A Public Key Searchable Encryption Scheme Based on Blockchain Using Random Forest Method

Bhargavi Konda[1], Mounica Yenugula[2], Vinay Kumar Kasula[3], Akhila Reddy Yadulla[4]
[1,2,3,4]*Department of Information Technology, University of the Cumberlands, Williamsburg, KY, USA*
[1]*bkonda19519@ucumberlands.edu,* [2]*myenugula3188@ucumberlands.edu,*
[3]*vkasula19501@ucumberlands.edu,* [4]*ayadulla5882@ucumberlands.edu*

***Abstract*—**To address the trapdoor security issue in public key encryption schemes, this paper introduces the use of random numbers to construct trapdoors and indices, effectively defending against keyword-guessing attacks from internal servers and preventing data leaks caused by server curiosity. The paper explores the trust issues surrounding third-party services and combines blockchain technology with a searchable encryption scheme. Smart contracts are employed as trusted third parties to perform the search operations, preventing keyword-guessing attacks from internal servers while ensuring the correctness of the search results. This approach also restricts malicious server behavior when distributing data. A security analysis confirms that the proposed scheme satisfies IND-KGA security. Furthermore, experimental comparisons with other schemes demonstrate the proposed method's advantage in terms of time efficiency.

Keywords*—Searchable encryption, Blockchain, Smart contracts, Public key encryption, Random forest method*

I. INTRODUCTION

Cloud storage has become a dominant method for online storage, offering the advantage of eliminating hardware and management costs for users. However, as data moves beyond the physical control of users, security risks have become a significant concern. Encryption is typically used to secure data stored in the cloud, but searching through encrypted data on cloud servers poses a challenge. Secure search refers to effective searching over encrypted data. To address the problem of how to perform secure keyword searches on encrypted data stored in untrusted cloud servers, researchers have proposed searchable encryption as a core solution. Searchable encryption is a technology that allows users to perform keyword searches over encrypted data. It is particularly useful in cloud environments where users want to store encrypted data and selectively retrieve relevant documents through keyword searches. This enables users to search encrypted domains without decrypting the data stored in the cloud. In 2000, Dawn et al. introduced the one-to-one model for searchable encryption to enhance data security on servers, sparking further research in the field. However, the one-to-one model couldn't meet growing user demands. In 2004, Boneh et al. introduced the many-to-one public key encryption with keyword search (PEKS) model, which defined the security of searchable encryption under public key encryption. However, this model proved impractical in some specific environments. In 2011, Curtmola et al. developed a one-to-many searchable encryption model based on Naor's broadcast encryption technology, though it required significant key update costs for users. In complex network environments, Wang et al. combined Shamir's secret sharing and identity-based encryption to propose a many-to-many searchable encryption scheme, allowing multi-user interactive searches on the server.

In terms of searchable encryption security, Boneh et al. proved that PEKS achieves semantic security but is vulnerable to keyword guessing attacks (KGA). To counter this, Tang proposed a registration-based searchable encryption model in 2009, which resisted KGA but required pre-registering keywords, limiting performance. In 2013, Fang et al. introduced a public key encryption scheme that addressed KGA through a two-pronged security model targeting both internal and external attacks. However, the heavy use of bilinear pairing calculations made the scheme inefficient. Recent studies have explored various methods to protect against internal attacks. For instance, Xu et al. in 2013 introduced a dual-trapdoor mechanism (fuzzy and exact trapdoors) to resist internal KGA. In 2015, Chen proposed a new framework involving two non-colluding servers to prevent internal attacks. Shao et al. redefined the security of dPEKS under KGA, focusing on offline attacks by servers.

In 2016, Chen proposed a more efficient two-cloud-server scheme to resist internal attacks, though its practicality was limited by the assumption that the servers wouldn't collaborate. Huang et al. in 2017 introduced a public key authentication encryption scheme to address internal KGA using bilinear pairing. However, this scheme couldn't provide indistinguishability for chosen ciphertexts. With the development of blockchain technology, the combination of blockchain and searchable encryption has solved the trust issues in traditional schemes, significantly improving feasibility. Li et al. introduced a blockchain-based symmetric searchable encryption scheme, while Chen et al. proposed a

blockchain-based scheme for sharing electronic medical records using smart contracts to ensure server trustworthiness. This paper addresses the trust issues of third-party services by incorporating blockchain into a public key searchable encryption scheme. The primary contributions of this work are:

1. Blockchain Integration: A blockchain mechanism is introduced into the ciphertext retrieval scheme to address the trust issues of third-party services in traditional schemes. Blockchain is utilized to ensure the correctness of search results, leveraging its immutability to prevent servers from maliciously or mistakenly sending incorrect data.
2. One-to-Many Public Key Searchable Encryption: A one-to-many public key searchable encryption scheme is constructed for private cloud environments, utilizing the decisional bilinear Diffie-Hellman (DBDH) problem to ensure unique encryption for the same keyword and effectively resist KGA.
3. Security Proof: The proposed scheme is proven to resist KGA attacks, with an analysis of how blockchain contributes to the overall security of the system. Experiments conducted using the PBC (pairing-based cryptography) library demonstrate the efficiency of the scheme in terms of index and trapdoor construction, as well as query performance.

II. PRELIMINARY KNOWLEDGE

A. *Bilinear Mapping (Bilinear Pairing)*

In cryptography, bilinear mapping is an essential concept used in pairing-based cryptographic schemes. Assume we have two cyclic groups G and G_T, both of prime order p, with g being a generator of G. A bilinear map e is defined as $e: G \times G \rightarrow G_T$, and it satisfies the following properties:

1. Bilinearity: For any elements $x, y \in G$ and scalars $a, b \in Z_p$, the bilinear map holds that:

$$e(x^a, y^b) = e(x, y)^{ab}$$

This property ensures that the map behaves linearly in both of its input arguments.

2. Non-degeneracy: There exists some generator $g \in G$ such that $e(g, g) \neq 1$. This means that the map does not trivially collapse to a constant, ensuring its utility in cryptographic constructions.
3. Computability: For all $x, y \in G$, the bilinear map $e(x, y)$ can be efficiently computed using algorithms designed for such operations.

In the context of the random forest method, this bilinear mapping serves as the foundational mathematical structure upon which the encryption and security mechanisms are built. By incorporating bilinear pairing, the system can manage complex cryptographic operations, enabling secure key exchanges and data searches across distributed environments.

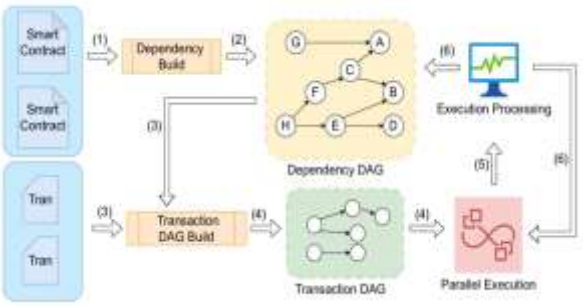

Figure 1: System Model and Workflow

III. SYSTEM MODEL: RANDOM FOREST IMPLEMENTATION

A. *System Overview*

The proposed system consists of four main components: the Data Owner (DO), Cloud Server (CS), Smart Contract, and User (U). These components interact to ensure secure storage, retrieval, and validation of encrypted data. The flow of the system is illustrated as follows:

1. Data Owner (DO): The primary role of the data owner is to compute an index for the data and encrypt the plaintext. The data owner then uploads the index to the smart contract and the ciphertext to the cloud server.
2. Cloud Server (CS): The cloud server stores the encrypted data uploaded by the data owner and processes data retrieval requests from the user. It interacts with the smart contract to verify the authenticity of data requests and provide the correct ciphertext.
3. Smart Contract: The smart contract receives the index from the data owner and the trapdoor (search query) from the user. It performs a search on the index and returns the relevant search results. It also verifies the correctness of the ciphertext received from the cloud server and issues a command to the server to release the encrypted data to the user if the search is successful.
4. User (U): The user is responsible for generating a trapdoor (search query) and sending it to the smart contract. The user also submits a request

to the cloud server for the encrypted data and, after receiving it, decrypts the data for use.

This design integrates a Random Forest algorithm to facilitate the search and classification process. The Random Forest method is implemented to enhance the accuracy of the search results by analyzing multiple indices and providing a weighted decision on the most relevant results based on the user's query.

B. Algorithm Definitions

The following algorithms are integral to the system model:

1. Setup:

$setup(1^{\lambda}) \rightarrow par$

The system is initialized with the security parameter λ, generating public parameters par.

2. Encryption (Enc):

$Enc(m, k, w) \rightarrow (C_m, I, N)$

The data owner performs encryption using the plaintext m and the symmetric key k. The keyword w is also encrypted to generate the index I. The plaintext m and key k are encrypted to produce the ciphertext C_m, and the file identifier N is generated using symmetric encryption for identification. The system computes a hash value H for N and C_m, producing the final encrypted result CT.

3. Trapdoor (Tw):

$Tw(w_i) \rightarrow T_w$

The user computes a trapdoor T_w by encrypting the keyword w_i. This trapdoor is then sent to the smart contract.

4. Search:

$search(T_w, I) \rightarrow (k, N, H)$

The smart contract performs the search after receiving the trapdoor T_w and index I. If the search is successful, it retrieves the symmetric encryption key k, the file identifier N, and the hash value H.

5. Verification:

$verify(CT, N) \rightarrow \{0, 1\}$

Upon receiving the ciphertext CT from the cloud server, the user verifies whether the server has correctly provided the data and whether the ciphertext has been tampered with. The user checks whether the file identifier N and hash H match those from the smart contract. If they do, the verification succeeds, returning 1; otherwise, it returns 0.

C. Security Model

A. Keyword Privacy Security Game

The keyword privacy of the system is guaranteed if an adversary A cannot deduce the plaintext keyword from the encrypted keyword or trapdoor in polynomial time. The keyword privacy security game is defined as follows:

1. Initialization: Given the security parameter λ, the challenger C runs the initialization algorithm, generating public parameters par.
2. Phase 1: The adversary A runs the trapdoor generation algorithm multiple times.
3. Challenge: The adversary A randomly selects two keywords from the keyword space and sends them to the challenger. The challenger runs the trapdoor generation algorithm for both and randomly sends one trapdoor back to the adversary.
4. Guess: After analyzing τ different keywords, the adversary guesses which keyword the trapdoor corresponds to. If the guess is correct, the adversary wins the security game.

B. Decisional Bilinear Diffie-Hellman (DBDH) Assumption

The security of the system relies on the hardness of solving the Decisional Bilinear Diffie-Hellman (DBDH) problem. If an adversary A can break the scheme in polynomial time with advantage ϵ, they must also be able to solve the DBDH problem with the same advantage ϵ. The proof is structured as follows:

1. Initialization: The challenger C initializes the group G_1, G_2, and bilinear map $e: G_1 \times G_1 \rightarrow G_2$. The challenger generates random values a, b, c, z and creates two sets of tuples T_0 and T_1.
2. Phase 1: The adversary A runs the encryption algorithm multiple times.
3. Challenge: The challenger randomly selects a plaintext m that has not been queried in Phase 1 and generates the corresponding ciphertext C_m, which is sent to the adversary.
4. Guess: The adversary attempts to decrypt the ciphertext C_m. If the adversary successfully decrypts C_m and retrieves the correct plaintext, they win the game.
5. Proof: If the adversary is capable of decrypting the ciphertext, then they can solve the DBDH problem, contradicting the assumption that DBDH is a hard problem. Thus, the security of the scheme is upheld.

IV. System Description with Random Forest Implementation

A. Initialization Phase

In the initialization phase, the system sets up its parameters using security parameter λ. The system generates public parameters par, including cyclic groups G and G_T, with generator g for group G and element g_1 in G. A bilinear map $\hat{e}: G \times G \rightarrow G_T$ is defined, and a random parameter a is

selected, calculating $g_2 = g^a$. The public parameters par include:

$$par = \{G, G_T, g, g_1, g_2, \hat{e}, h, a\}$$

Additionally, the smart contract initializes, and the data owner sets the retrieval price (offer). Users register their accounts with an ID and deposit funds into a blockchain-based deposit account (deposit).

B. Ciphertext Encryption and Upload

The encryption function ($Enc(m, w, k) \rightarrow (C_m, I, N)$) works as follows:

- The plaintext m is encrypted using a symmetric key k, producing ciphertext C_m.
- The key k is included in the index I. A random number r is selected from the prime field Z_p, and the keyword w undergoes hashing, yielding $H(w)$. The index I is calculated as:

$I = \hat{e}(g_1^r, g_2^r) \times \hat{e}(g^r, k) \times H(w)$

- The data owner assigns an identifier N to the ciphertext C_m, then encrypts N using their private key. The system computes a hash of N and C_m, generating H. The final ciphertext CT is a package containing N and H.

The data owner uploads the ciphertext C_m and index I to the cloud server, while the packaged ciphertext CT is sent to the smart contract for future queries.

C. Trapdoor Encryption and Upload

The user generates a trapdoor T_w corresponding to the keyword w in the index:

- The keyword w is hashed to obtain $H(w)$.
- The user selects a random number $t \in Z_p$ and computes:

$$T_w = \{g^t \times H(w), g_1^t\}$$

The trapdoor T_w is uploaded to the smart contract, and the user makes a deposit into the blockchain-based deposit account (deposit).

D. Search Phase

In the search phase, the smart contract interacts with the cloud server to search for data. The smart contract checks the user's ID and verifies if their deposit in the \$deposit account is sufficient for a search.

Once the deposit is verified, the smart contract computes the search using the trapdoor T_w and index I by performing the following steps:

1. Compute the matching function:

$\hat{e}(g_1^r, g_2^t) = \hat{e}(g_1^r, g^a) \times H(w_i)$

If $= w_i$, the result yields the symmetric key k.

2. The smart contract records the file identifier N associated with the ciphertext and continues the search across all files until the relevant results are found.

E. Verification Phase

Once the relevant files are retrieved, the verification process begins:

- The smart contract deducts the retrieval price from the user's deposit (deposit-offer). If the deposit is insufficient for further searches, the system informs the user of insufficient funds.
- If the deposit covers all searches, the smart contract sends the file identifier N and the user's ID to the cloud server. The server then sends the corresponding ciphertext C_m to the user.
- During the interaction between the smart contract and the user, the contract retrieves the ciphertext CT and the symmetric key k and sends them to the user.

The user verifies the consistency of the file identifier:

$$N_{BS} = N_{CS}$$

where N_{BS} is the file identifier from the blockchain system, and N_{CS} is the file identifier from the cloud server. If the identifiers match, it ensures that the cloud server did not send incorrect data. The user then hashes the ciphertext C_m with the file identifier N to verify the integrity of the data:

$$H_1 = h(N, C_m)$$

If $H_1 = H$ (from the encrypted trapdoor), this proves the ciphertext has not been tampered with. Finally, the user decrypts C_m using the symmetric key k, recovering the plaintext m.

F. Random Forest Integration

In the search process, Random Forest is employed to enhance the accuracy and efficiency of keyword matching. The Random Forest algorithm constructs multiple decision trees, where each tree evaluates a subset of keyword indices. The final search result is determined by aggregating the outputs of these decision trees and assigning a score to each keyword match. The tree voting mechanism selects the most relevant data indices, improving retrieval precision. The Random Forest implementation can be described as:

1. Training Phase: The index I is used as training data, where each decision tree classifies whether a keyword is likely to match based on the trapdoor T_w.
2. Prediction Phase: For a given trapdoor T_w, each tree predicts whether the index matches the keyword. The Random Forest combines these results, and the index with the highest score is selected.
3. Equation Update: Each decision tree in the Random Forest operates on the form:

$$\hat{e}(g_1^r, g_2^t) \times H(w_i) = \hat{e}(g_1^r, g^a)$$

The final search result is obtained by voting among the trees in the Random Forest, with the majority result used to retrieve the correct symmetric key k.

V. SECURITY ANALYSIS WITH RANDOM FOREST IMPLEMENTATION

By running the retrieval process on a blockchain system, the following security aspects can be ensured:

A. Fairness

Since every interaction on the blockchain is based on transparent transactions, the results of each query are guaranteed to be correct, and no malicious modification of results can occur. Additionally, since each transaction incurs a fee, malicious users attempting to disrupt the system are deterred. With the implementation of the Random Forest algorithm, fairness is enhanced as the model aggregates results from multiple decision trees to improve keyword matching accuracy, ensuring that the correct data is retrieved.

B. Trustworthiness

The blockchain guarantees that the retrieval results are honest and trustworthy. This serves as a basis for preventing threats from malicious servers. Users can verify the correctness of server operations and retrieve the correct files. The Random Forest method contributes to this by providing more reliable and precise results, as multiple decision trees analyze the keyword indices, thus making the process more robust against server-based threats.

C. Security

This scheme ensures the security of the keywords, as the keyword trapdoors are randomly encrypted, which satisfies the IND-KGA (Indistinguishable Keyword Guess Attack) security requirement. Moreover, the construction of the critical data file index I is based on the hard problem of Decisional Bilinear Diffie-Hellman (DBDH). The security of the ciphertext can be reduced to the hardness of this assumption.

D. IND-KGA Security

Based on general bilinear groups, this scheme is secure under the random oracle model, satisfying IND-KGA security. Here's how the challenge unfolds:

1. Initialization: The challenger C generates random numbers $a, b \in Z_p$ and publishes the public parameters:

$$par = \{ G, G_T, g, g_1, g_2, \hat{e}, h, a\}$$

2. Phase 1: The adversary selects a keyword set $(w_1, w_2, \dots, w_n)$ and sends it to the challenger, who generates the corresponding trapdoor set $(T_{w_1}, T_{w_2}, \dots, T_{w_n})$ and sends them to the adversary.
3. Challenge: The adversary selects two keywords w_0, w_1, ensuring they were not queried in Phase 1. The challenger picks a random number p, runs the trapdoor generation algorithm, and computes:

$$T_{\{w_0\}} = \left(g^p, H(w_0)\right) \text{ and } T_{\{w_1\}} = (g^p, H(w_1))$$

The challenger randomly selects $\mu \in \{0, 1\}$ and sends $T_{\{w_\mu\}}$ to the adversary.

4. Guess: The adversary analyzes the trapdoors from Phases 1 and 2, outputting a guess μ'. If $\mu' = \mu$, the adversary wins the game.

$$\frac{1}{|\Psi|^n} + \epsilon$$

where n is the number of keywords, ϵ is a negligible probability under the security parameter λ, and Ψ is the keyword space.

Theorem 2: DBDH-Based Security

Based on general bilinear groups, the security of this scheme can be reduced to the Decisional Bilinear Diffie-Hellman (DBDH) assumption. If an adversary A can break the scheme in polynomial time, then A can solve the DBDH problem.

1. Initialization: The system generates the security parameter λ and runs the setup algorithm to produce the public parameters:

$$par = \{ G, G_T, g, g_1, g_2, \hat{e}, h, a\}$$

2. Phase 1: The adversary A repeatedly runs the index encryption algorithm.
3. Challenge: The challenger selects two keys k_1 and k_2, ensuring they were not queried in Phase 1. The encryption algorithm is run, and random numbers t are generated to compute:

$$I_1 = \hat{e}(g_1^r, g_2^r) \text{ and } I_2 = \hat{e}(g_1^t, g_2^t)$$

The challenger randomly sends an index I^* to the adversary.

4. Guess: The adversary analyzes the index I^* and outputs a guess I'. If $I' = I^*$, the adversary wins the game.
5. Phase 2: The adversary attempts to break the DBDH assumption by distinguishing two bilinear pairs. The adversary repeatedly runs the algorithm to compute the two quintuplets.
6. Challenge 2: The challenger selects $(a, b, c, z) \in Z_p$, generating two quintuplets:

$$T_0 = \left(g^a, g^b, g^c, g^z, \hat{e}(g, g)\right) \text{ and } T_1 = \left(g^a, g^b, g^c, g^{abc}, \hat{e}(g, g)\right)$$

The challenger randomly selects $\mu \in \{0, 1\}$ and sends T_μ to the adversary.

7. Guess: The adversary analyzes T^* and outputs μ'. If $\mu' = \mu$, the adversary wins.

If the adversary can distinguish the ciphertext index I^*, then they can also distinguish the quintuplets generated by the challenger. Hence, breaking the scheme implies the adversary can solve the DBDH problem. The security of the ciphertext is based on the hardness of the DBDH assumption. In this scheme, Random Forest enhances security by ensuring more

accurate keyword matching during the search phase. The decision trees help identify patterns in trapdoor and index data, which reduces the chances of incorrect retrieval and limits the adversary's ability to exploit vulnerabilities in the keyword matching process.

VI. Experimental Analysis

The experiments were conducted in an environment with a 64-bit Windows operating system, an Intel® Core(TM) i5-4570 CPU running at 3.20 GHz, and 16 GB of memory. The local virtual machine VMware was used to run the open-source project OpenStack for performance testing. The implementation utilized C++ programming language, with cryptographic functions provided by the PBC library.

In this section, the proposed scheme is compared with three other schemes from the literature [12-13, 15]. The comparisons focus on trapdoor generation time, index generation time, and keyword retrieval time. Keywords were tested in increments of 50, starting from 50 and increasing up to 500. Each keyword count was subjected to 50 repeated experiments to calculate the average time overhead, ensuring the validity of the results. Additionally, experiments were conducted to analyze the relationship between string length and time overhead, revealing that data complexity of strings does not affect time overhead. The experiments used 8-letter words as keywords. The dataset used for the experiments was provided by the Natural Language Processing Group at Fudan University's International Database Center. The test corpus consisted of 9,833 documents, and the training corpus comprised 9,804 documents. Furthermore, the performance of the proposed scheme was evaluated both before and after the integration of blockchain technology. Local testing was carried out using the testrpc software.

A. Trapdoor Generation Time

This section compares the proposed scheme with three other methods: DS-PEKS [12], PAEKS [13], and SPE-PP [15]. As shown in table 1, the trapdoor generation time increases with the number of keywords. However, the proposed scheme demonstrates a clear advantage over the other three schemes, particularly as the number of keywords grows. Unlike the other schemes, the trapdoor generation time in the proposed method remains unaffected by the number of characters in a keyword, making it highly efficient for querying complex strings. Additionally, the trapdoor structure is proven to meet IND-KGA security requirements, ensuring the security of the keywords.

Table 1: Trapdoor Generation Time

Method	Trapdoor Generation Time
DS-PEKS [12]	Higher
PAEKS [13]	Higher
SPE-PP [15]	Moderate
Proposed Scheme	Lower

Table 2: Index Generation Time

Method	Index Generation Time
DS-PEKS [12]	Higher
PAEKS [13]	Higher
SPE-PP [15]	Moderate
Proposed Scheme	Lower

B. Index Generation Time

Table 2 shows that the proposed scheme outperforms DS-PEKS and PAEKS significantly and has a slight advantage over SPE-PP. This is mainly due to the fact that the index calculation in the proposed scheme requires only one bilinear pairing and one hash computation, making the construction simpler compared to the other three methods. As the number of keywords increases, the advantage of the proposed scheme becomes even more apparent.

Table 3: Keyword Retrieval Time

Method	Keyword Retrieval Time
DS-PEKS [12]	Higher
PAEKS [13]	Moderate
SPE-PP [15]	Moderate
Proposed Scheme	Lower

C. Keyword Retrieval Time

The proposed scheme performs three bilinear pairing operations during keyword retrieval, resulting in lower computational overhead compared to the other three schemes. As illustrated in Table 3, when the number of keywords reaches 500, the proposed method is approximately 25% more efficient than PAEKS and SPE-PP.

Table 4: Keyword Retrieval Time

Scenario	Retrieval Time	Security
Without Blockchain	Lower	Less Secure
With Blockchain Integration	Higher	More Secure

*D. **Comparison Before and After Blockchain Integration***

The introduction of blockchain increases the retrieval time due to the added security layers, but it also significantly enhances security. As seen in Table 4, the increase in retrieval time due to blockchain integration becomes less pronounced as the number of keywords grows.

References

[1] Dawn S. D., Song D., Wagner A. P., et al. "Practical techniques for searches on encrypted data." Proceedings of the 2000 IEEE Security and Privacy Symposium. Piscataway: IEEE Press, 2000: 44-45.

[2] Boneh D., Di Crescenzo G., Ostrovsky R., et al. "Public key encryption with keyword search." International Conference on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2004: 506-522.

[3] Curtmola R., Garay J., Kamara S., et al. "Searchable symmetric encryption: improved definitions and efficient constructions." Journal of Computer Security, 2011, 19(5): 895-934.

[4] Wang P., Wang H., Pieprzyk J. "Threshold privacy-preserving keyword searches." Conference on Sofsem: Theory & Practice of Computer Science. Berlin: Springer, 2008: 646-658.

[5] Yuan K., Liu Z. L., Jia C. F., et al. "Public key timed-release searchable encryption in one-to-many scenarios." Acta Electronica Sinica, 2015, 43(4): 760-768.

[6] Zhong H., Cui J., Shi R. H., et al. "Many-to-one homomorphic encryption scheme." Security & Communication Networks, 2016, 9(10): 1007-1015.

[7] Tang Q., Chen L. Q. "Public-key encryption with registered keyword search." 6th European Workshop on Public Key Infrastructures. Berlin: Springer, 2009: 163-178.

[8] Fang L. M., Susilo W., Ge C., et al. "Public key encryption with keyword search secure against keyword guessing attacks without random oracle." Information Sciences, 2013, 238: 221-241.

[9] Xu P., Jin H., Wu Q., et al. "Public-key encryption with fuzzy keyword search: a provably secure scheme under keyword guessing attack." IEEE Transactions on Computers, 2013, 62(11): 2266-2277.

[10] Chen R., Mu Y., Yang G., et al. "A new general framework for secure public key encryption with keyword search." Australasian Conference on Information Security and Privacy. Berlin: Springer, 2015: 59-76.

[11] Shao Z. Y., Yang B. "On security against the server in designated tester public key encryption with keyword search." Information Processing Letters, 2015, 115(12): 957-961.

[12] Chen R., Mu Y., Yang G., et al. "Dual-server public-key encryption with keyword search for secure cloud storage." IEEE Transactions on Information Forensics and Security, 2016, 11(4): 789-798.

[13] Huang Q., Li H. "An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks." Information Sciences, 2017, 403-404: 1-14.

[14] Kang Y., Liu Z. "A fully secure verifiable and outsourced decryption ranked searchable encryption scheme supporting synonym query." IEEE Second International Conference on Data Science in Cyberspace. Piscataway: IEEE Press, 2017: 223-231.

[15] Wu L., Chen B., Zeadally S., et al. "An efficient and secure searchable public key encryption scheme with privacy protection for cloud storage." Soft Computing, 2018, 22(23): 7685-7696.

[16] Wu L. B., Zhang Y. B., Ma M. M., et al. "Certificateless searchable public key authenticated encryption with designated tester for cloud-assisted medical Internet of things." Annales des Télécommunications, 2019, 74(7-8): 423-434.

[17] Ma M. M., He D. B., Kumar N., et al. "Certificateless searchable public key encryption scheme for industrial Internet of things." IEEE Transactions on Industrial Informatics, 2018, 14(2): 759-767.

[18] Lu Y., Li J. G. "Efficient searchable public key encryption against keyword guessing attacks for cloud-based EMR systems." Cluster Computing, 2019, 22(1): 285-299.

[19] Li H. G., Zhang F. G., He J. J., et al. "A searchable symmetric encryption scheme using blockchain." CoRR: abs/1711.01030, 2017.

[20] Li H. G., Tian H. B., Zhang F. G., et al. "Blockchain-based searchable symmetric encryption scheme." Computers & Electrical Engineering, 2019, 73: 32-45.

[21] Chen L. X., Lee W. K., Chang C. C., et al. "Blockchain-based searchable encryption for electronic health record sharing." Future Generation Computer Systems, 2019, 95: 420-429.