Security Analysis on a Blockchain-based Public-Key Authenticated Searchable Encryption Scheme

Hinata Nishino[1] and Keita Emura*[1,2]

[1.]Kanazawa University, Japan.
[2.]AIST, Japan.

October 26, 2026

Abstract

Du et al. (Security and Communication Networks, 2022) proposed a public-key authenticated searchable encryption scheme that employs Bloom filters and blockchain. In their scheme, Bloom filters are used to search encrypted keywords, while blockchain is used to guarantee the correctness of search results. In this paper, we demonstrate that Du et al.'s scheme leaks keyword information from ciphertexts. Our analysis focuses on the fact that the Bloom filter embedded in a ciphertext is uniquely determined by the Diffie-Hallman key and the keyword to be encrypted and is directly included in the ciphertext. We show that the success probability of the proposed attack is lower bounded by the probability that no false positives occur in the Bloom filter, and we evaluate this probability to confirm that the attack achieves a sufficiently high success rate. Furthermore, we examine Du et al.'s security model and their assumed usage scenarios, and discuss the validity of our attack under those conditions. We also consider a simple modification intended to prevent our attack and demonstrate that a slightly adapted version of the attack remains effective against the modified scheme. In addition, we show that previously generated ciphertexts can still pass verification even when the search results differ, indicating that the blockchain-based integrity verification mechanism is insufficient.

1 Introduction

Research Background. In cloud computing, the risk of data leakage from cloud servers makes it desirable to encrypt data before storing it in the cloud. While encryption ensures data confidentiality, it also limits the ability to search through the data, making it difficult for users to retrieve the desired information from the cloud server. To address this challenge, Public-Key Encryption with Keyword Search (PEKS) [4] has been proposed. PEKS enables data to remain encrypted while allowing the extraction of information containing specific keywords. In PEKS, a keyword is encrypted using the receiver's public key. During a search, a trapdoor corresponding to the keyword is generated using the receiver's secret key, and a test algorithm is then applied to verify whether the keyword used in the search matches the one associated with the trapdoor.

One major limitation of PEKS is its vulnerability to Keyword Guessing Attacks (KGA). In such an attack, if an adversary obtains a trapdoor, they can encrypt arbitrary keywords and use the test algorithm to determine whether the trapdoor corresponds to a chosen keyword. This results in

*Corresponding Author: k-emura@se.kanazawa-u.ac.jp

leakage of information about the receiver's search queries. From a privacy perspective, resistance to KGA is highly desirable. However, because PEKS allows anyone to encrypt arbitrary keywords, achieving KGA resistance is fundamentally impossible. To overcome this limitation, Public-Key Authenticated Encryption with Keyword Search (PAEKS) has been proposed [16, 18]. In PAEKS, the sender's secret key is incorporated during encryption, preventing adversaries from generating ciphertexts for arbitrary keywords.

PAEKS with Blockchain. Numerous studies have explored the integration of searchable encryption with blockchain technology [2, 5–15, 17, 19, 20, 23, 27–29, 31–40]. Among these, this paper focuses on the blockchain-based PAEKS (BB-PAEKS) scheme proposed by Du et al. [9], which incorporates blockchain to ensure the integrity of search results, which guarantees that the search result is correct. They claimed that the BB-PAEKS schemes offers three key advantages:

1. Resistance to KGA,
2. Significantly reduced computational cost compared to pairing-based PAEKS schemes, making it highly practical, and
3. Integrity of search results, achieved through the tamper-resistance of blockchain.

Our Motivation. Du et al.'s scheme implements search functionality using Bloom filters [3]. In this paper, we focus on the fact that Bloom filters are directly embedded in both the ciphertext and the trapdoor. Since Bloom filters are computed from the hash values of search keywords, the same Bloom filter is generated for identical keywords. Consequently, keyword information may potentially be leaked through the Bloom filter. Additionally, Du et al. claim that by storing a hash value of part of the ciphertext on the blockchain and verifying whether it matches the hash value generated from the ciphertext returned as the search result, the correctness of the search result is guaranteed. Specifically, they state:

> "*Our scheme supports the integrity verification of the results. It ensures the reliability of search results by using the tamper-proof characteristics of blockchain.*"

However, because the hash value can be computed independently of the actual search result, the integrity verification mechanism based on blockchain does not seem to truly guarantee the integrity of the search results.

Our Contribution. In this paper, our contributions are twofold. First, we demonstrate that Du et al.'s scheme leaks keyword information from ciphertexts. against the modified scheme. Second, we point out that the integrity verification mechanism using blockchain is insufficient, i.e., Du et al. scheme does not provide search integrity. Concretely, we demonstrate that the blockchain verification only proves ciphertext registration, not the completeness or correctness of the search results.

- Ciphertext Indistinguishability:
  - We construct an adversary that breaks full ciphertext indistinguishability (CI) by leveraging challenge keywords, specifically those included in the Bloom filter and those that are not.
    - In our proposed attacks, an adversary queries the challenge keyword to the encryption oracle, i.e., we employ "full" CI security. However, Du et al.'s security model, CI security, prohibits querying the challenge keyword to the encryption oracle. Thus, we discuss that the full CI security model to be valid later.

  - The adversary may fail to distinguish challenge keywords when Bloom-filter false positives occur. To address this, we analyze the false positive probability and demonstrate that the success probability of the proposed attack remains sufficiently high.
  - We also consider a simple modification intended to prevent our attack and demonstrate that our attack, with a slight adaptation, remains effective against the modified scheme.

- Search Integrity:
  - We show that even if the search result differs from the correct one, the verification will still succeed as long as the ciphertext was previously generated.
  - Our attack demonstrates that simply storing the hash values of (parts of) searchable ciphertexts on a blockchain and comparing them with the hash values of ciphertexts obtained as search results is not sufficient to guarantee search integrity. Even if the blockchain is assumed to ensure immutability, merely storing data does not necessarily satisfy security requirements from a cryptographic perspective (in this case, search integrity).

Disclaimer. Basically, the searchable encryption component of the Du et al. scheme operates independently of the blockchain, whereas the search integrity component relies on the blockchain. Therefore, the term "blockchain-based PAEKS" may not fully capture the functionality of the Du et al. scheme. Nevertheless, we intentionally adopt this terminology in accordance with the title of the original paper [9].

The primary aim of this paper is to propose an attack against an existing scheme and to discuss the background that led to this attack. Our analysis shows that the underlying security model is cryptographically inappropriate (i.e., fundamentally weak), and that the way blockchain is employed in the scheme is also unsuitable, as it does not contribute to achieving search integrity. These findings indicate that a fundamental redesign of the system is required. Given the severity and breadth of these issues, "fixing" the scheme is far from a matter of minor adjustments. Any meaningful revision would require redefining the system model, the security model, and the construction itself effectively amounting to an entirely new research project and a separate paper. Accordingly, we deliberately do not attempt to repair or redesign the Du et al. scheme in this work. Instead, we assert that identifying these structural problems and articulating why the scheme cannot be salvaged within its current framework constitutes a significant and valuable contribution in its own right.

2 Preliminaries

2.1 Bloom Filter

Bloom filter [3] is a data structure which can identify whether an element is a member of a set. In this paper, we employ the notation provided by Du et al.'s paper [9].

- $\mathsf{BF.Gen}(L, J)$: The Bloom filter generation algorithm takes the Bloom filter size $L \in \mathbb{N}$ and the number of hash functions $J \in \mathbb{N}$ as input. Define $[L] := \{1, 2, \ldots, L\}$ and a collection of hash functions $\mathcal{H} = \{h_j\}_{j\in[J]}$, where h_j takes a message $\mathsf{Meg} \in \{0,1\}^*$ as input, k as a key, and outputs $i \in [L]$. For each $i \in [L]$, denote the i-th element $\mathsf{BF}[i] \in \{0,1\}$ of the Bloom filter $\mathsf{BF} \in \{0,1\}^L$. Initialize $\mathsf{BF}[i] = 0$ for all i, then output $\mathcal{H}$ and BF.

- $\mathsf{BF.Upd}(\mathcal{H}, \mathsf{BF}, \mathsf{Meg})$: The Bloom filter update algorithm takes $\mathcal{H} = \{h_j\}_{j\in[J]}$, $\mathsf{BF} \in \{0,1\}^L$, and a message $\mathsf{Meg} \in \{0,1\}^*$ as input. It outputs a Bloom filter BF such that for all $j \in [J]$, $\mathsf{BF}[h_j(\mathsf{Meg}, k)] \leftarrow 1$.
- $\mathsf{BF.Check}(\mathcal{H}, \mathsf{BF}, \mathsf{Meg})$: The Bloom filter checking algorithm takes $\mathcal{H} = \{h_j\}_{j\in[J]}$, BF, Meg as input. It outputs 1 if $\mathsf{BF}[h_j(\mathsf{Meg}, k)] = 1$ holds for all $j \in [J]$, and 0 otherwise.

We note that the $\mathsf{BF.Upd}$ algorithm and the $\mathsf{BF.Check}$ algorithm do not explicitly take k as input. We follow the syntax given by Du et al. and assume that information of k is included in $\mathcal{H}$.

Bloom filters may produce false positives, where data not present in the set is incorrectly identified as present. The false positive probability is expressed as follows, where J is the number of hash functions and K is the number of messages inserted into the filter [24].

$$\Pr[\mathsf{FalsePositive}] = (1 - (1 - 1/L)^{JK})^J$$

2.2 Inverted Index

An inverted index is a data structure that records the positional information of keywords within a dataset. Du et al. [9] provide the following example of an index matrix representing the relationship between data and keywords.

Table 1: Index Matrix [9]

ID	Data	Keyword
1	d_1	w_1, w_2, w_4, w_5
2	d_2	w_1, w_3
3	d_3	w_1, w_2, w_4
4	d_4	w_1, w_2, w_3, w_4, w_5
5	d_5	w_3, w_4

As shown in Figure 1, Du et al. [9] assume that the same keywords may appear in multiple datasets. For example, w_1 contained in d_1 also appears in d_2, d_3, and d_4. An inverted index indicates which documents contain each keyword. For the above example, this corresponds to Table 2. Following the notation of Manning et al. [22], we use the terms Dictionary and Postings.

Table 2: Example of Inverted Index [9]

Dictionary	Postings
w_1	$1, 2, 3, 4$
w_2	$1, 3, 4$
w_3	$4, 5$
w_4	$1, 3, 4, 5$
w_5	$1, 4$

Combination of Inverted Index and PEKS: Encrypt the data using any standard encryption method, and encrypt the keywords using a searchable encryption scheme. The Dictionary column in Table 2 stores the ciphertexts of the keywords. Each encrypted data item retains the same ID as its original data. When a search query matches a keyword, the system returns the encrypted data associated with the corresponding ID according to the Postings column.

Information Leakage via Inverted Index. Table 2 suggests an interpretation in which each keyword is encrypted only once. Under this interpretation, the fact that data items 1, 2, 3, and 4 share the same keyword is intentionally revealed through the index structure. However, learning that these data items contain the same keyword without issuing any search query is fundamentally different from learning it through legitimate search results. In searchable encryption, some leakage arising from search results is typically tolerated, whereas keyword-related information is expected to remain hidden before a search is performed. Moreover, as mentioned in the previous paragraph, such leakage can be exploited to break the indistinguishability of the underlying encryption scheme combined with P(A)EKS. For example, suppose that an adversary specifies data items 2 and 4 as challenge plaintexts. Then, even if the underlying encryption scheme provides a strong security notion such as IND-CCA security, the index structure can be used to distinguish whether the challenge ciphertext corresponds to data item 2 or data item 4. Therefore, a straightforward integration of an inverted index with searchable encryption may reveal unintended information.

Alternatively, one may associate searchable ciphertexts with individual data items. In the data structure shown in Table 1, each keyword contained in each data item is encrypted independently. Consequently, the same keyword may be encrypted multiple times under the same public key. Since PEKS is designed to hide keyword information even when the same keyword is encrypted repeatedly, it is natural to consider such a setting when evaluating the privacy of searchable encryption schemes. Under this interpretation, whether multiple data items contain the same keyword is not revealed prior to search. Motivated by this distinction, this paper considers the scenario where a keyword may be encrypted multiple times.

3 Definition of BB-PAEKS

In this section, we present the syntax of BB-PAEKS and introduce the full CI security (Ciphertext Indistinguishability), which is the primary focus of this paper. The participants are the Data Sender (DS), Data Receiver (DR), Service Provider (SP), and the blockchain.

Definition 1 (Syntax of the BB-PAEKS)**.**

- $\mathsf{Setup}(1^\lambda)$*: The setup algorithm takes security parameter* λ *as input, and outputs the public parameter* pp*.*
- $\mathsf{KeyGen}_{\mathsf{DS}}(\mathsf{pp})$*: The data sender's key generation algorithm takes* pp *as input, and outputs a public-secret key pair* $(\mathsf{pk}_{\mathsf{DS}}, \mathsf{sk}_{\mathsf{DS}})$*.*
- $\mathsf{KeyGen}_{\mathsf{DR}}(\mathsf{pp})$*: The data receiver's key generation algorithm takes* pp *as input, and outputs a public-secret key pair* $(\mathsf{pk}_{\mathsf{DR}}, \mathsf{sk}_{\mathsf{DR}})$*.*
- $\mathsf{PAEKS}(\mathsf{pk}_{DR}, \mathsf{sk}_{DS}, w)$*: The encryption algorithm takes* pk_{DR}*,* sk_{DS}*, and a keyword* w *as input, and outputs the ciphertext of the keyword, denoted as* C_w*.*
- $\mathsf{Trapdoor}(\mathsf{pk}_{\mathsf{DS}}, \mathsf{sk}_{\mathsf{DR}}, w)$*: The trapdoor generation algorithm takes* $\mathsf{pk}_{\mathsf{DS}}$*,* $\mathsf{sk}_{\mathsf{DR}}$*, and a keyword* w' *as input, and outputs a corresponding trapdoor* $T_{w'}$*.*
- $\mathsf{Search}(C_w, T_w)$*: The search algorithm takes* C_w *and* T_w *as input, and outputs 0 or 1.*
- $\mathsf{Verify}(\mathsf{Rlt}, \mathsf{Acc})$*: The search result verification algorithm takes a set of search result* Rlt *and a checklist* Acc *as input, and outputs a proof* $\mathsf{proof} \in \{\mathsf{true}, \mathsf{false}\}$*.*

- $\mathsf{Dec}(\mathsf{Rlt}, \mathsf{sk}_{\mathsf{DR}})$*: The decryption algorithm takes* Rlt *and* $\mathsf{sk}_{\mathsf{DR}}$ *as input, and outputs the ID* ind_w *associated with the keyword* w*.*

- $\mathsf{Update}(w, k)$*: The update algorithm takes* w *and* k *as input, and outputs updated ciphertext* C_w*.*

Note that k, the input to the update algorithm, is a value computed within the PAEKS algorithm or the $\mathsf{Trapdoor}$ algorithm. In the actual scheme, it corresponds to the key used in Bloom filters.

Here is a brief overview of the BB-PAEKS workflow. First, DS encrypts the data and the keyword and sends them to SP, then submits the checklist Acc to the blockchain. DR generates a trapdoor for the keyword and sends it to SP. Using the trapdoor, SP searches for data containing the associated keywords and returns the corresponding ciphertext to both DR and the blockchain. The blockchain then generates proof from the search results and the checklist Acc provided by DS, and sends proof to DR. Note that Du et al. [9] mention sending the trapdoor to the blockchain[1]. However, since the blockchain does not use the trapdoor when creating proof for the search results, this step is omitted in our description.

Full CI Security. We introduce full CI security as follows. The difference from the definition given in [9] is that an adversary is allowed to send the challenge keywords to the encryption oracle. Let $\mathcal{C}$ be the challenger and $\mathcal{A}$ be an adversary.

Definition 2 (Full CI Security)**.**

- *Initialization:* $\mathcal{C}$ *runs* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{pk}_{\mathsf{DS}}, \mathsf{sk}_{\mathsf{DS}}) \leftarrow \mathsf{KeyGen}_{\mathsf{DS}}(\mathsf{pp})$, $(\mathsf{pk}_{\mathsf{DR}}, \mathsf{sk}_{\mathsf{DR}}) \leftarrow \mathsf{KeyGen}_{\mathsf{DR}}(\mathsf{pp})$*, and sends* $(\mathsf{pp}, \mathsf{pk}_{\mathsf{DS}}, \mathsf{pk}_{\mathsf{DR}})$ *to* $\mathcal{A}$*.*

- *Phase 1:* $\mathcal{A}$ *queries a keyword* w *to the encryption oracle and the trapdoor oracle, obtaining the ciphertext and the trapdoor respectively.*

- *Challenge:* $\mathcal{A}$ *declares challenge keywords* w_0^* *and* w_1^* *that were not sent to the trapdoor oracle.* $\mathcal{C}$ *randomly selects* $b \in \{0, 1\}$ *and sends the challenge ciphertext* $Cw_b{}^*$*.*

- *Phase 2:* $\mathcal{A}$ *queries a keyword* $w \notin \{w_0^*, w_1^*\}$ *to the trapdoor oracle, obtaining the ciphertext and the trapdoor respectively.*

- *Guess:* $\mathcal{A}$ *outputs* $b' \in \{0, 1\}$*. If* $b' = b$*, then* $\mathcal{A}$ *wins.*

$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Full\text{-}CI}}(\lambda) = |\Pr[b' = b] - \frac{1}{2}|$ *is defined as the advantage of* $\mathcal{A}$ *in this game. For a given security parameter* λ*, if* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Full\text{-}CI}}(\lambda)$ *is negligible, the BB-PAEKS scheme is defined as full CI-secure against Chosen Keyword Attack (CKA).*

Trapdoor Indistinguishability (TI) security, which ensures that no information about the keywords is leaked through the trapdoor, is also defined but omitted in this paper. Moreover, Du et al.'s work does not formally address the search integrity of the returned results. As mentioned before, they claimed that the integrity ensures the reliability of search results. Based on this description, we assume that the Verify algorithm outputs true if the search result is correct and false otherwise.

[1]Specifically, they state: "*The* DR *sends trapdoor* T_w *to the* SP *and Blockchain*".

4 BB-PAEKS

Du et al. essentially provide only the scheme and do not mention the underlying methodology. For improved readability, we first introduce the construction methodology as fully as possible as follows before the Du et al. scheme is given. Du et al. implicitly employ the Diffie-Hellman (DH) non-interactive key exchange protocol between DS and DR.

$$\mathsf{DH}_{\mathsf{Key}} = \mathsf{pk}_{\mathsf{DR}}^{\mathsf{sk}_{\mathsf{DS}}} = \mathsf{pk}_{\mathsf{DS}}^{\mathsf{sk}_{\mathsf{DR}}} = g^{xy}$$

When generating the Bloom filter $\mathsf{BF}_{\mathsf{DS}}$ contained in a ciphertext and the Bloom filter $\mathsf{BF}_{\mathsf{DR}}$ contained in a trapdoor, a hash function is derived from $\mathsf{DH}_{\mathsf{Key}}$ and the keyword w. Specifically, the set of hash functions $\mathcal{H} = \{h_j(k, w)\}$ is computed using the key $k = \mathcal{H}_1(\mathsf{DH}_{\mathsf{Key}}, w)$. Furthermore, when generating $\mathsf{BF}_{\mathsf{DR}}$, multiple random keywords $\mathsf{num}_1, \ldots, \mathsf{num}_J$ are inserted into the filter in addition to the actual keyword. Although Du et al. do not provide an explicit reason for this step, we assume that it is intended to prevent leakage of keyword information from the trapdoor. During the search process, if the following condition holds, the search keyword is considered to be contained in $\mathsf{BF}_{\mathsf{DS}}$, where $\langle \mathsf{BF}_{\mathsf{DS}}, \mathsf{BF}_{\mathsf{DR}} \rangle$ denotes the inner product of $\mathsf{BF}_{\mathsf{DS}}$ and $\mathsf{BF}_{\mathsf{DR}}$, and $|\mathsf{BF}_{\mathsf{DS}}|$ represents the number of bits set to 1 in $\mathsf{BF}_{\mathsf{DS}}$.

$$\langle \mathsf{BF}_{\mathsf{DS}}, \mathsf{BF}_{\mathsf{DR}} \rangle = |\mathsf{BF}_{\mathsf{DS}}|$$

Although Du et al. do not explicitly mention this, false positives may occur, and searches may fail depending on the random keywords inserted into $\mathsf{BF}_{\mathsf{DR}}$. Specifically, even if different keywords are inserted into $\mathsf{BF}_{\mathsf{DS}}$ and $\mathsf{BF}_{\mathsf{DR}}$, a search hit may still occur if a random keyword added to $\mathsf{BF}_{\mathsf{DR}}$ sets a bit that coincides with a bit set in $\mathsf{BF}_{\mathsf{DS}}$. We emphasize that this issue is independent of our attacks.

The identifier ind_w associated with data in the inverted index is encrypted using ElGamal encryption with $\mathsf{pk}_{\mathsf{DR}} = g^x$ as the public key and sent to SP as follows:

$$C_1 = g^{r_1}, C_2 = g^{xr_1} \cdot \mathsf{ind}_w$$

For example, w_1 is contained in d_1, d_2, d_3, and d_4 in Figure 2. Then, $\mathsf{ind}_{w_1} = \{1, 2, 3, 4\}$. Usually, a plaintext of the ElGamal encryption needs to be a group element, Du et al. did not mention how to encrypt ind_w. Here, we assume that the standard hybrid encryption technique is employed where a plaintext of the ElGamal encryption is a secret key and the actual plaintext is encrypted by a secret key encryption scheme (e.g., AES) using the secret key. We do not further discuss this point since it is independent of our attacks.

The hash value $C_3 = \mathcal{H}_2(C_2)$ of C_2 is sent to the blockchain. When SP submits the search result, its integrity is verified by comparing the hash value of C_2 contained in the result with C_3. More precisely, in the Verify algorithm, the original scheme introduced a set of encrypted file identifiers, denoted as B. However, its role is not clearly defined. In particular, the Verify algorithm takes Rlt and Acc as input and outputs true if the hash values, computed from parts of the ciphertexts contained in Rlt, are included in Acc. Since this procedure does not depend on B, we omit B from the following description. We emphasize that omitting B is independent of our attacks. The detailed procedure is as follows.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: Choose a cyclic group $\mathbb{G}_1$ of order p. Let J be the number of hash functions. Define three hash functions $\mathcal{H}_1 : \mathbb{G}_1 \times \{0,1\}^* \to \mathbb{Z}_p^*$, $\mathcal{H}_2 : \{0,1\}^* \to \{0,1\}^*$, $\mathcal{H}_3 : \mathbb{Z}_p^* \times \{0,1\}^* \to \mathbb{Z}_p^*$, and k as the key, defining the hash function family $\mathcal{H} = \{h_j(k, \mathsf{Meg})\}_{j \in [J]}$. Select a random generator g for $\mathbb{G}_1$. Output the public parameters $\mathsf{pp} = \{\mathbb{G}, p, g, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{H}\}$.

- $(\mathsf{pk}_{\mathsf{DS}}, \mathsf{sk}_{\mathsf{DS}}) \leftarrow \mathsf{KeyGen}_{\mathsf{DS}}(\mathsf{pp})$: DS randomly selects $y \in \mathbb{Z}_p^*$, generates $\mathsf{pk}_{\mathsf{DS}} = g^y$, and sets $\mathsf{sk}_{\mathsf{DS}} = y$.

- $(\mathsf{pk}_{\mathsf{DR}}, \mathsf{sk}_{\mathsf{DR}}) \leftarrow \mathsf{KeyGen}_{\mathsf{DR}}(\mathsf{pp})$: DR randomly selects $x \in \mathbb{Z}_p^*$, generates $\mathsf{pk}_{\mathsf{DR}} = g^x$, and sets $\mathsf{sk}_{\mathsf{DR}} = x$.

- $(C_{w,1}, C_{w,2}) \leftarrow \mathsf{PAEKS}(\mathsf{sk}_{\mathsf{DS}}, \mathsf{pk}_{\mathsf{DR}}, w)$: DS selects version serial number $\mathsf{VerInfo} \in \mathbb{Z}_p^*$. DS executes $\mathsf{BF.Gen}(L, J)$ and initializes the Bloom filter $\mathsf{BF}_{\mathsf{DS}}$ on the DS side. DS generates $\mathsf{DH}_{\mathsf{Key}} = \mathsf{pk}_{\mathsf{DR}}^{\mathsf{sk}_{\mathsf{DS}}}$ and key $k = \mathcal{H}_1(\mathsf{DH}_{\mathsf{Key}}, w)$. Using k and a keyword w, DS executes $\mathsf{BF.Upd}(\mathcal{H}, \mathsf{BF}_{\mathsf{DS}}, w)$ and outputs $\mathsf{BF}_{\mathsf{DS}}$. DS randomly selects $r_1 \in \mathbb{Z}_p^*$. For $\mathsf{ind}_w = \{id_i | w \in id_i\}$, DS computes $C_1 = g^{r_1}$, $C_2 = g^{xr_1} \cdot \mathsf{ind}_w$, $C_3 = \mathcal{H}_2(C_2)$. DS sends $C_{w,1} = \{\mathsf{BF}_{\mathsf{DS}}, C_1, C_2, \mathsf{VerInfo}\}$ to the SP and $C_{w,2} = \{C_3, \mathsf{VerInfo}\}$ to the blockchain. SP adds $C_{w,1}$ to the encrypted keyword set EDB, and the blockchain adds $C_{w,2}$ to Acc.

- $T_w \leftarrow \mathsf{Trapdoor}(\mathsf{sk}_{\mathsf{DR}}, \mathsf{pk}_{\mathsf{DS}}, w)$: DR executes $\mathsf{BF.Gen}(L, J)$ and initializes the Bloom filter $\mathsf{BF}_{\mathsf{DR}}$ on the DR side. DR computes $\mathsf{DH}_{\mathsf{Key}} = \mathsf{pk}_{\mathsf{DS}}^{\mathsf{sk}_{\mathsf{DR}}}$ and $k = \mathcal{H}_1(\mathsf{DH}_{\mathsf{Key}}, w)$. DR executes $\mathsf{BF.Upd}(\mathcal{H}, \mathsf{BF}_{\mathsf{DR}}, w)$ and outputs $\mathsf{BF}_{\mathsf{DR}}$. DR selects J random keywords $\mathsf{num}_1, \ldots, \mathsf{num}_J \in \{0,1\}^*$ and executes $\mathsf{BF.Upd}(\mathcal{H}, \mathsf{BF}_{\mathsf{DR}}, \mathsf{num}_\ell)$ for all $\ell \in [J]$. DR sends $T_w = \mathsf{BF}_{\mathsf{DR}}$ to SP.

- $\{0,1\} \leftarrow \mathsf{Search}(C_{w,1}, T_w)$: SP computes $\langle \mathsf{BF}_{\mathsf{DS}}, \mathsf{BF}_{\mathsf{DR}} \rangle$ and $|\mathsf{BF}_{\mathsf{DS}}|$. If $\langle \mathsf{BF}_{\mathsf{DS}}, \mathsf{BF}_{\mathsf{DR}} \rangle = |\mathsf{BF}_{\mathsf{DS}}|$ holds, it outputs 1; otherwise, it outputs 0. SP adds the matching ciphertext $C_{w,1}$ to the search result list Rlt and sends it to DR and the blockchain.

- $\mathsf{proof} \leftarrow \mathsf{Verify}(\mathsf{Rlt}, \mathsf{Acc})$: For all $C_{w,1} \in \mathsf{Rlt}$ with the form $C_{w,1} = \{\mathsf{BF}_{\mathsf{DS}}, C_1, C_2, \mathsf{VerInfo}\}$, it calculates the hash value $\mathcal{H}_2(C_2)$ of C_2 and checks it is included in Acc. If these hash values are included in Acc, output $\mathsf{proof} = \mathsf{true}$, otherwise $\mathsf{proof} = \mathsf{false}$. The verification result $\mathsf{proof} \in \{\mathsf{true}, \mathsf{false}\}$ and Rlt are then sent to DR.

- $\mathsf{ind}_w \leftarrow \mathsf{Dec}(\mathsf{Rlt}, \mathsf{sk}_{\mathsf{DR}})$: DR computes $\mathsf{ind}_w = C_2 / C_1^x$.

- $(C_{w,1}, C_{w,2}) \leftarrow \mathsf{Update}(w, k)$: DR selects version serial number $\mathsf{VerInfo}^* \in \mathbb{Z}_p^*$ and DS executes $\mathsf{BF.Gen}(L, J)$. DR computes $\mathsf{DH}_{\mathsf{Key}} = \mathsf{pk}_{\mathsf{DS}}^{\mathsf{sk}_{\mathsf{DR}}}$, $k_1 = \mathcal{H}_1(\mathsf{DH}_{\mathsf{Key}}, w)$, and $k_2 = \mathcal{H}_3(k_1, w)$. Using k_2 and keyword w, DR executes $\mathsf{BF.Upd}(\mathcal{H}, \mathsf{BF}_{\mathsf{DS}}, w)$ and outputs $\mathsf{BF}_{\mathsf{DS}}$. For $\mathsf{ind}_w = \{id_i | w \in id_i\}$, DR randomly selects $r_1 \in \mathbb{Z}_p^*$ and computes $C_1 = g^{r_1}$, $C_2 = g^{xr_1} \cdot \mathsf{ind}_w$, $C_3 = \mathcal{H}_2(C_2)$, and sends $C_{w,1} = \{\mathsf{BF}_{\mathsf{DS}}, C_1, C_2, \mathsf{VerInfo}^*\}$ to SP and $C_{w,2} = \{C_3, \mathsf{VerInfo}^*\}$ to the blockchain. SP adds $C_{w,1}$ to the encrypted keyword set EDB, and the blockchain adds $C_{w,2}$ to Acc.

5 Proposed Attacks

5.1 First Attack: For Full CI security

In the first attack, we exploit the fact that the Bloom filter $\mathsf{BF}_{\mathsf{DS}}$ contained in the ciphertext is uniquely determined by the DH key and the keyword exchanged between DS and DR. Algorithm 1 presents the attack procedure. The flow is summarized as follows: $\mathcal{A}$ queries the challenge keyword w_0^* to the encryption oracle and obtains the ciphertext $C_{w_0^*,1} = \{\mathsf{BF}'_{\mathsf{DS}}, C'_1, C'_2, \mathsf{VerInfo}'\}$. Then, $\mathcal{A}$ compares the $\mathsf{BF}_{\mathsf{DS}}$ in the challenge ciphertext $C_{w_b^*,1}$ with the $\mathsf{BF}_{\mathsf{DS}}$ in $C_{w_0^*,1}$. If they match, it outputs $b' = 0$; otherwise, it outputs $b' = 1$.

Algorithm 1 First Attack: for Full CI security

Input: $C_{w_b^*}$ where $C_{w_b^*} = (C_{w_b^*,1}, C_{w_b^*,2})$ and $C_{w_b^*,1} = \{\mathsf{BF}^*_{\mathsf{DS}}, \cdot, \cdot, \cdot\}$ and w_0^*
Output: b'
  $C_{w_0^*} \leftarrow$ EncQuery w_0^* where $C_{w_0^*} = (C_{w_0^*,1}, C_{w_0^*,2})$ and $C_{w_0^*,1} = \{\mathsf{BF}'_{\mathsf{DS}}, \cdot, \cdot, \cdot\}$
  Extract $\mathsf{BF}'_{\mathsf{DS}}$ from $C_{w_0^*}$
  Extract $\mathsf{BF}^*_{\mathsf{DS}}$ from $C_{w_b^*}$
  if $\mathsf{BF}^*_{\mathsf{DS}} == \mathsf{BF}'_{\mathsf{DS}}$ **then return** $b' = 0$
  else return $b' = 1$
  end if

The adversary fails only if the Bloom filters corresponding to w_0^* and w_1^* become indistinguishable. This event occurs with probability at most the probability that a false positive occurs in the Bloom filter. Thus, the success probability of the attack is lower bounded by the probability that no false positives occur. To evaluate this probability, we refer to the paper cited by Du et al. [30]. Given the number K of keywords inserted into the filter and the target false positive probability $\Pr[\mathsf{FalsePositive}]$, the filter size L is expressed as follows.

$$L = -K \times \mathsf{ln}(\Pr[\mathsf{FalsePositive}])/\mathsf{ln}(2)^2$$

In the BB-PAEKS scheme, $K = 1$ because a new Bloom filter is generated during encryption. Considering 128-bit security, $\Pr[\mathsf{FalsePositive}] = 2^{-128}$ is set as the (tentative) target value. Using these values, the filter size becomes $L = 185$. For this derived L, the optimal number J of hash functions that achieves the target false positive probability $\Pr[\mathsf{FalsePositive}]$ is given as follows.

$$J = \lceil L \times \mathsf{ln}(2)/K \rceil$$

For $L = 185$, the optimal number of hash functions is $J = 129$. Note that, the target false positive probability $\Pr[\mathsf{FalsePositive}] = 2^{-128}$ may be extremely low in practical settings. For example, [30] assumes a very large security parameter, setting $\Pr[\mathsf{FalsePositive}] = 10^{-4} \approx 2^{-13}$. Using these values, the filter size becomes $L = 20$ and the optimal number of hash functions is $J = 14$. Table 3 shows the degradation of $\Pr[\mathsf{FalsePositive}]$ when J is reduced from the optimal value of 129 to 14.

Table 3: Evaluation of false positive rate

L	J	$\Pr[\mathsf{FalsePositive}]$
20	14	$6.7 \times 10^{-5} \approx 2^{-14}$
185	129	$2.9 \times 10^{-39} \approx 2^{-128}$

Note that the parameter set $(L, J) = (20, 14)$ yields a false positive probability of $\Pr[\mathsf{FalsePositive}] \approx 6.7 \times 10^{-5} \approx 2^{-14}$, which is consistent with the practical target value $\Pr[\mathsf{FalsePositive}] = 10^{-4} \approx 2^{-13}$ adopted in [30]. Therefore, the attack is evaluated under realistic Bloom-filter parameters rather than artificially weakened settings.

5.2 Second Attack: For Integrity

Acc only stores the hash value C'_2 of the ciphertext generated by DS. Even if SP returns an incorrect ciphertext, the Verify algorithm still outputs true, unless DS never registered that ciphertext on the blockchain. This shows that the verification process depends solely on blockchain registration, not

on the correctness of the search result. Consequently, Du et al.'s scheme does not ensure search integrity.

Although the proposed integrity attack is technically simple, the fact that it has remained unnoticed is itself noteworthy. The attack highlights a fundamental issue: merely storing a value on a blockchain does not, in itself, provide evidence that the corresponding cryptographic computation has been performed correctly. Given the growing number of cryptographic protocols that incorporate blockchain technology, this observation serves as an important reminder that blockchain records should not be regarded as proof of correct cryptographic computation. Therefore, the significance of the attack extends beyond the specific scheme analyzed in this paper and highlights a more general design pitfall in blockchain-based cryptographic protocols.

Limitations of the Integrity Verification Model. Du et al. state that integrity verification is necessary in scenarios where SP provides services with a risk of ciphertext tampering. Specifically, they note

> "*Third-party servers often benefit in practice from providing retrieval services with a risk that ciphertext data may be tampered with by malicious parties or that the server may fail. As a consequence, the integrity of search results is also a concern that cannot be overlooked.*".

However, as mentioned earlier, the Verify algorithm outputs false only when DS passes the ciphertext to the SP but does not pass it to the blockchain. Note that by decrypting the ciphertext in the search results, DR can easily verify whether the keyword is included in the SP's output. On the other hand, if the SP fails to return encrypted data that should be included in the search results, then DR has no means of detecting this omission. Similarly, since the model relies on the blockchain to generate proofs based solely on the SP's output, the blockchain itself cannot detect such behavior either. Therefore, we conclude that the model warrants reconsideration.

6 Simple Countermeasure and its Limitation

The first attack exploits the fact that the Bloom filter $\mathsf{BF}_{\mathsf{DS}}$ embedded in the ciphertext is uniquely determined by the keyword and the DH key which is non-interactively shared between DS and DR. Note that Du et al.'s scheme updates the Bloom filter $\mathsf{BF}_{\mathsf{DR}}$ contained in the trapdoor using random keywords $\mathsf{num}_1, \ldots, \mathsf{num}_J$. Specifically, for all $\ell \in [J]$, the algorithm executes $\mathsf{BF.Upd}(\mathcal{H}, \mathsf{BF}_{\mathsf{DR}}, \mathsf{num}_\ell)$ to obtain $\mathsf{BF}_{\mathsf{DR}}$, making $\mathsf{BF}_{\mathsf{DR}}$ probabilistically determined. If $\mathsf{BF}_{\mathsf{DS}}$ were updated with random keywords instead of $\mathsf{BF}_{\mathsf{DR}}$, even ciphertexts with the same keyword would yield different $\mathsf{BF}_{\mathsf{DS}}$ values, and the first attack could not be directly applied. However, the modified scheme (hereinafter referred to as the modified scheme) still leaks keyword information from the ciphertext. Focusing on the $\mathsf{BF.Upd}$ algorithm, we observe that bits set to 1 in $\mathsf{BF}_{\mathsf{DS}}$ before the update always remain 1 after the update. Thus, when multiple ciphertexts encrypted with the same keyword are received, by setting bits that are 1 in all ciphertexts to 1 and bits that are 0 in any ciphertext to 0, it becomes possible to probabilistically reconstruct the pre-update $\mathsf{BF}_{\mathsf{DS}}$. Algorithm 2 presents an attack against the modified scheme. Let L denote the filter size, J the number of hash functions, q_{enc} the number of queries to the ciphertext oracle, and w_0^* the challenge keyword.

6.1 Probability Evaluation

Below, we evaluate the probability of recovering $\mathsf{BF}_{\mathsf{DS}}$ before the updates. For this evaluation, we adopt the same assumption as [24], cited by Du et al., that Bloom-filter hash outputs are

Algorithm 2 Modified Attack: against the Modified Scheme

Input: $L, J, q_{\mathsf{enc}}, w_0^*$

Output: $\mathsf{BF_{DS}}$ $\mathsf{BF_{DS}}[i] \leftarrow 1$ for all $i \in [L]$ $counter \leftarrow 0$

```
while counter < q_enc do
    counter = counter + 1
    C_{w_0^*} ← EncQuery w_0^* where C_{w_0^*} = (C_{w_0^*,1}, C_{w_0^*,2}) and C_{w_0^*,1} = {queryBF_DS, ·, ·, ·}
    Extract queryBF_DS from C_{w_0^*}
    for i = 1 to L do
        if queryBF_DS[i] == 0 then
            BF_DS[i] ← 0
        end if
    end for
end while
return BF_DS
```

independently and uniformly distributed over the L positions. Under this model, collisions among hash outputs may occur, and thus fewer than J distinct bits may be set by a single insertion. Accordingly, our analysis does not assume collision-free insertions. Let BitPos be a bit position that is 0 in the original $\mathsf{BF_{DS}}$. When a random keyword is inserted using J hash functions, the probability that BitPos is set to 1 is

$$\Pr[1 \leftarrow 0] = 1 - \left(1 - \frac{1}{L}\right)^J.$$

Indeed, each hash output is independently and uniformly distributed over the L positions of the Bloom filter. Therefore, the probability that none of the J hash outputs is mapped to BitPos is

$$\left(1 - \frac{1}{L}\right)^J,$$

yielding the above expression. In the modified attack, each $\mathsf{queryBF_{DS}}$ is generated by inserting J random keywords. Thus, the probability that BitPos is set to 1 in a single observed $\mathsf{queryBF_{DS}}$ is

$$p = 1 - \left(1 - \frac{1}{L}\right)^{J^2}.$$

We note that the modified scheme inserts J random keywords, where J is chosen to be equal to the number of hash functions. Therefore, a total of J^2 hash evaluations are performed. The attack fails to recover BitPos if BitPos is set to 1 in every one of the q_{enc} observed $\mathsf{queryBF_{DS}}$. Therefore, the probability of failing to recover BitPos is

$$\Pr[\mathsf{Fail}_{\mathsf{BitPos}}] = p^{q_{\mathsf{enc}}} = \left(1 - \left(1 - \frac{1}{L}\right)^{J^2}\right)^{q_{\mathsf{enc}}}.$$

Conversely, the probability of successfully recovering BitPos is

$$\Pr[\mathsf{Success}_{\mathsf{BitPos}}] = 1 - p^{q_{\mathsf{enc}}}.$$

Let N_0 denote the number of bit positions that are 0 in the original $\mathsf{BF_{DS}}$. Since collisions among hash outputs may occur, the number of 0-bit positions satisfies

$$N_0 \geq L - J.$$

Since a smaller value of N_0 yields a higher attack success probability, we evaluate the attack using the most favorable case for the adversary, namely $N_0 = L - J$. We assume that the recovery events of distinct 0-bit positions are mutually independent. Although hash collisions may introduce weak dependencies among bit positions, this assumption is commonly used in Bloom-filter analyses and provides a practical approximation. Under this assumption, the probability of recovering all 0-bit positions is

$$\Pr[\mathsf{Success}] = (1 - p^{q_{\mathsf{enc}}})^{N_0}.$$

Substituting the expression for p yields

$$\Pr[\mathsf{Success}] = \left(1 - \left(1 - \left(1 - \frac{1}{L}\right)^{J^2}\right)^{q_{\mathsf{enc}}}\right)^{N_0}.$$

For practical Bloom-filter parameters, Du et al. cite the setting $\Pr[\mathsf{FalsePositive}] = 10^{-4}$ from [30]. Using the standard Bloom-filter design equations with $K = 1$, this corresponds approximately to $L = 20$ and $J = 14$. Since collisions among hash outputs may occur, the number of 0-bit positions satisfies

$$N_0 \geq L - J.$$

Because the attack success probability decreases as N_0 increases, we evaluate the attack under the most favorable condition for the adversary, namely $N_0 = L - J = 6$. Table 4 presents the resulting attack success probability. The success probability exceeds 0.97 for $q_{\mathsf{enc}} = 2^{17}$. Therefore, the modified attack succeeds with high probability after a practical number of encryption queries.

Table 4: Evaluation of $\Pr[\mathsf{Success}]$ ($L = 20, J = 14, N_0 = 6$)

q_{enc}	$\Pr[\mathsf{Success}]$
2^{13}	0.00069
2^{14}	0.01677
2^{15}	0.18656
2^{16}	0.69172
2^{17}	0.97889

7 Conclusion

In this paper, we show that Du et al.'s scheme leaks keyword information through the Bloom filter embedded in the ciphertext. Although simple modifications can mitigate this attack, we also provide an effective variant against the modified scheme. Because these attacks lie outside Du et al.'s security model, we examine the validity of the assumed environment. We further analyze the integrity verification and point out that the blockchain-based proof mechanism warrants reconsideration.

Numerous attack methods against Bloom filters have been reported. Consider an adversary who outputs elements with a non-negligible probability of generating false positives. Existing results address adaptive adversaries who can select the next query after observing whether a given value is included in the Bloom filter [21, 25, 26]. Since our model allows obtaining Bloom filter values for encrypted queries, we believe these evaluations against adaptive adversaries are applicable. Additionally, an attack model leveraging machine learning has been proposed [1]. Given recent advances in machine learning, such attack models are natural. When designing searchable

encryption schemes using Bloom filters, evaluating them against adaptive adversaries and machine-learning-based attacks remains an important future challenge.

Acknowledgments. This work was supported by JSPS KAKENHI Grant Number JP25H01106. This paper was edited for grammar using Copilot.

References

[1] Ghada Almashaqbeh, Allison Bishop, and Hayder Tirmazi. Adversary resilient learned bloom filters. In *ASIACRYPT, Part II*, pages 171–202, 2025.

[2] Mandira Banik and Sanjay Kumar. Blockchain-based public key encryption with keyword search for medical data sharing in cloud environment. *J. Inf. Secur. Appl.*, 78:103626, 2023.

[3] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[4] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004.

[5] Partha Sarathi Chakraborty, Mangesh Shivaji Chandrawanshi, Puspesh Kumar, and Somanath Tripathy. BSMFS: blockchain assisted secure multi-keyword fuzzy search over encrypted data. In *IEEE Blockchain*, pages 216–221. IEEE, 2022.

[6] Lanxiang Chen, Wai-Kong Lee, Chin-Chen Chang, Kim-Kwang Raymond Choo, and Nan Zhang. Blockchain based searchable encryption for electronic health record sharing. *Future generation computer systems*, 95:420–429, 2019.

[7] Yang Chen, Chunlu Zhao, Jin Pan, and Yang Liu. Verifiable attribute-based multi-keyword ranked search scheme in blockchain. *The Computer Journal*, 68(5):510–519, 2025.

[8] Ningning Cui, Dong Wang, Jianxin Li, Huaijie Zhu, Xiaochun Yang, Jianliang Xu, Jie Cui, and Hong Zhong. Enabling efficient, verifiable, and secure conjunctive keyword search in hybrid-storage blockchains. *IEEE Trans. Knowl. Data Eng.*, 36(6):2445–2460, 2024.

[9] Haorui Du, Jianhua Chen, Fei Lin, Cong Peng, and Debiao He. A lightweight blockchain-based public-key authenticated encryption with multi-keyword search for cloud computing. *Security and Communication Networks*, 2022(1):2309834, 2022.

[10] Keita Emura. Comments on "blockchain-assisted public-key encryption with keyword search against keyword guessing attacks for cloud storage". *IEEE Trans. Cloud Comput.*, 13(4):1498–1499, 2025.

[11] Sheng Gao, Yuqi Chen, Jianming Zhu, Zhiyuan Sui, Rui Zhang, and Xindi Ma. BPMS: blockchain-based privacy-preserving multi-keyword search in multi-owner setting. *IEEE Trans. Cloud Comput.*, 11(3):2260–2272, 2023.

[12] Hongmu Han, Zhongpeng Wang, Zhigang Xu, Xinhua Dong, and Wenlong Tian. Enhancing iot security and efficiency: A blockchain-assisted multi-keyword searchable encryption scheme. *IEEE Access*, 12:148677–148692, 2024.

[13] Min Han and Peng Xu. Efficient and verifiable keyword search over public-key ciphertexts based on blockchain. *J. Inf. Secur. Appl.*, 89:103924, 2025.

[14] Ruiwei Hou, Fucai Zhou, Qiang Wang, Zi Jiao, Jintong Sun, and Zongye Zhang. Revokable blockchain-enabled ranked multi-keyword attribute-based searchable encryption scheme with mobile edge computing for vehicular. *IEEE Trans. Netw. Serv. Manag.*, 22(3):2764–2779, 2025.

[15] Shengshan Hu, Chengjun Cai, Qian Wang, Cong Wang, Xiangyang Luo, and Kui Ren. Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization. In *IEEE INFOCOM*, pages 792–800. IEEE, 2018.

[16] Qiong Huang and Hongbo Li. An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks. *Information Sciences*, 403:1–14, 2017.

[17] Shan Jiang, Jiannong Cao, Julie A. McCann, Yanni Yang, Yang Liu, Xiaoqing Wang, and Yuming Deng. Privacy-preserving and efficient multi-keyword search over encrypted data on blockchain. In *IEEE Blockchain*, pages 405–410. IEEE, 2019.

[18] Qinyi Li and Xavier Boyen. Public-key authenticated encryption with keyword search made easy. *IACR Commun. Cryptol.*, 1(2):16, 2024.

[19] Yihuai Liang, Yan Li, and Byeong-Seok Shin. Dynamic authenticated keyword search in hybrid-storage blockchain. *Future Gener. Comput. Syst.*, 155:53–65, 2024.

[20] Zheli Liu, Tong Li, Ping Li, Chunfu Jia, and Jin Li. Verifiable searchable encryption with aggregate keys for data sharing system. *Future Generation Computer Systems*, 78:778–788, 2018.

[21] Chen Lotan and Moni Naor. Adversarially robust bloom filters: Monotonicity and betting. *IACR Communications in Cryptology*, 2(1):24, 2025.

[22] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[23] Ying Miao, Keke Gai, Jing Yu, Yu-an Tan, Liehuang Zhu, and Weizhi Meng. Blockchain-empowered keyword searchable provable data possession for large similar data. *IEEE Trans. Inf. Forensics Secur.*, 20:1374–1389, 2025.

[24] James K. Mullin. A second look at bloom filters. *Communications of the ACM*, 26(8):570–571, 1983.

[25] Moni Naor and Noa Oved. Bet-or-Pass: Adversarially robust bloom filters. In *TCC, Part II*, pages 777–808, 2022.

[26] Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. In *CRYPTO, Part II*, pages 565–584, 2015.

[27] Zheng Yao Ng and Iftekhar Salam. Blockchain-based multi-keyword search on encrypted COVID-19 contact tracing data. In *ISPEC*, pages 75–92, 2022.

[28] Fanfan Shen, Lin Shi, Jun Zhang, Chao Xu, Yong Chen, and Yanxiang He. BMSE: blockchain-based multi-keyword searchable encryption for electronic medical records. *Comput. Stand. Interfaces*, 89:103824, 2024.

[29] Jin Sun, Nana Song, Lu Wang, Kexin Ye, and Mengna Kang. A blockchain-based multi-keyword rank search scheme for B+ tree inverted index. *Comput. Stand. Interfaces*, 93:103968, 2025.

[30] Shi-Feng Sun, Ron Steinfeld, Shangqi Lai, Xingliang Yuan, Amin Sakzad, Joseph K Liu, Surya Nepal, and Dawu Gu. Practical non-interactive searchable encryption with forward and backward privacy. In *Usenix Network and Distributed System Security Symposium 2021*. The Internet Society, 2021.

[31] Haiqin Wu, Boris Düdder, Liangmin Wang, Zhenfu Cao, Jun Zhou, and Xia Feng. Survey on secure keyword search over outsourced data: From cloud to blockchain-assisted architecture. *ACM Comput. Surv.*, 56(3):63:1–63:40, 2024.

[32] Xixi Yan, Suwei Feng, Yongli Tang, Pei Yin, and Dazhi Deng. Blockchain-based verifiable and dynamic multi-keyword ranked searchable encryption scheme in cloud computing. *J. Inf. Secur. Appl.*, 71:103353, 2022.

[33] Xiaodong Yang, Jiaqi Wang, Wanting Xi, Tian Tian, and Caifen Wang. A blockchain-based keyword search scheme with dual authorization for electronic health record sharing. *J. Inf. Secur. Appl.*, 66:103154, 2022.

[34] Hongtao Yu, Suhui Liu, Liquan Chen, and Yuan Gao. Blockchain-enabled one-to-many searchable encryption supporting designated server and multi-keywords for Cloud-IoMT. *J. Syst. Archit.*, 149:103103, 2024.

[35] Duo Zhang, Shangping Wang, Qian Zhang, and Yaling Zhang. Attribute based conjunctive keywords search with verifiability and fair payment using blockchain. *IEEE Trans. Serv. Comput.*, 16(6):4168–4182, 2023.

[36] Kai Zhang, Xinyi Hu, Jian Zhao, Lifei Wei, and Jianting Ning. Blockchain-based revocable key-aggregate searchable encryption for group data sharing in cloud-assisted industrial IoT. *IEEE Internet of Things Journal*, 12(11):16899–16911, 2025.

[37] Leyou Zhang, Runze Tian, Qing Wu, and Fatemeh Rezaeibagha. BA-AMST: A blockchain-assisted anonymous and multi-keyword searchable-traceable data sharing system in industrial internet of things. *Engineering Science and Technology, an International Journal*, 66:102055, 2025.

[38] Lizhe Zhang, Wei Luo, Jiahao Li, Zhijun Wu, and Ruiqi Li. Blockchain-assisted keyword search scheme for SWIM service based on improved CSC-Cuckoo filter. *Int. J. Comput. Intell. Syst.*, 17(1):254, 2024.

[39] Yinghui Zhang, Robert H Deng, Jiangang Shu, Kan Yang, and Dong Zheng. TKSE: Trustworthy keyword search over encrypted data with two-side verifiability via blockchain. *IEEE Access*, 6:31077–31087, 2018.

[40] Yuan Zhang, Chunxiang Xu, Jianbing Ni, Hongwei Li, and Xuemin Sherman Shen. Blockchain-assisted public-key encryption with keyword search against keyword guessing attacks for cloud storage. *IEEE Trans. Cloud Comput.*, 9(4):1335–1348, 2021.