performance Analysis of the Scheme

This section provides a comprehensive analysis of the proposed scheme from both security and performance perspectives. The analysis highlights the strengths of the scheme in terms of security guarantees and efficiency, while also addressing potential limitations.

6.1. Performance and Efficiency analysis

The proposed scheme introduces minimal overhead while providing robust validation mechanisms for returned results. The efficiency of the scheme can be analyzed based on the following key aspects:

Trapdoor Generation and Testing: The generation of trapdoors by the user and the execution of the test function by the server are computationally efficient. These operations are consistent with the original dPEKS and SAE-I schemes, ensuring that the proposed scheme does not introduce significant computational overhead during these critical steps. The types and number of operations required within each step of two proposed will be explained as follows. Besides the operations of index generation in dPEKS, the index generation phase of the first scheme needs a digital signature operation per keyword. The shared key can be generated through one exponential function, and the digital signature operation can be done through a MAC function per keyword. Therefore, if a document has w keywords, besides the operations of index generation in dPEKS, w digital signatures or an exponential function are required along with w times of performing MAC function. The operations required for index generation in dPEKS comprise two hash functions, two exponential functions, and one bilinear mapping per word. Hence, total operations required within index generation for a document which has w keywords comprise $2w$ hash functions, $2w$ exponential functions, w bilinear mappings and w digital signatures, or alternatively, $2w$ hash functions, $2w + 1$ exponential functions, w bilinear mappings and w times of performing MAC function.

Producing the trapdoor and performing the test function on the server-side are similar to the dPEKS scheme. Hence, three exponential functions and two hash functions are required within the trapdoor generation phase. Additionally, performing the test on the server-side requires a bilinear mapping, two exponential functions and two hash functions per each index existing on the server-side. If D documents exist on the server, each of which has w_{avg} keywords (indices) moderately, the number of indices existing on the server will be $w_{avg}.D$. Hence, this phase requires $w_{avg}.D$ bilinear mappings, $2.w_{avg}.D$ exponential functions and $2.w_{avg}.D$ hash functions. The proposed scheme includes an additional phase apart from the dPEKS which performs the verification of returned results. This phase is similar to the test function of the dPEKS, except for one more exponential

function. Hence, if D_{ret} documents are returned by the server, the required operations will be D_{ret} bilinear mappings, $2.D_{ret}+1$ exponential functions and $2.D_{ret}$ hash functions.

Operating the SAE-I within the index generation phase of the first proposed scheme, when the document contains w keywords, requires $2w$ hash function, w exponential functions, w bilinear mappings and w digital signatures, or alternatively, $2w$ hash functions, $w+1$ exponential functions, w bilinear mappings and w times of performing MAC function. Producing the trapdoor and performing the test function on the server-side are similar to the SAE-I scheme. Hence, one exponential function, one bilinear mapping and two hash functions are required within the trapdoor generation phase. Additionally, performing the test on the server-side is operated in $O(1)$ order of time and verification of returned documents is operated in $O(1)$ order of time as well.

Operating the dPEKS within the second proposed scheme requires one bilinear mapping, four exponential functions among G_1, one digital signature or MAC function, and two hash functions per each index generation. Hence, if the document contains w keywords, the required operations will be $2w$ hash function, $4w$ exponential functions, w bilinear mappings and w digital signatures, or alternatively, $2w$ hash functions, $4w+1$ exponential functions, w bilinear mappings and w times of performing MAC function. Producing the trapdoor is similar to the dPEKS scheme. Hence, three exponential functions and two hash functions are required within the trapdoor generation phase. Additionally, performing each test function requires three bilinear mappings, two exponential functions and one division among G_1, and two hash functions. If $w_{avg}.D$ indices exist on the server, this phase requires $3.w_{avg}.D$ bilinear mappings, $2.w_{avg}.D$ exponential functions and $2.w_{avg}.D$ hash functions. Moreover, performing the BuildTable function requires generating a trapdoor per each keyword in the dictionary by the receiver. Hence, if w_{Dict} words exist in the dictionary, this phase requires $3.w_{Dict}$ exponential functions and $2.w_{Dict}$ hash functions. Additionally, updating the table (UpdateTable) requires performing the test function up to the number of all keywords in the dictionary per each index. Hence, the worst-case of operating this phase requires $3.w_{Dict}.w$ bilinear mappings, $2.w_{Dict}.w$ exponential operations and $2.w_{Dict}.w$ hash functions.

Operating the SAE-I within the second proposed scheme, when the document contains w keywords, requires $2w$ hash function, $3w$ exponential functions, w bilinear mappings and w digital signatures, or alternatively, $2w$ hash functions, $3w+1$ exponential functions, w bilinear mappings and w times of performing MAC function. The required operations for producing the trapdoor are similar to the previous step. Hence, the trapdoor generation phase requires three exponential functions and two hash functions. If $w_{avg}.D$ indices exist on the server, operating the test phase requires $2.w_{avg}.D$ bilinear mappings and verification requires $2.w_{ret}$ bilinear mappings. Moreover, performing the BuildTable function requires and $2.w_{Dict}$ hash functions, $3.w_{Dict}$ exponential functions and w_{Dict} bilinear mappings. Additionally, the worst-case of operating the phase of updating the table (UpdateTable) requires $2.w_{Dict}.w$ bilinear mappings.

Scalability: The proposed scheme is designed to scale effectively in environments with multiple servers. By leveraging the Diffie-Hellman key exchange, the scheme allows multiple servers to perform testing collaboratively, reducing the search time and improving overall system performance. This is particularly beneficial in large-scale cloud environments where search operations are frequent and resource-intensive.

Storage Overhead: The scheme requires additional storage for digital signatures or MAC values associated with each index. However, this storage overhead is minimal compared to the security benefits it provides. The use of hash tables for storing indices further optimizes the search process, ensuring that the validation can be performed in constant time O (1).

Table 1 compares the required operations of the proposed schemes with other schemes, and ***Table 2*** compares the performance of them. There are some symbols and notions in ***Table 1*** which are as follows: the symbols e and h show exponential functions and hash functions respectively; ${p_p}^s$ marks the symmetric bilinear mapping; performing the MAC function is marked as MAC; and $mult$ is used to show the multiplication operation. In summary, the proposed scheme achieves a balance between security and efficiency. While it introduces some additional computational and storage costs, these are justified by the enhanced security guarantees and the ability to validate the integrity of returned results.

Table 1. The number of required operations for the proposed schemes is compared to that of other schemes.

Scheme	Encryption	Trapdoor Generation	Searching by server	Verification	Building table	Updating table
PEKS	$w(2e+2h+{p_p}^s)$	$h+e$	$w_{avg}.D.(h+{p_p}^s)$	-	-	-
dPEKS	$w(2e+2h+{p_p}^s)$	$3e+2h$	$w_{avg}.D.(2e+2h+{p_p}^s)$	-	-	-
SAE-I	$w(2h+e+{p_p}^s)$	$2h+e+{p_p}^s$	$O(1)$	-	-	-
[27]	$w(3e+3h+{p_p}^s)$	$3e+2h$	$w_{avg}.D.(2e+2h+{p_p}^s)+e+D_{ret}.Mult$	$(D_{ret}+1).Mult+e$	-	-
The first scheme merged with dPEKS	$w(2e+2h+{p_p}^s+MAC)+e$	$3e+2h$	$w_{avg}.D.(2e+2h+{p_p}^s)$	$D_{ret}.(2e+2h+{p_p}^s)+e$	-	-
The first scheme based on SAE-I	$w(2h+e+{p_p}^s+MAC)+e$	$2h+e+{p_p}^s$	$O(1)$	$O(1)$	-	-
The second scheme merged with dPEKS	$w(4e+2h+{p_p}^s+MAC)+e$	$3e+2h$	$w_{avg}.D.(2e+2h+3{p_p}^s)$	$D_{ret}.(2e+2h+3{p_p}^s)+e$	$w_{Dict}(3e+2h)$	$w_{Dict}.w(2e+2h+3{p_p}^s)$
The second scheme based on SAE-I	$w(2h+3e+{p_p}^s)$	$2h+e+{p_p}^s$	$2.w_{avg}.D.{p_p}^s$	$2.D_{ret}.{p_p}^s$	$w_{Dict}(2h+3e+{p_p}^s)$	$2.w_{Dict}.w.{p_p}^s$

Table 2. The performance of the proposed schemes is compared to that of other schemes.

Scheme	KGA attack	IKGA attack	Publicity	Verification of received results	Capability of detecting file removal	Fair payment	Capability of processing by multiple servers	Adapting with organization circumstances
PEKS	✗	✗	✓	✗	✗	✗	✓	✓
dPEKS	✓	In case of utilizing [34]	✓	✗	✗	✗	✗	✓
SAE-I	✓	✓	✗	✗	✗	✗	✓	✓
[26]	✗	✗	✓	✓	✓	✗	✓	✓
[27]	✓	In case of utilizing [34]	✓	✓	✗	✗	✗	✓
[29]	✗	✗	✓	✓	✓	✗	✓	✓
[33]	✓	In case of utilizing [34]	✓	✓	✓	✓	✗	✗
First scheme	✓	In case of utilizing [34] or SAE-I	✓	✓	✗	✗	✓	✓
Second scheme	✓	In case of utilizing [34] or SAE-I	✓	✓	✓	✗	✓	✓

6.2. Security analysis

The proposed schemes build upon the security foundations of the dPEKS and SAE-I schemes while introducing additional mechanisms to ensure the integrity of returned results. The security analysis focuses on the following key aspects:

Protection Against Malicious Servers: The primary objective of the proposed scheme is to prevent malicious servers from returning incorrect or manipulated results. By concatenating the index with the document ID and signing it, the scheme ensures that the server cannot forge or modify the indices. This mechanism guarantees that only valid, authenticated indices are returned to the user.

Resistance to Keyword Guessing Attacks: The scheme is designed to resist both offline and online keyword guessing attacks. By limiting the test function to the server and requiring the server's private key for testing, the scheme prevents external attackers from performing offline keyword guessing. Additionally, the use of digital signatures ensures that even if an attacker intercepts a trapdoor, they cannot generate a valid index without the data owner's private key.

Resistance to Injection Attacks: The scheme is resistant to injection attacks, where an external attacker attempts to inject malicious indices into the system. By requiring digital signatures for each index, the scheme ensures that only indices generated by authorized data owners are accepted.

The security of the second proposed scheme is analyzed in this section utilizing the dPEKS scheme. The security requirements of this scheme are stated as follows:

- An external intruder should be unable to recognize for which word an index has been generated by the user.
- An external intruder should be unable to recognize for which word the eavesdropped index has been generated.
- Having the public keys of the authentication server and the storage server, an external intruder should be unable to obtain the shared key between them.
- Having a shared key with the storage server, an authentication server should not be able to obtain the shared key between the storage server and another authentication server. Therefore, the authentication server will not be able to perform tests unassociated with itself (e.g. other zones of the organization).
- Having a shared key with the authentication server, a storage server should not be able to obtain the shared key between the authentication server and another storage server. Therefore, the storage server will not be able to perform tests unassociated with itself (e.g. data locating on another server).

The three last security requirements are fulfilled due to utilizing the Diffie-Hellman key exchange method, which method has the following characteristics:

- It is not possible to calculate g^{ab} during a polynomial time, knowing only g^a and g^b without a or b. Hence, it is required for an intruder to have $PRIV_{serv}$ or $PRIV_{aut-serv}$ in order to be able to calculate the shared key between the authentication server and the storage server as $g^{PRIV_{serv}.PRIV_{aut-serv}}$. Since he has none of these values, he cannot obtain the shared key.
- Considering a communication between three users a, b, c utilizing the Diffie-Hellman key exchange method, if they have three shred keys for mutual communications, i.e. $g^{PRIV_a.PRIV_b}$, $g^{PRIV_b.PRIV_c}$ and $g^{PRIV_a.PRIV_c}$, the user c, who does not have $PRIV_a$ and $PRIV_b$, cannot calculate the value of $g^{PRIV_a.PRIV_b}$ having his own private key and two shared keys i.e.

$g^{PRIV_a \cdot PRIV_c}$ and $g^{PRIV_b \cdot PRIV_c}$. Therefore, the authentication server 1 cannot access the shared key between the storage server and the authentication server 2 ($g^{PRIV_{serv} \cdot PRIV_{aut-serv_2}}$) utilizing $g^{PRIV_{serv} \cdot PRIV_{aut-serv_1}}$. Similarly, the storage server 1 cannot access the shared key between the authentication server and the storage server 2 ($g^{PRIV_{aut-serv} \cdot PRIV_{serv_2}}$) utilizing $g^{PRIV_{aut-serv} \cdot PRIV_{serv_1}}$.

According to the two mentioned characteristics, the three last attacks are not feasible. Since trapdoor generation in the proposed scheme is similar to the trapdoor generation algorithm of the dPEKS scheme, the security of the proposed scheme against the mentioned attacks is similar to the one of the dPEKS. The security of the proposed scheme against the first attack will be described as follows: Proving the security of indices in this scheme is performed through a security game which is illustrated in ***Figure 3***. There are two parties in this game who are the attacker and the challenger. The attacker intends to realize for which keyword his obtained index has been generated, and the challenger generates keys and indices. The game is played as follows:

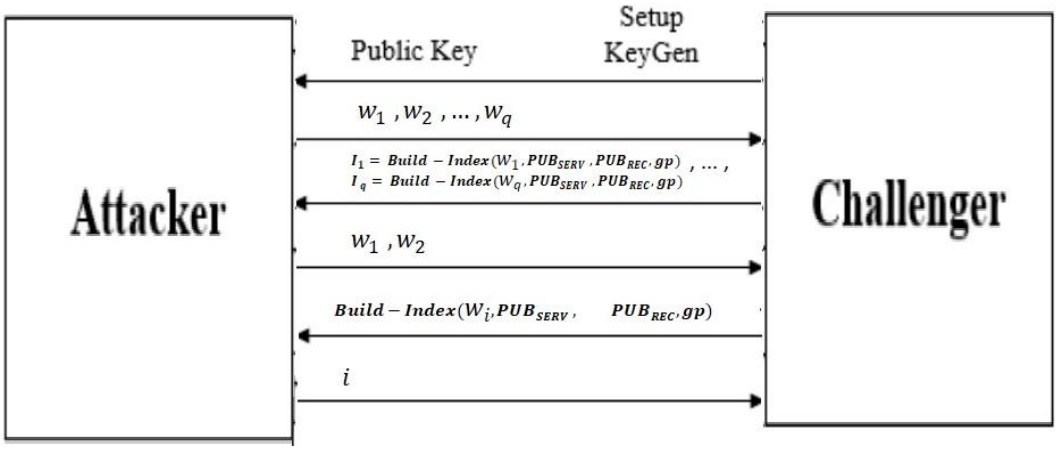

Figure 4. The presented Security game scheme

- First, the challenger performs the $Setup(k)$ algorithm in order to produce the required parameters. Afterwards, he generates public and private key-pairs for the receiver and the authentication server by performing $KeyGen_{\mathrm{REC}}(gp)$ and $KeyGen_{\mathrm{SERV}}(gp)$ algorithms, and generates the index for his intended keyword utilizing these two key-pairs.
- In the second step, an attacker requests the challenger to generate indices for a number of his intended keywords ($w_1, w_2, \dots, w_q$).
- The challenger calls the function $Build - Index(w_i, PUB_{SERV}, PUB_{REC}, gp)$ in order to generate indices, and sends them to the attacker subsequently.
- The attacker picks two keywords and sends them to the challenger.
- The challenger selects one of the two received keywords and generates an index for it, and sends the index for the attacker.
- Here, the attacker has to recognize to which word the received index belongs.

It can be easily proved that the adversary's advantage of the proposed algorithm is lower than the adversary's advantage of recognizing the word associated with the generated index using the dPEKS. Hence, if an attacker can solve it with the advantage ε, he can recognize the word associated with the generated index using the dPEKS with a higher advantage. Therefore, the adversary's advantage of this algorithm is negligible since it is proven in [15] that the adversary's advantage of recognizing the word associated with the generated index using the dPEKS is negligible. it is

explained as follows that the adversary's advantage of the proposed algorithm is lower than the adversary's advantage of the dPEKS:

If an algorithm exists that is able to guess the word w_i having a submitted index $(((PUB_{REC})^r, g^{r'}, H_3(e(PUB_{SERV2}, H_2(w_i)^r))^{r'}))$, an attacker will be able to send the trapdoor $(I' = (I'_1, I'_2, I'_3) = ((PUB_{REC})^r, g, H_3(e(PUB_{SERV2}, H_2(w_i)^r))))$ to the oracle by inserting $r' = 1$. This is despite the fact that the eavesdropped trapdoor, which is generated through the dPEKS, $I = (I_1, I_2) = ((PUB_{REC})^r, H_3(e(PUB_{SERV2}, H_2(w_i)^r)))$ is . In other words, it is enough to send the index as $I' = (I_1, g, I_2)$ to the oracle after eavesdropping it, and send the word returned by the oracle to the challenger.

7. Conclusions

In this study, we presented a novel method for validating the integrity of results returned from a cloud server in asymmetric searchable encryption schemes. Our approach leverages the Diffie-Hellman key exchange and digital signatures to ensure that the server cannot return data with incorrect or manipulated indices. This is achieved by concatenating the index with the document ID and signing it, ensuring that the server cannot forge or manipulate the indices. However, it is important to note that this scheme is most effective when the server is expected to return all relevant documents. In scenarios where the server may skip certain documents due to resource constraints, the scheme's effectiveness may be limited. This method is particularly effective in scenarios where the server may act maliciously/ maliciously/honest but curious or where data integrity is a critical concern. The proposed scheme introduces minimal overhead, requiring only a digital signature or an additional MAC function per keyword. Importantly, the generation of trapdoors by users and the execution of the test function remain efficient, with no significant performance degradation. The validation process, performed by the authentication server, involves a digital signature verification, an exponential function, and a dPEKS test function, ensuring that the returned results are both accurate and trustworthy. Furthermore, the proposed shared key mechanism can be extended to the dPEKS scheme, enabling multiple servers to perform testing collaboratively. This extension enhances the scalability and flexibility of the system, making it suitable for larger, distributed environments.

Funding sources

We gratefully acknowledge the financial support from the Iran National Science Foundation (INSF) [Research project 97008930].

References

[1] Fei Han, Jing Qin, Jiankun Hu, (2016). Secure searches in the cloud: A survey. *Future Generation of Computer Systems*, Volume 62, Pages 66-75, https://doi.org/10.1016/j.future.2016.01.007.

[2] Bösch, C., Hartel, P., Jonker, W., Peter, A., (2014). A Survey of Provably Secure Searchable Encryption. *ACM Computing Surveys*, Vol. 47, 2, pages 1-51, https://doi.org/10.1145/2636328.

[3] Arabnouri, A., Ebrahimi Atani, R., Azizzadeh, S., (2020). Security Analysis of Public Key Searchable Encryption Schemes against Injection Attacks", *Cryptology ePrint Archive*, Report 2020/1530, https://eprint.iacr.org/2020/1530.

[4] Song, D.X., Wagner, D., Perrig, A., (2000). Practical techniques for searches on encrypted data. *Proceeding 2000 IEEE Symposium on Security and Privacy*. S&P 2000, Berkeley, CA, USA, 2000, pp. 44-55, https://doi.org/10.1109/SECPRI.2000.848445.
[5] Goh, E., .Secure indexes. *Cryptology ePrint Archive*, Rep. 2003/216, https://eprint.iacr.org/2003/216.
[6] Chang, YC., Mitzenmacher, M., (2005) "Privacy Preserving Keyword Searches on Remote Encrypted *Data. Applied Cryptography and Network Security (ACNS)*, Lecture Notes in Computer Science, Vol 3531. Springer, Berlin, Heidelberg, https://doi.org/10.1007/11496137_30.
[7] Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R., (2011). Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions., *Journal of Computer Security*, Vol. 19, No. 5, Pages: 895-934, https://doi.org/10.3233/JCS-2011-0426.
[8] Golle, P., Staddon, J., Waters, B., (2004). Secure Conjunctive Keyword Search over Encrypted Data. *Applied Cryptography and Network Security. ACNS 2004*. Lecture Notes in Computer Science, Vol 3089. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-24852-1_3.
[9] Kurosawa, K., Ohtaki, Y., (2012). UC-Secure Searchable Symmetric Encryption. *Financial Cryptography and Data Security. FC 2012*. Lecture Notes in Computer Science, Vol 7397. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-32946-3_21.
[10] Chai, Q., Gong, G., (2012). Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. *IEEE International Conference on Communications (ICC)*, Ottawa, ON, pages: 917-922, https://doi.org/10.1109/ICC.2012.6364125.
[11] Moataz, T., Shikfa, A., (2013). Boolean symmetric searchable encryption", *ASIA CCS '13: Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, Pages 265–276, https://doi.org/10.1145/2484313.2484347.
[12] Chamani, J., Papadopoulos, D., Papamanthou, C., Jalili, r., (2018) .New Constructions for Forward and Backward Private Symmetric Searchable Encryption. *CCS '18: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Pages 1038–1055, https://doi.org/10.1145/3243734.3243833.
[13] Boneh, D., Crescenzo, G, Ostrovsky, R., Persiano, G., (2004). Public Key Encryption with Keyword Search", *Advances in Cryptology - EUROCRYPT 2004*. Lecture Notes in Computer Science, Vol 3027. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-24676-3_30.
[14] Tang, Q., Chen, L., (2010). Public-Key Encryption with Registered Keyword Search. *Public Key Infrastructures, Services and Applications. EuroPKI 2009*. Lecture Notes in Computer Science, Vol 6391. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-16441-5_11.
[15] Rhee, H. S., Park, J. H., Susilo, W., Lee, D. H., (2010). Trapdoor security in a searchable public-key encryption scheme with a designated tester. *Journal of Systems and Software*, Vol. 83, Issue 5, Pages 763-771, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2009.11.726.
[16] Yau, W., Phan, R., Heng, W., Goi, B., (2013). Keyword guessing attacks on secure searchable public key encryption schemes with a designated tester. *International Journal of Computer Mathematics*, Vol. 90(12), Pages: 2581-2587, https://doi.org/10.1080/00207160.2013.778985.
[17] Chen, Y., (2015). SPEKS: Secure Server-Designation Public Key Encryption with Keyword Search against Keyword Guessing Attacks. *The Computer Journal*, Vol. 58, Issue 4, Pages Pages: 922–933, https://doi.org/10.1093/comjnl/bxu013.
[18] Chen, R., Mu, Y., Yang, G., Guo, F., Huang, X., Wang, X., Wang, Y., (2016). Server-Aided Public Key Encryption With Keyword Search. *IEEE Transactions on Information Forensics and Security*, Vol. 11, No. 12, pages: 2833-2842, https://doi.org/10.1109/TIFS.2016.2599293.
[19] Sun, L., XU, C., Zhang, M., Chen, K., (2018). Hongwei LI, "Secure searchable public key encryption against insider keyword guessing attacks from indistinguishability obfuscation. *Journal of Science China Information Sciences*, Vol. 61, Issue 3, https://doi.org/10.1007/s11432-017-9124-0.
[20] Zhang, J., Song, C., Wang, Z., Yang, T., Ma, W., (2018). Efficient and Provable Security Searchable Asymmetric Encryption in the Cloud. *IEEE Access*, Vol. 6, pp. 68384-68393, https://doi.org/10.1109/ACCESS.2018.2872743.
[21] Chen, R., Mu, Y., Yang, G., Guo, F., Wang, X., (2015). A New General Framework for Secure Public Key Encryption with Keyword Search. *Information Security and Privacy. ACISP 2015*, Lecture Notes in Computer Science, Vol 9144. Springer, Cham. https://doi.org/10.1007/978-3-319-19962-7_4.
[22] Liu, P., Wang, J., Ma, H., Nie, H., (2014). Efficient Verifiable Public Key Encryption with Keyword Search Based on KP-ABE. *Ninth International Conference on Broadband and Wireless Computing, Communication and Applications*, Pages: 584-589, https://doi.org/10.1109/BWCCA.2014.119.

[23] Hwang, Y., Lee, P., (2007). Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-user System. *Pairing-Based Cryptography – Pairing 2007*, Lecture Notes in Computer Science, Vol 4575, Springer, https://doi.org/10.1007/978-3-540-73489-5_2.

[24] Shi, E., Bethencourt, J., Chan, T. H., Song, D., Perrig, A., (2007). Multi-Dimensional Range Query over Encrypted Data. *IEEE Symposium on Security and Privacy (SP'07)*, Berkeley, CA, 2007, pp. 350-364, https://doi.org/10.1109/SP.2007.29.

[25] Boneh, D., Waters, B., (2007). Conjunctive, Subset, and Range Queries on Encrypted Data. *Theory of Cryptography. TCC 2007*, Lecture Notes in Computer Science, Vol 4392. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-70936-7_29.

[26] Boneh, D., Kushilevitz, E., Ostrovsky, R., Skeith, W.E., (2007). Public Key Encryption That Allows PIR Queries. *Advances in Cryptology - CRYPTO 2007*. Lecture Notes in Computer Science, Vol 4622. Springer, Berlin, Heidelberg, https://doi.org/10.1007/978-3-540-74143-5_4.

[27] Arabnouri, A., Security Evaluation of Searchable Encryption protocols. MSc Thesis, University of Guilan, 2019, (in Persian).

[28] Arabnouri, A., Ebrahimi Atani, R., Azizzadeh, S., (2020). Security Analysis of Public Key Searchable Encryption Schemes against Injection Attacks", *Cryptology ePrint Archive*, Report 2020/1553, https://eprint.iacr.org/2020/1530.

[29] Arabnouri, A., Shafieinejad, A., (2024). BACASE-SH: Blockchain-based authenticated certificate-less asymmetric searchable encryption for smart healthcare. *Peer-to-Peer Networking and Applications*, Vol.17, Pages: 2298–2314, https://doi.org/10.1007/s12083-024-01687-x.

[30] Amorim, I., Costa, I. (2023). Leveraging Searchable Encryption through Homomorphic Encryption: A Comprehensive Analysis. *Mathematics*, Vol.11(13), 2948. https://doi.org/10.3390/math11132948.

[31] Duan, G., Li, S., (2023). Verifiable and Searchable Symmetric Encryption Scheme Based on the Public Key Cryptosystem. *Electronics*, Vol. 2(18), 3965. https://doi.org/10.3390/electronics12183965

[32] Fugkeaw, S., Hak, L., Theeramunkong, T., (2024). Achieving Secure, Verifiable, and Efficient Boolean Keyword Searchable Encryption for Cloud Data Warehouse. *IEEE Access*, Vol. 12, pp. 49848-49864, https://doi.org/10.1109/ACCESS.2024.3383320.

[33] Shen, F., Shi, L., Zhang, J., Xu, C., Chen, Y., He, Y., (2024). BMSE: Blockchain-based multi-keyword searchable encryption for electronic medical records. *Computer Standards & Interfaces*, Vol. 89, 103824, ISSN 0920-5489, https://doi.org/10.1016/j.csi.2023.103824.

[34] Meng, L., Chen, L., Tian, Y., Manulis, M., Liu, S., (2024). FEASE: Fast and Expressive Asymmetric Searchable Encryption. *33rd USENIX Security Symposium*, ISBN: 978-1-939133-44-1, pages = 2545-2562.

[35] Qingji Zheng, Shouhuai Xu, Giuseppe Ateniese, "VABKS: Verifiable attribute-based keyword search over outsourced encrypted data," IEEE INFOCOM 2014 - IEEE Conference on Computer Communications, Toronto, ON, (2014), pp. 522-530, https://doi.org/10.1109/INFOCOM.2014.6847976

[36] Binrui Zhu, Jiameng, SunJing Qin, Jixin Ma (2018) "The Public Verifiability of Public Key Encryption with Keyword Search". In: Hu J., Khalil I., Tari Z., Wen S. (eds) Mobile Networks and Management. MONAMI 2017. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 235. Springer, Cham. https://doi.org/10.1007/978-3-319-90775-8_24

[37] Pengliang Liu, Jianfneg Wang, Hua Ma, Haixin Nie, "Efficient Verifiable Public Key Encryption with Keyword Search Based on KP-ABE", 2014 Ninth International Conference on Broadband and Wireless Computing, Communication and Applications, Guangdong, 2014, pp. 584-589, https://doi.org/10.1109/BWCCA.2014.119

[38] Rui Zhang, Rui Xue, Ting Yu, Ling Liu, "PVSAE: A Public Verifiable Searchable Encryption Service Framework for Outsourced Encrypted Data," 2016 IEEE International Conference on Web Services (ICWS), San Francisco, CA, 2016, pp. 428-435, https://doi.org/10.1109/ICWS.2016.62

[39] Shengshan Hu, Chengjun Cai, Qian Wang, Cong Wang, Xiangyang Luo, Kui Ren, "Searching an Encrypted Cloud Meets Blockchain: A Decentralized, Reliable and Fair Realization," IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, Honolulu, HI, 2018, pp. 792-800, https://doi.org/10.1109/INFOCOM.2018.8485890

[40] Yinghui Zhang, Robert H. Deng, Jiangang Shu, Kan Yang, Dong Zheng, "TKSE: Trustworthy Keyword Search Over Encrypted Data With Two-Side Verifiability via Blockchain," in IEEE Access, vol. 6, pp. 31077-31087, 2018, https://doi.org/10.1109/ACCESS.2018.2844400

[41] Shahzaib Tahir, Muttukrishnan Rajarajan, "Privacy-Preserving Searchable Encryption Framework for Permissioned Blockchain Networks," 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and

Social Computing (CPSCom) and IEEE Smart Data (SmartData), Halifax, NS, Canada, 2018, pp. 1628-1633, https://doi.org/10.1109/Cybermatics_2018.2018.00272

[42] Rahnama Lashkami, S., Ebrahimi Atani, R., Arabnouri, A., Salemi, G., (2020). A Blockchain Based Framework for Complete Secure Data Outsourcing with Malicious Behavior Prevention. 28th Iranian Conference on Electrical Engineering (ICEE), Tabriz, Iran, 2020, pp. 1-7, https://doi.org/10.1109/ICEE50131.2020.9260866

[43] Arabnouri, A., Ebrahimi Atani, R., (2018). Asymmetric searchable encryption secure against Insider Key Guessing Attacks. The 26th Iranian Conference on Electrical Engineering (ICEE) 08-10 May 2018, Mashhad, Iran.