Asymptotically Optimal and Secure Multiwriter/Multireader Similarity Search

HYUNSOO KWON1 AND CHANGHEE HAHN2

1Samsung Electronics, Hwaseong 18448, South Korea
2Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul 01811, South Korea

Corresponding author: Changhee Hahn (chahn@seoultech.ac.kr)

This work was supported by the Seoul National University of Science and Technology.

ABSTRACT Privacy-preserving similarity search is a method of data retrieval from potentially untrusted hosts based on the similarity between encrypted data items. In this setting, a major concern is how to support searches when multiple users (multireader) request for searching similar items over data encrypted by multiple data owners (multiwriter). Unfortunately, previous similarity search schemes address this by enforcing users to communicate with data owners. This limitation incurs a significant communication overhead. Moreover, these schemes use deterministic algorithms to encrypt data, which not only violates the privacy of data but also complicates the proof of semantic security. In this paper, we propose an efficient and secure multiwriter/multireader similarity search scheme over encrypted data in cloud storage. In the proposed scheme, the cloud server is able to perform searches without incurring any interaction between users and data owners. Thus, we achieve asymptotically optimal communication cost. We provide rigorous proofs of data privacy in the standard model. Then, we show the proposed scheme achieves semantic security based on the data privacy. An in-depth experiment on an INRIA image dataset demonstrates the practicality of the proposed scheme.

INDEX TERMS Similarity search, searchable encryption, cloud computing security.

I. INTRODUCTION

Delegating management of sensitive data to potentially untrusted hosts such as cloud service providers poses privacy concerns [1], [2]. Data encryption before outsourcing it may seem effective, but doing so complicates some important functions such as similar data searches. Privacy-preserving similarity search schemes were proposed to address this problem by enabling data retrieval based on the similarity between encrypted data items [23], [24], [25]. In these schemes, a data owner (or writer) and a user (a reader) must share a common secret key to encrypt their data contents and queries, respectively. However, sharing secrets is impractical in modern data outsourcing systems because the data owner and the user are unlikely to be in the same trust domain.

To tackle this issue, Cui et al. showed how to enable similarity searches even when data owners (multiwriter) and users (multireader) encrypt their data contents and queries under different keys they each generate (This is called a multiwriter/multireader (M/M) setting.) [27]. In this scheme, however, users must communicate with all of the data owners to re-encrypt their queries under the data owners’ keys during searches. Because both queries and data owners can be numerous in M/M settings, such limitation critically limits the scalability of similarity searches. Hahn et al. proposed a solution minimizing such an overhead [28]. Their scheme adopts a trusted key server which generates secret keys on behalf of data owners and users. These secret keys are correlated with some public parameters which are used by the cloud server to perform searches without re-encryption. However, securely retrieving the encrypted similar data and decrypting them are not presented in the scheme. Moreover, a formal security proof is not presented in the study.

Another problem of previous similarity search schemes is the violation of query privacy. Assume a user sends a series of encrypted queries, but no match is found in a database. In this scenario, an adversary can infer the plain query data from the ciphertext by exploiting side information such as a
reference query log [5]. We refer to this case as no match found (NMF) and securing queries when NMF occurs as NMF security. Such leakage of the plain query data is due to using deterministic algorithms to encrypt queries [23], [24], [25], [26], [27]. In this regard, the best way to achieve NMF security is to randomly select search queries. However, randomizing them in the previous schemes impedes correct searches by making even the same data look totally different.

Last but not least problem of previous schemes is lack of formal security proof. Specifically, Kuzu et al. [23] proposed a formally defined semantic security notion for similarity search. The key to proving semantic security is how to reduce the security of a similarity search scheme to the security of algorithms used for encrypting the owners’ data contents and users’ queries. Kuzu et al. addressed this by using a deterministic pseudo-random permutation for encryption. Unfortunately, however, using deterministic algorithms is against achieving NMF security. In this respect, the best approach to solve this dilemma is to use semantically secure algorithms designed specifically for encrypting data contents and queries. Then, we can reduce the security of a similarity search scheme to the security of these algorithms. To the best of our knowledge, however, none of the previous similarity search schemes have achieved this so far.

In this paper, we propose an efficient and secure similarity search scheme for cloud storage. A key challenge when designing this scheme is the provision of search capabilities for M/M settings without relying on users to communicate with data owners. To this end, we divide both the data content and the query into two components and encrypt them independently under different keys. The first component is for the actual data encryption while the second is for the randomization. We apply a bilinear mapping function to each pair of components. We then compare the resulting values to test whether the query term has matches in the data content. In this way, the search process does not require any communication between the users and data owners to re-encrypt queries. Thus, it is asymptotically optimal in terms of communication. Next, we rigorously prove the security of the proposed algorithms for encrypting data contents and queries in the standard model. Then, we prove that the proposed scheme is semantically secure based on the data privacy. Lastly, we provide extensive performance analyses comparing our scheme with the most advanced similarity search scheme recently proposed for M/M settings [27]. Specifically, we implement the two schemes in a comparative experimental evaluation using INRIA Copydays data [30], a set of real-world images that are widely used for evaluating similarity search. We demonstrate the applicability of the proposed scheme to practical cloud storage systems.

II. RELATED WORK
Similarity searches over encrypted data stem from searchable encryption [9], [10], [11] which allows privacy-preserving searches. In searchable encryption, a data owner (writer) encrypts searchable indexes which are attached to the encrypted database, a user (reader) encrypts trapdoors which is used as search queries, and an untrusted server runs an equality test between the indexes and trapdoors without decrypting them. Since most searchable encryption schemes focus only on single keyword searches, they do not provide flexible search capabilities.

To address the issue, fuzzy searchable encryption was introduced [22], which is similar to conventional searchable encryption in that it also provides a keyword search. The main difference is that, as “fuzzy” indicates, it supports flexibility when searching for keywords, i.e., minor typos or format inconsistencies are accepted. However, the fuzzy searchable encryption is still inefficient because, in real-word datasets, even the same data can have diverse formats, encodings, or edits [30], [31].

To enhance accuracy of similarity search, most schemes use identifiable information extracted from data such as images to measure the similarity between different data items [23], [24]. Specifically, as a building block, these schemes rely on an approximate near neighbor search algorithm called locality sensitive hash [3]. Since an LSH function, on input two similar but different data items, returns the same hash values. These values can be used to test the similarity between data. To preserve privacy, the LSH output values are encrypted and published as a searchable index (and trapdoor) in the similarity search schemes. If the encryption algorithm is deterministic, one can test whether two data items are similar if their searchable index and trapdoor are the same.

Depending on the number of readers and writers, searches on encrypted data are built on one of the following models [4]: (1) single writer / single reader (S/S); (2) multiwriter / single reader (M/S); (3) single writer / multireader (S/M); and (4) multewriter / multireader (M/M). For example, searchable symmetric encryption (SSE) supports S/S settings where the indexes and trapdoors are produced by the same user [10], while public key encryption with keyword search (PEKS) supports M/S settings where multiple users encrypt indexes, and only a private key holder can search [9].

In similarity searches, a practical challenging issue is how to efficiently support multiple readers and writers, while achieving high performance and scalability. Because SSE and PEKS are not suitable for multireader settings, recent work on S/M settings has allowed both the data owner to encrypt indexes and multiple users to search [18], [19], [20]. In [18], each user encrypts his search queries using his secret key, and those encrypted queries need to be re-encrypted under the data owner’s key. This encryption is deterministic so anyone can check whether queries are repeated or not. In [19] and [20], the data owner publishes his public key, and each user makes trapdoors by encrypting their queries using that public key and randomly chosen values. Given the searchable index and trapdoor, an untrusted server can perform an equality test without any form of re-encryption. However, query encryption is dependent on the data owner’s public key. Thus, extending their scheme to M/M settings
requires each query to be repeatedly encrypted under the public keys of all the data owners.

Unfortunately, this efficiency problem in M/M settings has not yet been solved for similarity searches [23], [24], [26], [27]. Similarity search schemes are limited to either S/S [23], [24], [25] or S/M settings [26]. Although Cui et al. [27] proposed a similarity search scheme for M/M settings, the scheme suffers from inefficient communication overhead. Specifically, a user is forced to communicate with data owners to re-encrypt their queries under the data owners’ secret key. In this scheme, a user and each data owner mutually compute a public value, called a digest, which is later uploaded to the cloud to re-encrypt the queries. Furthermore, the number of these digests increases in proportion to the number of data owners, leading to an unacceptable communication overhead in practice.

In terms of security, recent proposals showed that searches on encrypted data could be vulnerable to data recovery attacks [5]. Later, Cash et al. proposed a similar but improved form of attack, which can be successful with less knowledge about the dataset [7]. In these attacks, search queries can be guessed with high probability, assuming that an adversary has background information about the dataset. These attacks exploit the fact that queries are deterministically encrypted. Therefore, if the encrypted queries are deterministically generated, the adversary can then learn whether some of the queries are repeated or not and furthermore infer which data the queries contain by exploiting data recovery attacks [5].

Unfortunately, to the best of our knowledge, all previous similarity search schemes use deterministic encryption algorithms to encrypt searchable indexes and queries [23], [24], [26], [27]. This is because conventional randomizing encryption hinders the search capabilities of legitimate users. Thus, query privacy is not preserved, especially when no matching data are found. Recently, Hahn et al. proposed a query randomizing technique in the similarity search literature [28]. However, this scheme only provides a way to locate similar data stored in the cloud server and lacks a data retrieval procedure for users. Furthermore, the security of the scheme is not formally proved in the standard model and creating a new scheme which can be is one of the main goals of this paper.

Zhang et al. proposed a file-injection attack which leads to the query leakage even if the search pattern is hidden [8]. The attack exploits only the data returned in response to each query. To perform the attack, an adversary injects maliciously-chosen data into the dataset. If a query returns the injected data as a search result, the adversary can then easily determine the plaintext query. Existing solutions such as ORAM based constructions are known to be secure against the file-injection attack [36], [37]. However, they suffer from a large bandwidth cost, which makes the schemes impractical to use in the real world.

Recently, several schemes were proposed to (partially) guarantee security against file-injection attacks by achieving forward secrecy [38], [39], [40]. In [38], given newly inserted data, an adversary cannot perform searches with previously sent queries even if the queries and the newly inserted data contain the same plaintext. Two follow-up studies improve the security by providing backward secrecy [39] or the efficiency by supporting parallel searches [40]. Unfortunately, all the proposed techniques are limited to S/S settings and single keyword searches. Moreover, forward secrecy only guarantees privacy of previously sent queries, not future queries. Therefore, constructing secure similarity searches in the presence of file-injection attackers, especially for M/M settings, is still the open and challenging problem, which we leave as a future work.

III. BACKGROUND

In this section, we briefly introduce locality sensitive hashing and its application to similarity searches, bilinear maps, and security assumptions. We also give algorithmic definitions of similarity searches in M/M settings.

A. LOCALITY SENSITIVE HASHING

Locality sensitive hashing (LSH) is an approximation algorithm that extracts identifiable features from data to measure the similarity between different items [3]. The basic idea of LSH is to reduce the dimensionality of high-dimensional data using a set of hash functions that map similar items to the same values with high probability. Given a data item, an LSH function extracts a sub-feature set \( \{f_1, \ldots, f_l\} \). Then, it conducts equality testing to measure the similarity between the feature sets of two items and this process determines the search accuracy of LSH-based similarity searches.

B. SIMILARITY SEARCH USING LSH

Searching similar data based on LSH is to check whether the number of LSH matches exceeds some pre-defined threshold. Assume we have \( \{f_i| f_i \leftarrow LSH(M)\}_{i \in \{1, \ldots, l\}} \) as a set of searchable indices and \( \{f_i'| f_i' \leftarrow LSH(M')\}_{i \in \{1, \ldots, l\}} \) as a set of trapdoors, and we want to determine whether \( M \) and \( M' \) are similar. To do this, a counter variable, say \( count \), is initialized as 0, and for all \( i \in \{1, \ldots, l\} \), if \( f_i = f_i' \), \( count \) is increased by 1. In the end, given a pre-determined threshold \( th \), if \( count \geq th \), \( M \) and \( M' \) are considered to be similar.

One can realize similarity search schemes [23], [24], [26] using conjunction keyword searches [19], [20] by replacing keywords with LSH values and vice versa [27]. Thus, the proposed scheme can also be applied to conjunction keyword searches. However, none of the existing keyword and similarity search schemes are asymptotically optimal regarding M/M settings, a problem which we attempt to solve in the context of similarity search.

C. BILINEAR MAPS

Let \( G \) and \( G_T \) be multiplicative cyclic groups of prime order \( p \) and \( \mathbb{Z}_p^\times \) be a group under multiplication modulo \( p \). Let \( e : G \times G \rightarrow G_T \) be a bilinear map with the properties:

\[ e(ab, c) = e(a, c) e(b, c) \]

\[ e(a, bc) = e(a, b) e(a, c) \]

\[ e(a, b) = e(b, a) \]

\[ e(1, b) = 1 \]

\[ e(a, b) = e(a, c) \Rightarrow a = c \]
Bilinearity: for all \( P, Q \in \mathbb{G} \) and \( a, b \in \mathbb{Z}_p^* \), we have \( e(P^a, Q^b) = e(P, Q)^{ab} \).

Symmetry: for all \( P, Q \in \mathbb{G} \), we have \( e(P, Q) = e(Q, P) \).

Nondegeneracy: \( e(g, g) \neq 1 \), where \( g \) is the generator of \( \mathbb{G} \).

We say that \( \mathbb{G} \) is a bilinear group if both the group operation in \( \mathbb{G} \) and the bilinear map \( e : \mathbb{G} \times \mathbb{G} \rightarrow \mathcal{G}_T \) are efficiently computable.

**D. SECURITY ASSUMPTION**

We define three complexity assumptions: Diffie-Hellman (DH), Bilinear Diffie-Hellman (BDH), and \( q \) Bilinear Diffie-Hellman Exponent (\( q \)-BDHE). These assumptions are used to provide the security for a query, index, and file, respectively.

1) **DIFFIE-HELLMAN ASSUMPTION**

Let \( \mathbb{G} \) be a bilinear group of prime order \( p \) and \( g \) be a generator of \( \mathbb{G} \). The DH problem in \( \mathbb{G} \) is defined as follows. Given the following set of elements \( y = (g, g^a, g^b) \in \mathbb{G}^3 \) as input, algorithm \( \mathcal{A} \) has advantage \( \epsilon \) in solving DH in \( \mathbb{G} \) if \( \Pr[\mathcal{A}(y) = g^{ab}] \geq \epsilon \), where the probability is over the random choice of generators \( g \) in \( \mathbb{G} \), the random choice of \( \alpha, \beta \in \mathbb{Z}_p^* \), and the random bits used by \( \mathcal{A} \). DH is intractable if all polynomial time algorithms have a negligible advantage in solving DH.

Algorithm \( \mathcal{B} \), which outputs \( \text{coin} \in \{0, 1\} \), has advantage \( \epsilon \) in solving decisional DH in \( \mathbb{G} \) if the following holds:

\[
|\Pr[\mathcal{B}(y, T = g^{ab}) = 0] - \Pr[\mathcal{B}(y, T = R) = 0]| \geq \epsilon.
\]

2) **BILINEAR DIFFIE-HELLMAN ASSUMPTION**

Let \( \mathbb{G} \) be a bilinear group of prime order \( p \) and \( g \) be a generator of \( \mathbb{G} \). The BDH problem in \( \mathbb{G} \) is defined as follows. Given the following set of elements \( y = (g, g^a, g^b, g^s) \in \mathbb{G}^4 \) as input, algorithm \( \mathcal{A} \) has advantage \( \epsilon \) in solving BDH in \( \mathbb{G} \) if \( \Pr[\mathcal{A}(y) = g^{ab + s}] \geq \epsilon \), where the probability is over the random choice of generators \( g \) in \( \mathbb{G} \), the random choice of \( \alpha, \beta, \gamma \in \mathbb{Z}_p^* \), and the random bits used by \( \mathcal{A} \). BDH is intractable if all polynomial time algorithms have a negligible advantage in solving BDH.

Algorithm \( \mathcal{B} \), which outputs \( \text{coin} \in \{0, 1\} \), has advantage \( \epsilon \) in solving decisional BDH in \( \mathbb{G} \) if the following holds:

\[
|\Pr[\mathcal{B}(y, T = g^{ab + s}) = 0] - \Pr[\mathcal{B}(y, T = R) = 0]| \geq \epsilon.
\]

3) **q-BILINEAR DIFFIE-HELLMAN EXPO\( nent \) ASSUMPTION**

Let \( \mathbb{G} \) be a bilinear group of prime order \( p \) and \( g \) be a generator of \( \mathbb{G} \). The \( q \)-BDH problem in \( \mathbb{G} \) is defined as follows. Given the following set of elements \( y = (g, g^a, g^{a^q}, g^{a^{2q}}, \ldots, g^{a^{(q-1)q}}) \in \mathbb{G}^{2q+1} \) as input, algorithm \( \mathcal{A} \) has advantage \( \epsilon \) in solving \( q \)-BDH in \( \mathbb{G} \) if \( \Pr[\mathcal{A}(y) = g^{a^{q+1} + s}] \geq \epsilon \), where the probability is over the random choice of generators \( g \) in \( \mathbb{G} \), the random choice of \( \alpha, s \in \mathbb{Z}_p^* \), and the random bits used by \( \mathcal{A} \). \( q \)-BDH is intractable if all polynomial time algorithms have a negligible advantage in solving \( q \)-BDH.

**TABLE 1. Notations.**

| Symbol | Description |
|--------|-------------|
| \( \lambda \) | Security parameter used in the setup phase |
| \( M \) | Owner’s data needed to generate a searchable index |
| \( M' \) | User’s data needed to generate a trapdoor |
| \( LSH(\cdot) \) | Locality sensitive hash function which, given data, returns subfeatures |
| \( l \) | Number of subfeatures |
| \( f_s \) | \( s \)-th subfeature |
| \( PK \) | A set of public parameters shared among all participating entities |
| \( MK \) | Master secret key |
| \( SK \) | Short-term secret key issued by the key server and held by the user |
| \( DK \) | Long-term decryption key issued by the key server and held by the user |
| \( C_M \) | Searchable index corresponding to \( M \), created by the data owner |
| \( C'_M \) | Cryptertext encrypting \( M \) |
| \( T_M \) | Trapdoor corresponding to \( M \), created by the user |
| \( th \) | Threshold which determines the similarity criterion for the two files |

Algorithm \( \mathcal{B} \), which outputs \( \text{coin} \in \{0, 1\} \), has advantage \( \epsilon \) in solving decisional \( q \)-BDHE in \( \mathbb{G} \) if \( \Pr[\mathcal{B}(y, T = e(g, g)^{a^{q+1}s}) = 0] - \Pr[\mathcal{B}(y, T = R) = 0] \geq \epsilon \).

**E. ALGORITHM DEFINITIONS OF SIMILARITY SEARCH**

We define the similarity search scheme for \( M/M \) settings. The notations we use are described in Table 1. The scheme consists of the following six algorithms:

1) Setup(\( \lambda \)) \rightarrow (PK, MK). The setup algorithm takes as input the security parameter \( \lambda \). It outputs the public parameter \( PK \) and master key \( MK \).

2) BuildIndex(PK, \( f_1, \ldots, f_l \)) \rightarrow (C_M). The index generation algorithm takes as input \( PK \) and the subfeatures \( f_1, \ldots, f_l \) extracted from \( M \). It outputs the encrypted searchable index \( C_M \).

3) Encrypt(PK, \( M \)) \rightarrow (C'_M). The encryption algorithm takes as input \( PK \) and \( M \). It outputs the ciphertext \( C'_M \).

4) Trapdoor(PK, MK, SK, \( f'_1, \ldots, f'_l \)) \rightarrow (T_M'). The trapdoor generation algorithm is divided into two parts:

- **Key generation and Trapdoor generation.**
  - **Key generation.** This takes as input \( MK \) and \( PK \) and outputs the secret key \( SK \) and decryption key \( DK \).
  - **Trapdoor generation.** This takes as input \( SK \) and the subfeatures \( f'_1, \ldots, f'_l \) extracted from \( M' \). It outputs the trapdoor \( T_M' \).

5) Search(C_M, C'_M, T_M', \( th \)) \rightarrow (C_M' or \( \emptyset \)). The similarity search algorithm takes as input \( C_M, C'_M, T_M' \), and threshold \( th \). The algorithm tests the similarity between \( C_M \) and \( T_M' \) using \( th \) and outputs \( C_M' \) if they are similar. The algorithm returns an empty set, \( \emptyset \), if no matching data are found.

6) Decrypt(DK, C_M') \rightarrow (M). The decryption algorithm takes as input \( DK \) and \( C'_M \). It outputs \( M \).

**IV. SYSTEM AND SECURITY MODEL**

**A. SYSTEM MODEL**

The proposed similarity search scheme consists of a key server, data owners (writers), users (readers), and a cloud server. The role of each entity can be described as follows:
1) **Key server.** This is a trusted authority that issues the public parameter. It also interacts with the user and issues the private key (secret key and decryption key) of the user. The public parameters are mainly used for searchable index generation and file encryption, while the private key is used for search query generation and file decryption. It is important to note that a trusted to a similar extent to this key server is also included in the previous searchable encryption schemes [19], [27].

2) **Data owners.** These are the owners of files to be uploaded to a publicly accessible cloud server. Each data owner encrypts data and searchable indexes. The encrypted data and indexes are then uploaded to a publicly accessible cloud server.

3) **Users.** These are the clients who make search queries. Users interact with the key server to obtain a secret key, a decryption key, and to generate trapdoors. Users encrypt their plain query data under distinct secret keys to preserve privacy. Then, they send the trapdoors to the cloud server as a search query. Upon receiving the search results from the cloud server, each user recovers the plaintext data.

4) **Cloud server.** This is a cloud storage service provider that stores the data uploaded by multiple data owners. It performs a similarity search over encrypted data, upon receiving search queries from users. The cloud server is assumed to be honest-but-curious. Thus, it will honestly execute the assigned tasks but try to learn as much information as possible from the encrypted data and search queries. We assume, as the previous schemes have done [16], [17], [19], [21], that the cloud server does not collude with users.

**B. SECURITY MODEL**

1) **NMF IN LARGE QUERY UNIVERSE**

A large query universe refers to an infinite set of queries in the sense that it is possible to add a new query to the system after initialization. On the contrary, in a small query universe, a finite set of queries is fixed at the system setup. In the small query universe, a file-injection attacker can recover plain queries by exploiting access pattern even though queries are encrypted [8]. Moreover, NMF is unlikely to occur if, say, an injected file is intended to match all of the queries of the pre-defined query set. Thus, unless the access pattern is hidden, the query privacy is unreachable in the presence of such a powerful adversary.

In the large query universe, however, performing the file-injection attacks could be impractical because the adversary is unable to guess in advance which queries a user will make. Since users can generate any query of interest in our setting, they may sometimes send a query that will not match all items in the encrypted database. Thus, NMF security is a highly desirable security goal to guarantee the query privacy in the proposed scheme.

2) **NMF SECURITY**

When NMF occurs, deterministic encryption can render the query privacy unattainable since an adversary can infer plain query data [5]. Thus, the best way toward NMF security is to make the query encryption semantically secure. We say that the proposed scheme is **NMF-secure** if the adversary observes a number of encrypted queries that will result in NMF but cannot distinguish pairs of ciphertexts.

It is important to note that guaranteeing NMF security in the presence of such adversary is the same as guaranteeing the semantic security in the presence of eavesdroppers. Therefore, we define NMF security for data privacy using the following game between adversary \( A \) and challenger \( C \).

**Data Privacy Game: \( Game^\text{data}_A \)**

- **Setup.** \( C \) runs a setup algorithm and obtains the public parameter \( PK \) and master key \( MK \). \( A \) is given \( PK \) and outputs the pair of data \( m_0 \) and \( m_1 \).
- **Challenge.** \( C \) picks a random coin \( coin \in \{0, 1\} \) and runs an encryption algorithm to derive \( Enc_{m_0, \text{coin}} \). \( C \) returns \( Enc_{m_0, \text{coin}} \) to \( A \).
- **Guess.** \( A \) outputs its guess \( coin' \in \{0, 1\} \) for \( coin \). The output of the game is 1 if \( coin = coin' \).
- **Restriction.** The queries that \( A \) obtains are limited such that \( (m \neq m_0) \wedge (m \neq m_1) \).

**Definition 1 (NMF Security):** A similarity search scheme has indistinguishability in the presence of an eavesdropper, or is NMF-secure, if for all PPT adversaries \( A \), there is a negligible function \( \text{negl} \) such that

\[
\Pr[Game^\text{data}_A(\lambda) = 1] \leq 1/2 + \text{negl}(\lambda),
\]

where \( \lambda \) is a security parameter and the probability is taken over the randomness used by \( A \) and the randomness used in the data privacy game.

By slightly modifying the data privacy game, we can define the query, index, and file privacy game, respectively. That is, NMF security is not only used for query privacy, but it can also be used for index and file privacy. We give their formal privacy games as follows.

3) **QUERY PRIVACY GAME: \( Game^\text{query}_A \)**

- **Setup.** \( C \) runs the \( \text{Setup} \) algorithm and obtains the public parameter \( PK \) and master key \( MK \). \( A \) is given \( PK \) and outputs a pair of subfeatures \( f_0' \) and \( f_1' \).
- **Challenge.** \( C \) picks a random coin \( coin \in \{0, 1\} \) and runs the \( \text{Trapdoor} \) algorithm to derive \( T_{coin} \), which encrypts \( f_{coin}' \). \( C \) returns \( T_{coin} \) to \( A \).
- **Guess.** \( A \) outputs its guess \( coin' \in \{0, 1\} \) for \( coin \). The output of the game is 1 if \( coin = coin' \).
- **Restriction.** The queries obtained by \( A \) are limited such that \( (f' \neq f_0') \wedge (f' \neq f_1') \).

A similarity search scheme has indistinguishability in the presence of an eavesdropper if, for all PPT adversaries \( A \), there is a negligible function \( \text{negl} \) such that

\[
\Pr[Game^\text{query}_A(\lambda) = 1] \leq 1/2 + \text{negl}(\lambda).
\]

We define the index and file privacy games below similarly.
4) INDEX PRIVACY GAME: \( \text{Game}^{\text{index}} \)
- **Setup.** \( C \) runs the Setup algorithm and obtains the public parameter \( PK \) and master key \( MK \). \( A \) is given \( PK \) and outputs a pair of subfeatures \( f_0 \) and \( f_1 \).
- **Challenge.** \( C \) picks a random coin \( coin \in \{0, 1\} \) and runs the \( \text{BuildIndex} \) algorithm to derive \( C_{\text{coin}} \), which encrypts \( f_{\text{coin}} \). \( C \) returns \( C_{\text{coin}} \) to \( A \).
- **Guess.** \( A \) outputs its guess \( coin' \in \{0, 1\} \) for \( coin \). The output of the game is 1 if \( coin = coin' \).
- **Restriction.** The queries obtained by \( A \) are limited such that \( (f \neq f_0) \land (f \neq f_1) \).

A similarity search scheme has indistinguishability in the presence of an eavesdropper if, for all PPT adversaries \( A \), there is a negligible function \( \text{negl} \) such that

\[
\Pr[\text{Game}^{\text{index}}(\lambda) = 1] \leq 1/2 + \text{negl}(\lambda).
\]

5) FILE PRIVACY GAME: \( \text{Game}^{\text{file}}_{A} \)
- **Setup.** \( C \) runs the Setup algorithm and obtains the public parameter \( PK \) and master key \( MK \). \( A \) is given \( PK \) and outputs a pair of files \( M_0 \) and \( M_1 \).
- **Challenge.** \( C \) picks a random coin \( coin \in \{0, 1\} \) and runs the \( \text{Encrypt} \) algorithm to derive \( C'_{\text{coin}} \), which encrypts \( M_{\text{coin}} \). \( C \) returns \( C'_{\text{coin}} \) to \( A \).
- **Guess.** \( A \) outputs its guess \( coin' \in \{0, 1\} \) for \( coin \). The output of the game is 1 if \( coin = coin' \).
- **Restriction.** The queries obtained by \( A \) are limited such that \( (M \neq M_0) \land (M \neq M_1) \).

A similarity search scheme has indistinguishability in the presence of an eavesdropper if, for all PPT adversaries \( A \), there is a negligible function \( \text{negl} \) such that

\[
\Pr[\text{Game}^{\text{file}}_{A}(\lambda) = 1] \leq 1/2 + \text{negl}(\lambda).
\]

6) ADAPTIVE SEMANTIC SECURITY
When matches are found, one may learn additional information such as search and access patterns, by observing the search results. Hiding such patterns, however, incurs an expensive computation cost, a high number of communication rounds and large storage costs, deterring its widespread use in practice [4]. Therefore, we aim to hide everything except for the patterns. Our security goal can be achieved via \textit{adaptive semantic security}, a formal security notion for similarity search [23]. Specifically, a similarity search scheme is said to achieve adaptive semantic security for all probabilistic polynomial time (PPT) adversaries \( A \), if there exists a simulator \( S \) that can adaptively simulate all the information accessible to \( A \) from what is revealed during a search.

To formally capture security properties, we recapitulate the security definitions given by Kuzu et al. [23]. We start by defining search and access patterns. In a nutshell, the search pattern indicates whether the queries are repeated or not. The formal definition is as follows.

**Definition 2 (Search Pattern):** Let \( \{q_1, \ldots, q_n\} \) be the \( n \) consecutive queries set and \( \pi \) be a binary matrix s.t. \( \pi[i, j] = 1 \) if \( q_i = q_j \) and \( \pi[i, j] = 0 \) otherwise.

Note that deterministic encryption of queries directly reveals the search pattern in the presence of an eavesdropping adversary, something we want to avoid in our construction. Next, the access pattern refers to the information revealed from the search result, formally written as follows.

**Definition 3 (Access Pattern):** Let \( \{T_1, \ldots, T_n\} \) be the trapdoors for the query set \( \{q_1, \ldots, q_n\} \) and \( D(T) \) be a collection of identifiers of searchable indexes that match \( T \). The access pattern \( A_p() \) for the \( n \) trapdoors is set to \( \{A_p(T_1) = D(T_1), \ldots, A_p(T_n) = D(T_n)\} \).

The access pattern can be used to guess about the trapdoors. For example, one trapdoor may return three files, while another may return many, say ten, files. This indicates that the predicate (e.g., a threshold) in the first trapdoor is more restrictive than that in the second.

In a similarity search, a query consists of the subfeatures that are used to construct the trapdoors. Thus, we need to capture the search pattern for subfeatures. The similarity pattern is an extension of Definition 2, formally stated as follows.

**Definition 4 (Similarity Pattern):** Let \( \{f_1^M, \ldots, f_n^M\} \) be the subfeatures of \( M_i \), \( \{f_1^L_1, \ldots, f_n^L_1\}, \ldots, \{f_1^L_j, \ldots, f_n^L_j\} \) be the \( n \) consecutive queries and \( [i] \) be the \( i \)-th subfeature of the \( M_i \). For \( 1 \leq i, p \leq n \) and \( 1 \leq j, r \leq l \), the similarity pattern \( S_p() \) is a binary matrix s.t. \( [i][j], p[r] = 1 \) if \( f_i^L_j = f_p^L_r \) and \( [i][j], p[r] = 0 \) otherwise.

We now formally define a history, trace, and view. In brief, a history is a collection of data and queries generated by data owners and users, respectively. A trace implies all the information that a data owner leaks publicly. A view is the suite of the encrypted history which is accessible to an adversary. Their definitions are given as follows.

**Definition 5 (History):** Let \( D \) be the collection of (file, searchable index) pairs that data owners have and \( Q \) be the set of \( n \) consecutive queries made by users. \( H_n = (D, Q) \) is then defined as an \( n \)-query history.

**Definition 6 (Trace):** Let \( C = \{C_{M_1}, \ldots, C_{M_n}\} \) be the collection of encrypted searchable indexes, \( C' = \{C'_{M_1}, \ldots, C'_{M_n}\} \) be the collection of encrypted files, \( |C| \) and \( |C'| \) be the size of \( C \) and \( C' \), respectively, and \( S_p(H_n) \) and \( A_p(H_n) \) be the similarity and access patterns of \( H_n \). A trace of \( H_n \) is \( \gamma(H_n) = \{((C_{M_1}), \ldots, C_{M_n}), (|C'_{M_1}|, \ldots, |C'_{M_n}|), S_p(H_n), A_p(H_n)\} \).

**Definition 7 (View):** Let \( C = \{C_{M_1}, \ldots, C_{M_n}\} \) be the collection of encrypted searchable indexes, \( C' = \{C'_{M_1}, \ldots, C'_{M_n}\} \) be the collection of encrypted files, and \( T = \{T_{M_1}, \ldots, T_{M_n}\} \) be the trapdoors for \( H_n \). The view is the information accessible to an adversary, and the view of \( H_n \) is \( \nu(H_n) = \{C, C', T\} \).

Given the definitions of trace \( \gamma(H_n) \) and view \( \nu(H_n) \), the security goal we aim to achieve is straightforward: we will build a simulator \( S \) who can build a simulated view \( \nu_S(H_n) \) from \( \gamma(H_n) \). If an adversary \( A \) who has access to a real view \( \nu_R(H_n) \) cannot distinguish \( \nu_R(H_n) \) from \( \nu_S(H_n) \), a similarity
search scheme is said to achieve adaptive semantic security. Formally, we have the following definition.

**Definition 8 (Adaptive Semantic Security for Similarity Searchable Encryption):** Let $H_n$ be a random history from all possible histories, $v_R(H_n)$ be the real view, and $v_S(H_n)$ be the simulated view from $γ(H_n)$. A similarity search scheme achieves adaptively semantic security if there exists a simulator $S$ such that, for all polynomial size distinguishers $D$ and a large $r$,

$$|Pr[D(v_R(H_n)) = 1] − Pr[D(v_S(H_n)) = 1]| \leq \frac{1}{poly(r)}.$$ 

### V. PROPOSED SCHEME

In this section, we describe the proposed similarity search scheme for M/M settings in detail. Without loss of generality we assume that there are $K$ data owners and $N$ users, respectively. Fig. 1 depicts how the cloud server performs searches over the data uploaded by the $j^{th}$ ($1 \leq j \leq K$) data owner when it receives a search query from the $i^{th}$ ($1 \leq i \leq N$) user. As shown in the similarity search procedure, it is important to note that the $i^{th}$ user need not interact with the $j^{th}$ data owner to re-encrypt his query under the secret key of the $j^{th}$ data owner. Whereas, in the previous scheme [27], all users must interact with all data owners to allow the cloud server to perform searches.

### A. THE CONSTRUCTION

Let $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p^*$ be a cryptographic hash function. In what follows, we assume that a set of subfeatures $\{f_1, \ldots, f_l\}$ extracted from $M$ is generated as $f_i := H(LSH(M))$, where the probability of $f_i = 0$ is negligible. This extraction process is performed prior to the index and trapdoor generation algorithms. Specifications for the algorithms of the proposed scheme are described below.

- **Setup($λ$) → ($PK, MK$) by key server.** The setup algorithm takes as input the security parameter $λ$. It chooses a generator $g \in G$ and four random values $α, β, α, z \in \mathbb{Z}_p$. It publishes the public parameter and master secret key as $PK = \{g, (g^α, g^β, g^z)\}$, $MK = \{g^αg^β\}$.

- **BuildIndex($PK, f_1, \ldots, f_l$) → ($C_M$) by data owners.** The searchable index generation algorithm takes as input $PK$ and a set of subfeatures $f_1, \ldots, f_l$ extracted from $M$. It chooses random $r_1, \ldots, r_l \in \mathbb{Z}_p$. For all $i \leq l$, it computes $C_{1,i} = g^{r_i}$, $C_{2,i} = e(g, g^{αr_i})$, and $C_{3,i} = g^{βr_i}$. It publishes the searchable index as $C_M = \{C_{1,i}, C_{2,i}, C_{3,i}\}_{i=1,\ldots,l}^*$. The algorithm then skips the decryption key generation process.

- **Encrypt($PK, M$) → ($C'_M$) by data owners.** The encryption algorithm takes as input $PK$ and $M$. It chooses a random $s \in \mathbb{Z}_p$ and computes $C_1 = g^s$, $C_2 = M \cdot e(g, g^{αs})$, and $C_3 = g^{βs}$. It publishes the ciphertext as $C'_M = \{C_1, C_2, C_3\}$.

- **Trapdoor($PK, MK, SK, f'_1, \ldots, f'_l$) → ($T_M'$).** The trapdoor generation algorithm consists of key generation and trapdoor generation:

  - **KeyGen($PK, MK$) → ($SK, DK$).** The algorithm chooses random $t_1, \ldots, t_l$. For all $i \leq l$, it computes $K_{1,i} = g^{αt_i}$ and $K_{2,i} = g^{βt_i}$. It publishes the secret key as $SK = \{K_{1,i}, K_{2,i}\}_{i=1,\ldots,l}$. Next, the algorithm chooses a random $t$ and computes $D_1 = g^{αt}$ and $D_2 = g^{βt}$. It publishes the decryption key as $DK = \{D_1, D_2\}$. While the secret key is used only once to generate a trapdoor, the decryption key is the long-term key used to decrypt a ciphertext. Thus, if the decryption key has already been issued, the algorithm then skips the decryption key generation process.

  - **TrapGen($SK, f'_1, \ldots, f'_l$) → ($T_M'$).** Having a set of subfeatures $f'_1, \ldots, f'_l$ extracted from $M'$, for all $i \leq l$, the algorithm computes $T_{1,i} = K_{1,i}^{f'_i}$ and $T_{2,i} = K_{2,i}^{f'_i}$. It publishes the trapdoor as $T_M' = \{T_{1,i}, T_{2,i}\}_{i=1,\ldots,l}^*$. The algorithm then skips the trapdoor generation process.

- **Search($C_M, C'_M, T_M'$, $th$) → ($C'_M$ or $∅$) by cloud server.** The search algorithm takes as input $C_M, C'_M, T_M'$, and the threshold $th$. It sets the temporary variable $count$ as 0. For all $i \leq l$, it checks if

$$\frac{e(C_{1,i}, T_{1,i})}{e(C_{3,i}, T_{2,i})} = C_{2,i}.$$

If the above equation holds, then the algorithm returns $C'_M$; otherwise it returns $∅$, indicating that $M$ and $M'$ are dissimilar with regard to $th$.

- **Decrypt($DK, C'_M$) → ($M$) by users.** The decryption algorithm takes as input $DK$ and $C'_M$. It recovers $M$ by computing the following in sequence

$$\frac{e(C_{1,1}, T_{1,1})}{e(C_{3,1}, T_{2,1})} = e(g^{-r_1}, K_{1,1}^{f'_1}) = e(g^{αr_1}, g^{αf'_1}g^{βf'_1}) = e(g^{αr_1}, g^{βf'_1}) = e(g, g^{r_1αβf'_1}) = C_{2,1}.$$

The search and decryption correctness of the proposed scheme can be demonstrated as follows.

- **Search correctness.** Suppose the two $i$-th subfeatures $f_i$ and $f'_i$ are the same. Then, given the $i$-th searchable index ($C_{1,i}, C_{2,i}, C_{3,i}$) and the $i$-th trapdoor ($T_{1,i}, T_{2,i}$), the **Search** algorithm computes

$$\frac{e(C_{1,i}, T_{1,i})}{e(C_{3,i}, T_{2,i})} = e(g^{αt_i}, K_{1,i}^{f'_i}) = e(g^{αt_i}, g^{αf'_i}) = e(g^{αt_i}, g^{αf'_i}g^{βf'_i}) = e(g^{αt_i}, g^{αf'_i}) = e(g^{αt_i}, g^{βf'_i}) = C_{2,i}.$$
data owners can use Encrypt to encrypt a large volume of data efficiently. For example, combined with any symmetric encryption algorithm such as AES.

The proposed encryption algorithm incurs an additional overhead in the key issuing process whenever he runs the trapdoor algorithm. This, however, the user should receive a new secret key from the key server with the user to transfer the secret key. To deal with this issue, we propose two countermeasures: (1) extending to a key-reusable scheme and (2) issuing keys in batches.

1) EXTENSION TO KEY-REUSABLE SIMILARITY SEARCH
The first effective solution to the efficiency problem is to make secret keys reusable. To this end, we use an additive masking technique [34] such that after receiving SK from the key server for the first time, a user additively masks SK using a randomly chosen value, say \( \chi \), to generate a trapdoor. In this way, the user need not interact with the key server for subsequent queries. More precisely, we use every algorithm in the proposed scheme as it is, except for the Trapdoor algorithm, which is modified as follows:

- \( \text{Trapdoor}(PK, SK, f_1', \ldots, f_l') \rightarrow (T_{M'}) \). The algorithm chooses a random value \( \chi \) and computes \( g^{\alpha \chi} \) and \( g^\beta \). Given \( SK = \{K_{1,i}, K_{2,i}\}_{i \in \{1, \ldots, l\}} \), the algorithm computes \( K_{1,i} \cdot g^{\alpha \chi} \) and \( K_{2,i} \cdot g^\beta \). Having a set of subfeatures \( f_1', \ldots, f_l' \) extracted from \( M' \), for all \( i \leq l \), the algorithm computes \( T_{1,i} = K_{1,i} \cdot g^{\alpha \chi} f_i' \) and \( T_{2,i} = K_{2,i} \cdot g^\beta f_i' \).

It publishes the trapdoor as \( T_{M'} = \{T_{1,i}, T_{2,i}\}_{i \in \{1, \ldots, l\}} \).

Since the message encryption is \( C_2 = M \cdot e(g, g)^{\alpha \chi s} \), the algorithm recovers \( M = C_2/e(g, g)^{\alpha \chi s} \) correctly.

B. KEY ENCAPSULATION
The proposed encryption algorithm Encrypt can be combined with any symmetric encryption algorithm such as AES to encrypt a large volume of data efficiently. For example, data owners can use Encrypt to encrypt the AES secret key while the actual data encryption is performed using the AES encryption algorithm. Users then receive a pair of ciphertexts, i.e., the AES secret key encryption part and the data encryption part, as a search result, decrypt the first and second ciphertext accordingly.

C. EFFICIENCY ISSUE
The re-use of previously issued secret keys to generate subsequent queries may render the trapdoor deterministic. Thus, the user should receive a new secret key from the key server whenever he runs the trapdoor algorithm. This, however, incurs an additional overhead in the key issuing process because the key server should establish a secure channel with the user to transfer the secret key. To deal with this issue, we propose two countermeasures: (1) extending to a key-reusable scheme and (2) issuing keys in batches.
unknown (see §IV-B). This is because how to simulate the randomizing component $\chi$ in the context of standard-model adversaries is unclear. Thus, we would like to leave the formal security proof of this extension, which is outside the focus of this study for future work.

2) ISSUING SECRET KEYS IN BATCHES
The second solution to the efficiency problem is to issue a large number of secret keys at once. The user then picks a key to make a query, discards the key once used, and later chooses a fresh key for the next query. This approach allows the user to make randomized trapdoors without contacting the key server until the issued secret keys run out, but creates a storage overhead for the user when saving the keys. However, the cost of storing the keys is acceptable for modern computing systems. When implementing the proposed scheme based on the PBC library [29], key sizes range from 512 to 2048 bytes, depending on the number of subfeatures, and even recent light-weight computing devices can provide sufficient memory space. For example, the VoCore2 [33], a coin-sized cheap Linux computer, supports storage of up to 2 TB. Thus, issuing secret keys in batches is realistic for practical systems.

D. IMPLEMENTATION ISSUE
The data $M$ is an element of $G_T$, the size of which is 128 bytes according to the implementation of the scheme, but real-world data sizes, particularly for cases such as multimedia, are likely to be greater than that of $G_T$ elements. To efficiently address this issue, we can implement the Encrypt algorithm as a hybrid encryption composed of key encapsulation and symmetric encryption. That is, $M$ is replaced by a symmetric key in $G_T$, and the actual data is encrypted using symmetric encryption, AES for example, under the symmetric key.

E. REVOCATION ISSUE
The issuance of (long-term) decryption keys to users may potentially raise concerns about key revocation in the case of the user revocation or key compromise. An effective way to address this is to adopt key-revocable encryption algorithms such as [35]. Specifically, the data $M$ is encrypted (or decrypted) using the key-revocable encryption (decryption) algorithm in [35]. Users get issued revocable decryption keys efficiently address this issue, we can implement the scheme. Finding similar files and encrypting (or decrypting) with the proposed similarity search because, in the proposed scheme, the corresponding data are designed to be independent of each other. That is, one can freely replace the data encryption (decryption) part of the proposed scheme with any encryption schemes. Therefore, the key revocation problem can be effectively handled under the proposed scheme in this way.

VI. SECURITY ANALYSIS
In this section, we prove the proposed scheme guarantees query, index, and file privacy. Then, we prove the proposed scheme is adaptive security safe. It is important to note that the adaptive semantic security of the proposed scheme is reduced to the security of data privacy which is formally proved in the standard model.

Theorem 1: Suppose the decisional DH assumption holds. Then, the proposed scheme has query privacy.

Proof: Suppose adversary $A$ has the non-negligible advantage $\epsilon = \text{Adv}_A$ in Game$^{\text{query}}_A$ against the proposed scheme. Using $A$, we show how to build a simulator $B$ that solves the decisional DH problem in $G$ without relying on bilinear maps.\(^2\)

Initialization: Simulator $B$ takes in a DH challenge tuple $(y, T)$.

Setup: $B$ parses $y = (g, g^a, g^b)$, chooses a random $b \in \mathbb{Z}_p$, and implicitly sets $b' = b + \beta$ by computing $g^{b'} = g^b \cdot g^\beta$. $B$ computes $e(g^a, g^{b'})$, picks random $a, z \in \mathbb{Z}_p$, and computes $g^a, g^{az}$. $B$ sets $PK = (g, e(g, g)^{az}, g^a, g^{az})$, and gives $PK$ to $A$. $A$ outputs a pair of subfeatures $f'_0$ and $f'_1$.

Challenge: $B$ chooses a random $c_0 \in \{0, 1\}$. It chooses a random $t \in \mathbb{Z}_p$ and computes $T_1 = T_{c_0} \cdot g^{abf'_0} \cdot g^{azc_0f'_0}$, and $T_2 = g^{bf'_0} \cdot B$ gives $A$ $(T_1, T_2)$.

Guess: $A$ eventually outputs guess $c_0'$ for $c_0$. Following this, $B$ outputs $0$ to guess that $T = g^{bf'_0}$ if $c_0' = c_0$. Otherwise, it outputs $1$ to indicate that $T$ is a random element in $G_T$.

If $T = g^{ab}$, then $T_1$ satisfies the following:

$$T_{c_0} \cdot g^{abf'_0} \cdot g^{azc_0f'_0} = g^{abf'_0} \cdot g^{azc_0f'_0} = g^{abf'_0} \cdot g^{azc_0f'_0},$$

which indicates that $T_1$ is well formed and thus $B$ simulates the challenge trapdoor correctly.

When the input tuple is sampled from $(y, T)$, where $T = g^{ab}$, then $A$’s view is identical to its view in Game$^{\text{query}}_A$ and therefore we have $\text{Pr}[B(y, T = g^{ab}) = 0] = \frac{1}{2} + \text{Adv}_A$. When the input tuple is sampled from $(y, T)$, where $T$ is a random group element, then the subfeature is completely hidden from the adversary, and therefore we have $\text{Pr}[B(y, T = R) = 0] = \frac{1}{2}$. Thus, with $g, a, T$ uniform in $G, \mathbb{Z}_p$, and $G_T$, respectively, $B$ solves the decisional DH problem with a non-negligible advantage. This completes the proof of the theorem.

Theorem 2: Suppose the decisional BDH assumption holds. Then, the proposed scheme has index privacy.

Proof: Suppose adversary $A$ has the non-negligible advantage $\epsilon = \text{Adv}_A$ in Game$^{\text{index}}_A$ against the proposed scheme. Using $A$, we show how to build a simulator $B$ that solves the decisional BDH problem in $G$.

\(^2\)Note that, given $(g, g^a, g^b)$ and $T$, the simulator can trivially distinguish whether $T$ is $g^{ab}$ or a random value by determining whether the two values $e(g^a, g^b)$ and $e(T, g)$ are the same or not.
**Initialization:** Simulator $B$ takes in a BDH challenge tuple $(y, T)$.  

**Setup:** $B$ parses $y = (g, g^a, g^b, g^c)$. $B$ computes $e(g^a, g^b)$, picks random $a, z \in \mathbb{Z}_p$, and computes $g^a, g^{ac}$. The simulator sets $PK = (g, e(g, g^b), g^a, g^{ac})$, and gives $PK$ to $A$. $A$ outputs a pair of subfeatures $f_0$ and $f_1$.  

**Challenge:** $B$ chooses a random coin $\in \{0, 1\}$. It chooses a random $b \in \mathbb{Z}_p$ and implicitly sets $b' = \gamma + b$ by computing $g^{b'} = g^\gamma \cdot g^b$. $B$ sets $C_{1,*} = g^{b'}, C_2, = T_{\text{coin}}, e(g, g)^{a+b'}$, and $C_{3,*} = g^{ac}$. $B$ gives $A$ $(C_{1,*}, C_{2,*}, C_{3,*})$.  

**Guess:** $A$ eventually outputs guess $\text{coin'}$ for coin. Following this, $B$ outputs 0 to guess that $T = e(g, g)^{a+b'}$ if $\text{coin'} = \text{coin}$. Otherwise, it outputs 1 to indicate that $T$ is a random element in $\mathbb{G}$.  

If $T = e(g, g)^{a+b'}$, then $C_{2,*}$ satisfies the following:  

\[
T_{\text{coin}}, e(g, g)^{a+b'} = e(g, g)^{a+b'}, T_{\text{coin}} \cdot e(g, g)^{(b'-\gamma)}_{\text{coin}} = e(g, g)^{a+b'}_{\text{coin}},
\]

which indicates that $C_{2,*}$ is well formed and thus $B$ simulates the challenge index correctly.  

When the input tuple is sampled from $(y, T)$, where $T = e(g, g)^{a+b'}$, then $A$’s view is identical to its view in $\text{Game}_A^\text{index}$ and therefore we have $\Pr[B(y, T = e(g, g)^{a+b'}) = 0] = \frac{1}{2} + \text{Adv}_A$. When the input tuple is sampled from $(y, T)$, where $T$ is a random group element, then the file is completely hidden from the adversary, and therefore we have $\Pr[B(y, T = R) = 0] = \frac{1}{2}$. Thus, with $g, a$ and $T$ uniform in $\mathbb{G}$, $\mathbb{Z}_p$, and $\mathbb{G}_T$, respectively, $B$ solves the decisional BDH problem with a non-negligible advantage. This completes the proof of the theorem.  

**Theorem 4:** If the proposed scheme ensures query, index, and file privacy, then the proposed similarity search scheme has adaptive semantic security.  

**Proof:** The proof will show the existence of the polynomial simulator $S$ in which the simulated view $v_S(H_n)$ and the real view $v_R(H_n)$ are computationally indistinguishable. Let the real and simulated view be as follows:  

\[
v_R(H_n) = \{(C_{M_1}, \cdots, C_{M_\ell}), (C'_{M_1}, \cdots, C'_{M_\ell})\},
\]

\[
v_S(H_n) = \{(C'_{M_1}, \cdots, C'_{M_\ell}), (C'_{M_1}, \cdots, C'_{M_\ell})\},
\]

Let $\gamma(H_n)$ be the trace of $H_n$. Then $S$ exploits $\gamma(H_n)$ to adaptively generate the simulated view $v_S(H_n)$ as follows:  

**A. SIMULATING SEARCHABLE INDEXES**  
$S$ chooses $\{C_{M_1}, \cdots, C_{M_\ell}\}$, each of which are randomly selected such that, for all $i \in \{1, \ldots, n\}$, $|C_{M_i}| = |C_{M_i}|$. Based on the indistinguishability property of the proposed scheme in relation to the $\text{BuildIndex}$ algorithm, the searchable index is computationally indistinguishable from a random value. Thus, $C_{M_i} \leftarrow \text{BuildIndex}(PK, M_i)$ and $C'_{M_i}$ are computationally indistinguishable.  

**B. SIMULATING FILES**  
$S$ chooses $m$ random $\{C'_{M_1}, \cdots, C'_{M_\ell}\}$ such that, for all $i \in \{1, \ldots, n\}$, $|C'_{M_i}| = |C'_{M_i}|$. Based on the indistinguishability property of the proposed scheme in relation to the $\text{Encrypt}$ algorithm, the file is computationally indistinguishable from a random value. Thus, $C_{M_i} \leftarrow \text{Encrypt}(PK, M_i)$ and $C'_{M_i}$ are computationally indistinguishable.  

**C. SIMULATING TRAPDOORS**  
$S$ chooses $n$ random $\{T_{M_1}^*, \cdots, T_{M_\ell}^*\}$ such that, for all $i \in \{1, \ldots, n\}$, $|T_{M_i}^*| = |T_{M_i}|$. Based on the indistinguishability property of the proposed scheme in relation to the $\text{Trapdoor}$ algorithm, the trapdoor is computationally indistinguishable from a random value. Thus, $T_{M_i} \leftarrow \text{Trapdoor}(PK, M_k, SK, f_i)$ and $T_{M_i}^*$ are computationally indistinguishable.
D. ACHIEVING ADAPTIVE SEMANTIC SECURITY

We have shown that each of the three simulations was taken independently by $S$, and the simulation succeeded due to the indistinguishability property. We now show how $S$, who has access to the Trapdoor and BuildIndex algorithm, simulates the similarity and access pattern to generate $v_S(H_a)$ which is computationally indistinguishable from $v_R(H_a)$.

- **Simulating the similarity pattern.** $S$ chooses the random subfeatures $\{f_i^{M_1}, \ldots, f_i^{M_4}\}$ for all $i \in \{1, \ldots, n\}$. For any $1 \leq j, r \leq n$ and $1 \leq j, r \leq l$, if $[f_i^j, p^i_r] = 1$, then set $f_i^{M_4} = f_i^{M_4}/f_i^{M_4}$. For each element in $\{f_i^{M_1}, \ldots, f_i^{M_4}\}$, $S$ runs the Trapdoor algorithm and outputs the trapdoor $T_i^{M} = (T_{1,i}, T_{2,i})_{i \in \{1, \ldots, l\}}$.

- **Simulating the access pattern.** $S$ chooses the randomly selected files $\{C_i^{M_1}, \ldots, C_i^{M_4}\}$ and the corresponding random subfeatures $\{f_i^{M_1}, \ldots, f_i^{M_4}\}$ for all $i \in n$. For each $(A_p(T_{M}), S)$ replaces the subfeatures with those who are used to generate the simulated trapdoor. $S$ then runs the BuildIndex algorithm and outputs the searchable index $C_i^{M_4} = \{C_{1,i}, C_{2,i}, C_{3,i}\}$.

Since each component of $v_S(H_a)$ and $v_R(H_a)$ are computationally indistinguishable and $v_S(H_a)$ satisfies the similarity and access pattern, the proposed scheme achieves adaptive semantic security, concluding the proof.

VII. PERFORMANCE ANALYSIS
A. THEORETICAL ANALYSIS

We provide a theoretical comparative analysis of the proposed scheme and Cui et al.’s similarity search scheme in an M/M setting [27], which is the only similarity search scheme proposed so far for an M/M setting; others are either conjunctive keyword search schemes [12], [13], [14], [15] or similarity search schemes in S/S settings [23], [24]. The comparative results are shown in Table 2. In the table, $l$ refers to the number of subfeatures, $m$ is the number of data owners, and $H$, $P$, $E_G$, $E_{G_T}$, and $M_{G_T}$ represent the time to compute a cryptographic hash function, pairings, exponentiations in $G$ and $G_T$, and a multiplication in $G_T$, respectively. Note that $P$, $E_G$, and $E_{G_T}$ require significantly more computation time than $H$ and $M_{G_T}$.

In Table 2, ‘Trapdoor.S’, ‘Index.S’, and ‘Ciphertext.S’ represent the size of the trapdoors, searchable indexes, and corresponding ciphertext, respectively. ‘Re-Enc.S’ and ‘Re-Enc.O’ refer respectively to the size of the re-encrypting components and the number of re-encrypting operations. Lastly, ‘Trapdoor.O’, ‘Index.O’, ‘Decrypt.O’, and ‘Search.O’ represent the number of computations required to generate trapdoors and searchable indexes, to decrypt the ciphertext, and to perform searches, respectively.

According to the analysis, Cui et al.’s scheme requires additional computation overhead for re-encryption, while our scheme does not. Specifically, re-encrypting the trapdoors in Cui et al.’s scheme leads to a linear increase in pairing operations with respect to $m$ and $l$. The trapdoor and index sizes of our scheme are approximately two times bigger than those of Cui et al.’s scheme. This is because our scheme enables trapdoors to be randomized, making it resilient to known vulnerabilities in which attackers exploit the deterministic properties of trapdoors (see Sect. II). In contrast, the Trapdoor algorithm of Cui et al.’s scheme is deterministic, so the search pattern is leaked even when no matching data are found, thereby reducing query privacy.

In terms of data encryption and decryption, encrypting a data item requires three exponentiation operations, the resulting ciphertext size is $2|G| + |G_T|$, and two pairing operations required to decrypt the ciphertext. In addition, Cui et al.’s scheme does not support any form of data encryption or decryption; the scheme merely returns IDs as a search result.

We note that this can be somewhat problematic because, in Cui et al.’s scheme, data owners must rely on an additional secure channel to transfer the corresponding data to users. Lastly, both Cui et al.’s and our scheme require approximately the same number of operations for the search process.

B. IMPLEMENTATION & SETUP

We implement the proposed scheme and Cui et al.’s scheme [27] using the pairing-based cryptography library 0.5.14 written in C (PBC) [29] on Ubuntu 16.04.3 (64 bit) with an Intel Xeon CPU E5-2620 (2.10GHz x16) and 32GB RAM. We use a Type-A curve and a Type-D curve to implement the proposed scheme and [27], respectively. Different types of curve are used because our scheme is based on symmetric pairing while [27] is based on asymmetric pairing.

To evaluate actual performance, we use real-world images from the INRIA Copydays dataset [30]. It contains an original set of images and corresponding modifications, i.e., scaled down, lossy JPEG-compressed, or cropped, making it appropriate for evaluating the performance of a similarity search. For each image of the dataset, we use pHash to generate the subfeatures [32].

The proposed scheme and [27] share certain algorithms, such as Setup, BuildIndex, Trapdoor, and Search. These algorithms are evaluated in detail by changing different parameters including the number of subfeatures ($l$), the threshold ($th$), and the number of data owners ($N$). To correctly examine how each parameter influences the algorithms, we fix two parameters while modifying the other.

Note that the number of subfeatures $l$ affects search accuracy and efficiency. Intuitively, a smaller $l$ implies less accurate but faster searches, while a larger $l$ implies more accurate...
but slower searches. Search accuracy also depends on the type of data being searched for. Thus, the optimal number of $l$ needs to be chosen manually. Although improving the accuracy of similarity searches is beyond the scope of this paper, we observe that search accuracy degrades exponentially as $l$ is decreased below four [3], so we set the lowest value of $l$ at four. All of the other parameters are chosen manually to better compare the proposed scheme and [27] in terms of efficiency.

C. EVALUATION

1) COMMUNICATION ANALYSIS

Fig. 2 depicts the communication cost of the proposed scheme and Cui et al.’s scheme [27]. Note that, since the M/M model is designed for data sharing, we assume that users also play the role of data owners. In the figure, we measure the performance of the proposed scheme by deferring the size of $l$ ($l$ is either 4, 10, or 16). This is because the key size of the proposed scheme is linear with respect to $l$.

Fig. 2(a) shows a small-scale data sharing scenario in which the numbers of users and data owners is much less than those of typical cloud storage environments (in which there may be millions of users and data owners). The communication cost of the proposed scheme grows linearly with the number of users, while that of Cui et al.’s scheme shows quadratic growth. More specifically, the proposed scheme outperforms Cui et al.’s scheme when the number of users is around 640 when $l = 4$, 1,410 when $l = 10$, and 2,180 when $l = 16$. The performance gap continues to widen dramatically as $N$ increases, as shown in Fig. 2(b). Therefore, the proposed scheme is much more efficient than Cui et al.’s scheme in terms of communication cost, and more suitable for cloud storage scenarios.

2) COMPUTATION ANALYSIS

Fig. 3 presents the comparative experimental results of the proposed scheme and Cui et al.’s scheme with respect to the Setup algorithms and the 3-tuple $(l, th, N)$. We fix $th = 1$ and $N = 1$ and change $l$ in Fig. 3(a), fix $l = 16$ and $N = 1$ and change $th$ in Fig. 3(b), and fix $l = 10$ and $th = 1$ and change $N$ in Fig. 3(c). While the setup time of the proposed scheme remains the same regardless of $(l, th, N)$, the setup time of Cui et al.’s scheme increases linearly with $N$ in Fig. 3(c). This is because in Cui et al.’s scheme each user interacts with all data owners to compute the re-encrypting components. Thus, the Setup algorithm of the proposed scheme does not depend on the 3-tuple $(l, th, N)$, while that of Cui et al.’s scheme varies with $N$.

It was observed that the BuildIndex and Trapdoor algorithms had linear relationships with $l$ under both schemes (see Fig. 4(a) and 5(a), respectively). However, while the time required for BuildIndex increases linearly as $l$ increases, it remains constant with respect to $(th, N)$, as shown in Fig. 4(b) and Fig. 4(c). In a similar way, the time required for Trapdoor, as seen in Fig. 5(b) and 5(c) remains steady with respect to $(th, N)$. In contrast, Fig. 6 illustrates that the time required for the Search depends on $th$. We observe that our scheme takes less search time than Cui et al.’s scheme does, despite ours requiring more pairing operations. This is because we use a different type of curve to implement the proposed scheme. Specifically, ours uses a Type-A curve for symmetric pairing while Cui et al.’s scheme uses a Type-D curve for asymmetric pairing. In the PBC library [29], Type-A is the fastest and Type-D is slower but effective when elements are short. In fact, it is widely known that symmetric pairings can be implemented much more efficiently than standard asymmetric pairings [41]. Thus, the Search algorithm takes less time in the proposed scheme.

The performance analysis shows the proposed scheme to be always faster than Cui et al.’s scheme, except for Trapdoor execution. Specifically, our Trapdoor algorithm requires four exponentiations, the first two of which are computationally time consuming while the other two generate insignificant computational overhead. In contrast, the Trapdoor algorithm of Cui et al.’s scheme requires only one computationally intensive exponentiation. According to Fig. 5, Cui et al.’s Trapdoor algorithm is approximately two times faster than ours.

We present the total running time of our scheme and Cui et al.’s scheme in Fig. 7. As shown in Fig. 7(a) and Fig. 7(b), both our scheme and Cui et al.’s scheme have a linear relationship with respect to $(l, th)$. Specifically, Fig. 7(a) and Fig. 7(b) show that our scheme is faster than Cui et al.’s scheme. The total running time of Cui et al.’s scheme becomes much longer when we fix $(l, th)$ and modify $N$ (Fig. 7(c)) while that of ours remains approximately constant because none of its algorithms depend on $N$. To conclude, the proposed scheme is always faster than Cui et al.’s scheme for the 3-tuple $(l, th, N)$.  

FIGURE 2. Communication cost for different number of users ($N$).
3) RUNNING TIME OF ENCRYPT AND DECRYPT ALGORITHMS

The proposed scheme provides encryption and decryption of the data which correspond to the searchable indexes, while Cui et al.’s scheme does not. We fix the number of data to 1 and run the Encrypt and Decrypt algorithms. According to our experiment, the average time required for our Encrypt and Decrypt algorithms is 0.01543 ms and 0.01443 ms, respectively. The Encryption algorithm requires three exponentiation operations (two operations are in $\mathbb{Z}_p$ and the other
is in $G_T$) and the Decryption algorithm requires two pairing operations. Moreover, neither algorithm has a dependency on the volume of $l$ or $t$, so the running time of the algorithms remains constant.

VIII. CONCLUSION

In this paper, we proposed a secure and efficient similarity search scheme for M/M settings. The proposed scheme guarantees asymptotically optimal similarity searches and query privacy even when no matching data are returned. We rigorously proved the security of the proposed scheme in terms of query, index, and file privacy under standard complexity assumptions, and then showed that the proposed scheme features adaptive semantic security. As a future research direction, we suggest two interesting issues. The first is how to extend our scheme to support forward and backward privacy, which are of utmost importance for dynamic data updates. The second is to remove the reliance on the trusted entity, e.g., the key server in the proposed scheme.

REFERENCES

[1] K. Ren, C. Wang, and Q. Wang, “Security challenges for the public cloud,” IEEE Internet Comput., vol. 16, no. 1, pp. 69–73, Jan./Feb. 2012.

[2] D. Song, E. Shi, I. Fischer, and U. Shankar, “Cloud data protection for the masses,” Computer, vol. 45, no. 1, pp. 39–45, Jan. 2012.

[3] A. Gionis, P. Indyk, and R. Motwani, “Similarity search in high dimensions via hashing,” in Proc. VLDB, 1999, pp. 518–529.

[4] C. Bösch, P. Hartel, W. Jonker, and A. Peter, “A survey of provably secure searchable encryption,” ACM Comput. Surveys, vol. 47, no. 2, pp. 1–51, Jan. 2015.

[5] R. Kumar, J. Novak, B. Pang, and A. Tomkins, “On anonymizing query logs via token-based hashing,” in Proc. 16th Int. Conf. World Wide Web, 2007, pp. 629–638.

[6] M. S. Islam, M. Kuzu, and M. Kantarcioğlu, “Access pattern disclosure in searchable encryption: Ramification, attack and mitigation,” in Proc. NDSS, vol. 20, Feb. 2012, p. 12.

[7] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, “Leakage-abuse attacks against searchable encryption,” in Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2015, pp. 668–679.

[8] Y. Zhang, J. Katz, and C. Papamanthou, “All your queries are belong to us: The power of file-injection attacks on searchable encryption,” in Proc. Usenix Secur., 2016, pp. 707–720.

[9] D. Boneh, C. G. Di, R. Ostrovsky, and G. Persiano, “Public key encryption with keyword search,” in Proc. Int. Conf. Theory Appl. Cryptogr. Techn., 2004, pp. 506–522.

[10] D. Xiaoling Song, D. Wagner, and A. Perrig, “Practical techniques for searches on encrypted data,” in Proc. IEEE Symp. Secur. Privacy, May 2000, pp. 44–55.

[11] P. Golle, J. Staddon, and B. Waters, “Secure conjunctive keyword search over encrypted data,” in Proc. IEEE In. Conf. Appl. Cryptogr. Netw. Secur., Jan. 2004, pp. 31–45.

[12] Y. H. Hwang and P. J. Lee, “Public key encryption with conjunctive keyword search and its extension to a multi-user system,” in Proc. Int. Conf. Pairing-Based Cryptogr., 2007, pp. 2–22.

[13] P. Wang, H. Wang, and J. Pieprzyk, “Common secure index for conjunctive keyword-based retrieval over encrypted data,” in Proc. Workshop Secure Data Manage., 2007, pp. 108–123.

[14] P. Wang, H. Wang, and J. Pieprzyk, “An efficient scheme of common secure indices for conjunctive keyword-based retrieval on encrypted data,” in Proc. Int. Workshop Inf. Secur. Appl., 2008, pp. 145–159.

[15] P. Wang, H. Wang, and J. Pieprzyk, “Keyword field-free conjunctive keyword searches on encrypted data and extension for dynamic groups,” in Proc. Int. Conf. Cryptol. Netw. Secur., 2008, pp. 178–195.

[16] F. Bao, R. H. Deng, X. Ding, and Y. Yang, “Private query on encrypted data in multi-user settings,” in Proc. Int. Conf. Inf. Secur. Pract. Exper. Berlin, Germany: Springer, 2008, pp. 71–85.

[17] W. Sun, S. Yu, W. Lou, Y. T. Hou, and H. Li, “Protecting your right: Verifiable attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud,” IEEE Trans. Parallel Distrib. Syst., vol. 27, no. 4, pp. 1187–1198, Apr. 2016.

[18] R. A. Popa and N. Zeldovich, “Multi-key searchable encryption,” JACR Cryptol. ePrint Arch., pp. 1–18, Sep. 2013.

[19] A. Kiayias, O. Okuz, A. Russell, Q. Tang, and B. Wang, “Efficient encrypted keyword search for multi-user data sharing,” in Proc. Eur. Symp. Res. Secur., 2016, pp. 173–195.

[20] S. F. Sun, J. K. Liu, A. Sakzad, R. Steinfeld, and T. H. Yu, “An efficient non-interactive multi-client searchable encryption with support for Boolean queries,” in Proc. Eur. Symp. Res. Secur. Comput. Secur., 2016, pp. 154–172.

[21] H. Cui, X. Yuan, and C. Wang, “Harnessing encrypted data in cloud for secure and efficient image sharing from mobile devices,” in Proc. IEEE Conf. Comput. Commun. (INFOCOM), Apr. 2015, pp. 2659–2667.

[22] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, “Fuzzy keyword search over encrypted data in cloud,” in Proc. INFOCOM, 2010, pp. 1–5.

[23] M. Kuzu, M. S. Islam, and M. Kantarcioğlu, “Efficient similarity search over encrypted data,” in Proc. IEEE 28th Int. Conf. Data Eng., Apr. 2012, pp. 1156–1167.

[24] X. Yuan, X. Wang, C. Wang, A. Squicciarini, and K. Ren, “Enabling privacy-preserving image-centric social discovery,” in Proc. IEEE 34th Int. Conf. Distrib. Comput. Syst., Jun. 2014, pp. 198–207.

[25] X. Yuan, X. Wang, C. Wang, C. Yu, and S. Nutanong, “Privacy-preserving similarity joins over encrypted data,” IEEE Trans. Inf. Forensics Security, vol. 12, no. 11, pp. 2763–2775, Nov. 2017.

[26] M. Strizhov and I. Ray, “Multi-keyword similarity search over encrypted cloud data,” in Proc. IFIP Int. Inf. Secur. Conf. Berlin, Germany: Springer, 2014, pp. 52–65.

[27] H. Cui, X. Yuan, Y. Zheng, and C. Wang, “Enabling secure and effective near-duplicate detection over encrypted in-network storage,” in Proc. 35th Annu. IEEE Int. Conf. Comput. Commun., Apr. 2016, pp. 1–9.

[28] C. Hahn, H. J. Shin, H. Kwon, and J. Hur, “Efficient multi-user similarity search over encrypted data in cloud storage,” Wireless Pers. Commun., vol. 107, no. 3, pp. 1–17, 2018.
[29] B. Lynn. (2010). The Pairing-Based Cryptography (PBC) Library. [Online]. Available: http://crypto.stanford.edu/pbc

[30] INRIA Copydays. Accessed: Jan. 20, 2022. [Online]. Available: http://lear.inrialpes.fr/people/jegou/data.php#copydays

[31] Y. Ke, R. Sukthankar, L. Huston, Y. Ke, and R. Sukthankar, “Efficient near-duplicate detection and sub-image retrieval,” in Proc. ACM Multimedia, 2004, p. 5.

[32] Python Software Foundation. Accessed: Jan. 20, 2022. [Online]. Available: https://pypi.python.org/pypi/ImageHash

[33] (2017). VoCore2. [Online]. Available: http://vocore.io/v2.html

[34] E. Trichina and A. Bellezza, “Implementation of elliptic curve cryptography with built-in counter measures against side channel attacks,” in Proc. Int. Workshop Cryptogr. Hardw. Embedded Syst., 2002, pp. 98–113.

[35] A. Boldyreva, V. Goyal, and V. Kumar, “Identity-based encryption with efficient revocation,” in Proc. 15th ACM Conf. Comput. Commun. Secur., 2008, pp. 417–426.

[36] S. Garg, P. Mohassel, and C. Papamanthou, “TWORAM: Round-optimal oblivious RAM with applications to searchable encryption,” IACR Cryptol. ePrint Arch., vol. 2015, p. 1010, Sep. 2015.

[37] E. Stefanov, C. Papamanthou, and E. Shi, “Practical dynamic searchable encryption with small leakage,” in Proc. Netw. Distrib. Syst. Secur. Symp., vol. 14, 2014, pp. 23–26.

[38] R. Bost, “$\Sigma_{\text{sign}}$—Forward secure searchable encryption,” in Proc. ACM CCS, vol. 16, 2016, pp. 1143–1154.

[39] R. Bost, B. Minard, and O. Ohrimenko, “Forward and backward private searchable encryption from constrained cryptographic primitives,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2017, pp. 1465–1482.

[40] K. S. Kim, M. Kim, D. Lee, J. H. Park, and W.-H. Kim, “Forward secure dynamic searchable symmetric encryption with efficient updates,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2017, pp. 1449–1463.

[41] L. Chen, P. Morrissey, and N. P. Smart, “Pairings in trusted computing,” in Proc. 2nd Int. Conf. Pairing-Based Cryptogr., vol. 5209. Cham, Switzerland: Springer, 2008, pp. 1–17.

HYUNSOO KWON received the B.S. degree from Chung-Ang University, Seoul, South Korea, in 2014, and the M.S. and Ph.D. degrees from Korea University, Seoul, in 2016 and 2020, respectively, all in computer science. He is currently a Senior Researcher with Samsung Electronics, South Korea. His research interests include information security, network security, and cloud computing security.

CHANGHEE HAHN received the B.S. and M.S. degrees in computer science from Chung-Ang University, Seoul, South Korea, in 2014 and 2016, respectively, and the Ph.D. degree from the Department of Computer Science and Engineering, College of Informatics, Korea University, South Korea, in 2020. He was a Postdoctoral Researcher at Korea University, from 2020 to 2021. He is currently an Assistant Professor with the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul. His research interests include information security and cloud computing security.