International Journal of Web Services Research
Volume 20 • Issue 1

Secure Cloud Storage and Retrieval of Personal Health Data From Smart Wearable Devices With Privacy-Preserving Techniques

Zhuolin Mei, Jiujiang University, China
Jing Yu, Jiujiang University, China
Jinzhou Huang, Hubei University of Arts and Science, China*
Bin Wu, Jiujiang University, China
Zhiqiang Zhao, Ningxia Normal University, China
Caicai Zhang, Zhejiang Institute of Mechanical and Electrical Engineering, China
Jiaoli Shi, Jiujiang Key Laboratory of Network and Information Security, China
Xiancheng Wang, Jiujiang University, China
Zongda Wu, Shaoxing University, China

ABSTRACT

With the increasing awareness of personal health, personal health data management has become an important part of people's lives. Smart wearable devices (SWDs) collect people's personal health data, and then store the data on cloud. Authorized entities access the data to provide personalized health services. However, these personal health data contain a large amount of sensitive information, which may pose a significant threat to people's lives and property. To address this, this paper proposes a privacy-preserving solution. SWD data is encrypted, and secure indexes are created using Bloom filter and 0-1 encoding. Encrypted data and indexes are stored in a semi-trusted cloud. Only authorized entities can access the ciphertexts, ensuring secure personalized health management. Extensive experiments validate the scheme's efficiency in index construction, query token generation, and ciphertext search. Security analysis confirms no external entity, including the cloud, gains additional information during retrieval.

KEYWORDS

Cloud Storage, Data Retrieval, Personal Health Data, Privacy Protection, Smart Wearable Device

INTRODUCTION

With the increasing public health awareness, personal health data management is becoming an important component in people's lives (Zeng et al., 2015). Smart wearable devices (SWDs), such as smartwatches, smart bracelets, etc., can collect motion data (such as motion trajectory and status, etc.) and physiological data (such as heart rate and blood pressure, etc.) of the SWD wearer (SWDW)

DOI: 10.4018/IJWSR.331388

*Corresponding Author

anytime and anywhere. In a personal health data management, the data collected by SWDs are typically organized into structured records. For example, data collected by SWDs show that on June 16, 2023, at 8:15 a.m., the SWDW's heart rate was 80 while located at 31°N, 114°E. This information can be represented as a record <2023, 6, 16, 08, 15, 1, 114, 0, 31, 80>, where 1 represents east longitude and 0 represents north latitude. However, due to the small storage space of SWDs and the risk of accidental damage or loss, the collected data are often automatically transmitted to the paired smartphone through a Bluetooth connection and then uploaded to the cloud to obtain unlimited storage space and use the cloud's data backup and disaster recovery mechanisms to ensure that the data are permanently available. In addition, when the data collected by SWDs are uploaded to the cloud as part of electronic health records, medical institutions, insurance companies, or other health management institutions can access and use the data to provide more personalized health management services (Zeng et al., 2018). However, the data collected by SWDs contain sensitive information about the SWDWs. Once this sensitive information is leaked, it may affect personal image, property safety, and even life safety. Therefore, ensuring the security and privacy of sensitive information collected by SWDs and outsourced to the cloud is very important. Encryption is an effective solution to protect the data collected by SWDs. However, traditional encryption methods (e.g., block encryption) cannot support the most common data operations in the cloud, such as ciphertext retrieval (Cui et al., 2023). Although new ciphertext retrieval schemes have been proposed, they have limitations when applied to personal health data management.

The searchable encryption method for numerical data is widely recognized as one of the most effective methods for securely storing and retrieving numerical data (such as time, location, and heart rate collected by SWDs). Efficient searchable encryption methods for numerical data primarily fall into two categories: order-preserving encryption and bucket schemes. Agrawal et al. (2004) propose an order-preserving encryption (OPE) scheme. The main idea of an OPE scheme is to embed order information into the ciphertexts so that the order of the ciphertexts is consistent with that of the plaintexts. Specifically, for any data x>y, it holds Enc(x)>Enc(y), where Enc denotes the encryption algorithm in an OPE scheme. Therefore, an OPE scheme can be used by the cloud to support efficient range queries on ciphertexts. However, most current OPE schemes (Agrawal et al., 2004; Peng et al., 2017; Popa et al., 2011; Quan et al., 2018) only support queries on the ciphertexts of single-dimensional data, and queries on the ciphertexts of personal health data are rarely involved (Zhan et al., 2022). Additionally, since the order information of ciphertexts is revealed in an OPE scheme, attackers can use this information to accurately infer the corresponding plaintexts, leading to potential data security issues (David & Nagaraja, 2004). Another kind of effective method for securely storing and retrieving numerical data is a bucketization scheme (Wang & Ravishankar, 2013; Hore et al., 2004; Hore et al., 2012). In a bucketization scheme, data are divided into multiple buckets, and all data within a bucket are treated as a single unit. A secure encryption scheme is then used to encrypt all the data within each bucket, making the ciphertexts within the same bucket indistinguishable and effectively protecting the order information between them. During ciphertext querying, if the query range intersects with a certain bucket, all the ciphertexts within that bucket are returned as the query result. The number of ciphertexts within each bucket can be adjusted to balance the security of the order information and the accuracy of the query result. To further improve the efficiency of the bucketization scheme, bucketization-based index schemes have been proposed (Wang & Ravishankar, 2013; Mei et al., 2018). However, the scheme in the reference (Wang & Ravishanka, 2013) uses complex matrix calculations, which is not efficient enough. The scheme in the reference (Mei et al., 2018) requires building a tree index (each internal node has n child nodes) over the buckets and works well only for uniformly distributed datasets.

To address the limitations of previous schemes, we propose a Privacy-Preserving Storage and Retrieval Method for Personal Health Data (PPSRMPHD). Our method involves constructing binary trees for the collected data. The security of the binary trees can be ensured by using 0-1 encoding (Gupta & McKeown, 2001) and Bloom filter (Bloom, 1970) techniques. Additionally, SWDs generate query tokens for authorized entities (such as doctors, medical institutions, insurance companies, and other health management organizations) based on their query conditions. These entities can only obtain the query tokens from SWDs after being authorized by the SWDWs. After obtaining the query tokens, these entities send them to the cloud to perform secure and efficient searches on the secure binary tree indexes, and retrieve the search results. We summarize our proposed PPSRMPHD scheme in three main aspects: (1) We construct secure binary tree indexes by using 0-1 encoding and Bloom filter techniques. (2) Based on secure binary tree indexes, we propose a PPSRMPHD scheme. (3) We conduct comprehensive experiments and analyze the correctness and security of the PPSRMPHD in detail.

RELATED WORK

Agrawal et al. (2004) first proposed an Order-Preserving Encryption (OPE) scheme, which preserves the partial order between plaintexts in ciphertexts. This property of OPE enables efficient comparison and searching of ciphertexts without decryption. However, the scheme lacks a formal security definition or analysis. Boldyreva et al. (2009) later proposed a rigorous security definition for OPE, which requires the scheme not to leak any information other than the partial order relation. They proved that no OPE scheme can satisfy this definition and proposed a weaker one, which requires ciphertexts to be indistinguishable from random increment function values. Based on their work, many related studies have been conducted (Boldyreva et al., 2011; Dyer et al., 2017; Krendelev et al., 2014; Teranishi et al., 2014; Xiao & Yen, 2012), but most only consider single-dimensional data and ignore the large amount of multi-dimensional data in the real world. Recently, Zhan et al. (2022) proposed an efficient Multi-Dimensional Order-Preserving Encryption scheme (MDOPE) which uses a network data structure and prefix encoding and Bloom filter techniques to enable queries on encrypted multi-dimensional data.

Bucketization schemes (Hore et al., 2004; Hore et al., 2012) are proposed to reduce order information leakage in OPE schemes. In a bucketization scheme, data are grouped into buckets, and all data within a bucket are treated as a whole. When a query matches a bucket, all data within that bucket are retrieved as the query results. Since the data within each bucket cannot be distinguished, the ordering information within each bucket has been protected. The first bucketization scheme is proposed in Hacigümüş et al. (2002), and subsequent research aims to improve bucketization schemes' query security and efficiency (Hore et al., 2004; Lee, 2014). Lee (2014) proposed an ordered bucketization scheme to improve the search efficiency by organizing all buckets in an ordered manner (i.e., the data in a bucket are smaller than the data in another bucket).

Previous bucketization schemes (Hore et al., 2004; Hore et al., 2012; Hacigümüş et al., 2002) required local storage and search of buckets on the user side, which is inconvenient. To address this issue, Wang and Ravishankar (2013) proposed using Asymmetric Scalar-Product Preserving Encryption (ASPPE) (Wong et al., 2009) to process the buckets and construct a secure index called $\hat{R}$-tree, which can be stored and searched on a remote cloud server. Similarly, Mei et al. (2018) built a tree index (each internal node has n child nodes, and each leaf node is associated with a bucket) to support range queries over encrypted personal health data. However, their scheme works well only for uniformly distributed datasets.

PRELIMINARIES

A Bloom filter (Bloom, 1970) is a probabilistic data structure used to test whether an element is a member of a set. It consists of three components, including (i) a bit array A of n bits, (ii) k independent hash functions $h_1, h_2, \ldots, h_k$, where $h_i : \{0,1\}^* \rightarrow [1,n]$ and $i \in [1,k]$, and (iii) a dataset

$D = \{d_1, d_2, \ldots, d_m\}$ which contains m different data. Given a data d', the Bloom filter can judge whether $d' \in D$ or $d' \notin D$ by using the following method:

1. The Bloom filter initializes all the bits of the bit array A to 0.
2. To add an element d_j to the Bloom filter, the filter calculates the hash value $h_i(d_j)$ and sets the bit at position $h_i(d_j)$ in the bit array A to 1, where $i \in [1,k]$ and $j \in [1,m]$.
3. To test whether an element d' is in the dataset D, the Bloom filter calculates the hash value $h_i(d')$ for each of the k independent hash functions, where $i \in [1,k]$. If the bit at position $h_i(d')$ in the bit array A is 1, the data d' is considered to be a member of the dataset D. However, if any of the bits at these positions are 0, then d' is not in D.

Given an integer a, its 0-1 encoding (Lin & Tzeng, 2005) is defined as two sets S_a^0 and S_a^1 respectively, where $S_a^0 = \{a_n a_{n-1} \cdots a_{i+1} 1 \mid a_i = 0, 1 \le i \le n\}$ and $S_a^1 = \{a_n a_{n-1} \cdots a_i \mid a_i = 1, 1 \le i \le n\}$. Given two integers x and y, the 0-1 encoding can be used to determine whether x or y is greater than the other. Specifically, $S_x^1 \cap S_y^0 \neq \varnothing \Leftrightarrow x > y$ and $S_x^1 \cap S_y^0 = \varnothing \Leftrightarrow x \le y$. These conclusions have been proven by Lin and Tzeng (2005). For example, 5 and 4 are two integers. The binary value of 5 is $(0101)_2$ and the binary value of 4 is $(0100)_2$. The 0-1 encoding sets of 5 and 4 can be easily calculated $S_5^0 = \{1, 011\}$, $S_5^1 = \{01, 0101\}$, $S_4^0 = \{1, 011, 0101\}$ and $S_4^1 = \{01\}$. As $S_5^1 \cap S_4^0 \neq \varnothing$, there is $5 > 4$. On the contrary, as $S_4^1 \cap S_5^0 = \varnothing$, there is $4 \le 5$.

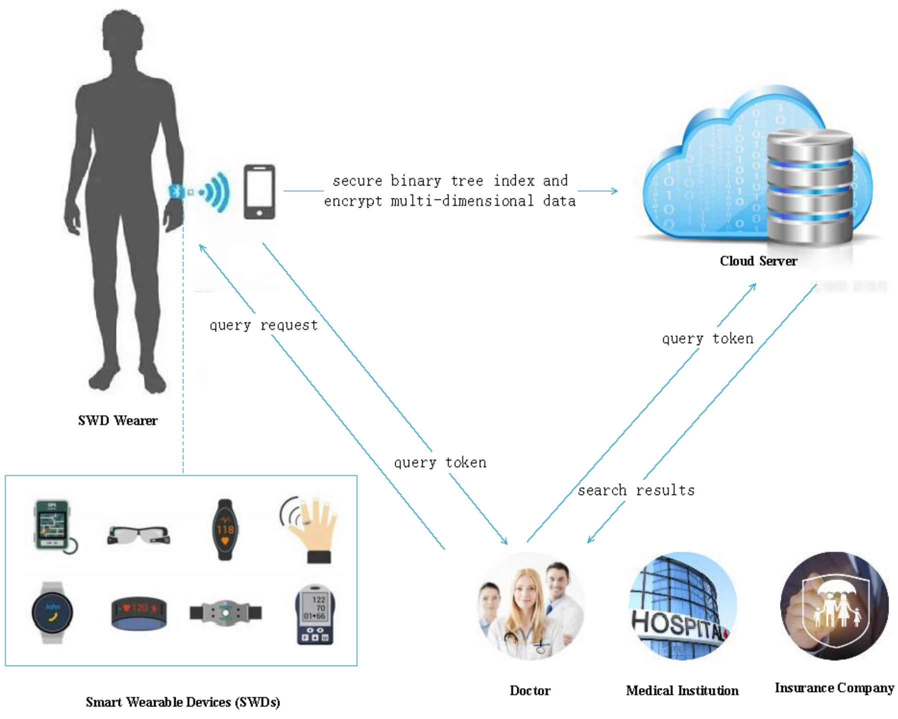
SYSTEM MODEL

Our system model involves four parties (as shown in Fig. 1). (1) SWDW, who wears a SWD, which collects personal health data from the SWDW. Then, the SWD transmits the collected data to the paired smartphone through a Bluetooth connection. (2) The paired smartphone, which encrypts the collected data, builds secure binary tree indexes and, finally, outsources the ciphertexts and secure binary tree indexes to the cloud. (3) A certain entity (such as a medical institution, doctor, insurance company, or other health management organization), who wants to access the SWDW's personal health data. The entity requests a query token for his or her query request, and finally sends the query token to the cloud to retrieve personal health data. (4) Semi-trusted cloud, which follows designated protocols and procedures. The cloud performs ciphertext retrieval and sends search results to the entity. After receiving the ciphertexts, the entity decrypts the ciphertexts to obtain the SWDW's personal health data.

Definition 1 – Correctness: Given a query range Q, the cloud performs data retrieval using the secure binary tree indexes and returns the search results $C^* = \{C_1^*, C_2^*, \ldots, C_i^*\}$. For each search result C_j^* ($1 \le j \le i$), the scheme is considered correct if the decryption data d_j of C_j^* falls in the query range Q.

Definition 2 - Security (Zhan et al., 2022; Guo et al., 2018): Given a leakage function F, if all adversaries A are unable to reveal more information than the leakage function F, then the scheme is considered secure. The leakage function is defined as $F(u,v) = position_{diff}(u,v)$, where $position_{diff}(u,v)$ returns the position of the first difference between u and v.

Figure 1. System Model



For the purpose of brevity, we use abbreviations for several frequently appearing terms in the rest of this paper: SWD is short for smart wearable device; SWDW is short for smart wearable device wearer; ENT is short for the entity such as doctor, medical institution, insurance company, and other health management organization.

CONSTRUCTION OF THE PRIVACY-PRESERVING STORAGE AND RETRIEVAL METHOD FOR PERSONAL HEALTH DATA (PPSRMPHD)

This section first provides an overview of the construction of the PPSRMPHD and then provides a detailed description of the processes for key generation, encoding, index construction, token generation, search, and decryption algorithms.

The proposed PPSRMPHD scheme enables secure and fast query for encrypted personal health data. First, the SWD collects personal health data from a SWDW, converts these data into table records, and stores these table records in a table. Then, the SWD creates a binary tree for each column in the table. Next, the SWD runs the index construction algorithm to process the binary trees using 0-1 encoding and Bloom filter techniques to obtain secure binary tree indexes. Additionally, the SWD needs to encrypt all the table records in the table by using a secure encryption scheme. Finally, the SWD stores the encrypted data and secure binary tree indexes in the cloud. When a query request from an ENT is received, the SWDW decides whether or not to generate a query token for the ENT. If the SWDW agrees to generate a query token for the ENT, the SWDW executes the token generation algorithm. This algorithm uses 0-1 encoding and hash functions from the Bloom filter to process the query request and generate a query token. Upon receiving the query token, the cloud performs data retrieval over the secure binary tree indexes in a top-down manner and returns the ciphertexts to the ENT. Finally, the ENT decrypts the ciphertexts in the search results using the decryption key to obtain the SWDW's personal health data.

Please note that there are many other encryption schemes that can be used to encrypt the personal health data of a SWDW. However, the focus of this paper is on the security and efficient query for encrypted personal health data in the cloud. Therefore, in order to clearly describe the PPSRMPHD scheme, we only use a simple symmetric encryption algorithm (such as AES), denoted by SE. Thus, during the decryption process, the ENT needs to request the decryption key from the SWDW, which is also the encryption key used to encrypt the SWDW's personal health data.

Key Generation Algorithm - $KeyGen(1^\lambda) \to SK$: The algorithm takes a security parameter λ as input and calculates a secret key SK as output. The algorithm is executed by the SWD. The details of the algorithm are as follows:

1. The algorithm $KeyGen$ adopts a secure encryption scheme $SE = (SE.Gen, SE.Enc, SE.Dec)$. $KeyGen$ recalls $SE.Gen$ to generate the first part of the secret key $sk_1 = SE.Gen(1^\lambda)$, which is used to encrypt all the personal health data collected by the SWD.
2. The algorithm $KeyGen$ chooses k pseudo-random seeds $sd_1, sd_2, \ldots, sd_k$ as the second part of the secret key $sk_2 = (sd_1, sd_2, \ldots, sd_k)$, which is used in the hash functions of the Bloom filter. Each hash function in the Bloom filter takes a value and a pseudo-random seed as input, and outputs a hash value. The hash value outputted by a hash function cannot be correctly calculated without $sk_2 = (sd_1, sd_2, \ldots, sd_k)$. Hence, the security of secure binary tree indexes can be ensured by this type of hash functions in the Bloom filter. For the same reason, the security of query tokens can also be ensured by this type of hash functions.
3. Finally, the algorithm $KeyGen$ outputs $SK = (sk_1, sk_2)$ as the secret key.

Encoding Algorithm - $Encoding(sv) \to C_{sv}$: The algorithm takes a split value sv as input, and outputs the encoding of sv, denoted by C_{sv}. The algorithm is a sub-algorithm of index construction algorithm $IndexGen$ (see the following paragraphs). The details of the algorithm $Encoding$ are as follows:

1. The algorithm $Encoding$ calculates the binary value of sv, denoted by c_{sv}. The length of c_{sv} is l, which is a sufficiently large positive integer. If the length of c_{sv} is less than l, the algorithm $Encoding$ pads 0s at the high-order positions of c_{sv}.
2. For the security concern, the algorithm $Encoding$ randomizes the binary value c_{sv}. Specifically, the algorithm $Encoding$ randomly selects a number r and calculates its binary value, denoted by c_r. The length of c_r is also l. Then, the algorithm $Encoding$ pads c_r after c_{sv}, i.e., $c_{sv} \mid c_r$, where $\mid$ represents the concatenation operation of two binary values.
3. The algorithm $Encoding$ creates a bit array A_{sv}. Each bit in A_{sv} is initialized to 0.
4. For data comparison, the algorithm $Encoding$ calculates the 0-encoding of $c_{sv} \mid c_r$, denoted by $S^0_{c_{sv}|c_r}$. Then, the algorithm $Encoding$ uses the hash functions $h_1, h_2, \ldots, h_k$ in the Bloom filter and the second part secret key $sk_2 = (sd_1, sd_2, \ldots, sd_k)$ to process the binary values in $S^0_{c_{sv}|c_r}$. Specifically, the algorithm $Encoding$ calculates hash values:

$$V_{sv} = \{h_1(s, sd_1), h_2(s, sd_2), \ldots, h_k(s, sd_k) \mid s \in S^0_{c_{sv}|c_r}\}$$

and then sets the bit at $v \in A_{sv}$ position of A_{sv} to 1.

5. The algorithm $Encoding$ outputs C_{sv} that $C_{sv} = A_{sv}$.

Example 1: Suppose the split value is 5, its 4-bit binary value is $(0101)_2$. If the random number is 8, its 4-bit binary value is $(1000)_2$. After padding, the binary value is $(0101\ 1000)_2$. Its 0-type encoding is $\{1,\ 011,\ 0101\ 11,\ 0101\ 101,\ 0101\ 1001\}$. Then, the algorithm $Encoding$ uses the hash functions $h_1, h_2, \ldots, h_k$ in the Bloom filter and the second part secret key $sk_2 = (sd_1, sd_2, \ldots, sd_k)$ to process the binary values in $\{1,\ 011,\ 0101\ 11,\ 0101\ 101,\ 0101\ 1001\}$, and then can obtain a set of hash values. Next, the algorithm $Encoding$ sets these hash values positions in a bit array to 1. Finally, the bit array is as the output of $Encoding(5)$.

Index Construction Algorithm: $Index(T) \rightarrow T^*$: The algorithm takes a binary tree T as input and constructs a secure binary tree index T^* as output. The algorithm is executed by the SWD.

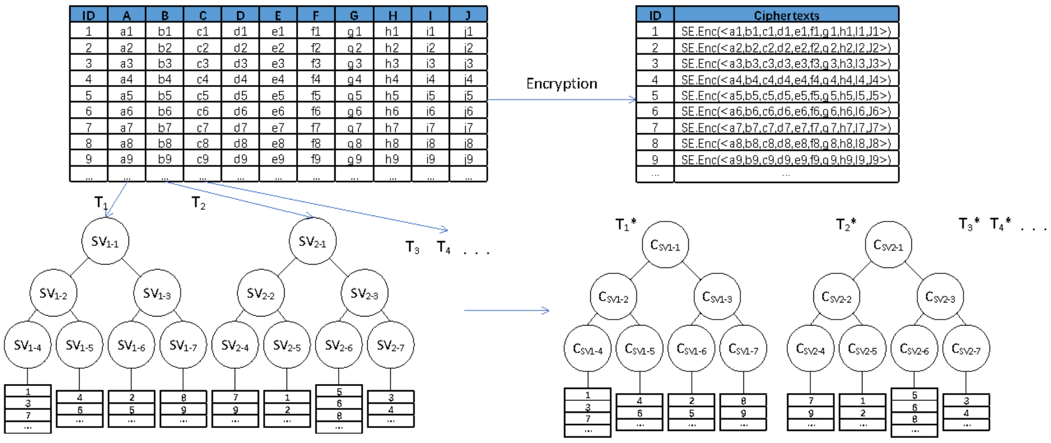
Before constructing secure binary tree indexes, the SWD needs to do some preprocessing works. First, the SWD converts all the collected personal health data into table records and then stores them in a table. Second, the SWD adds a new column to the table and assigns a unique identification (ID) for each table record. Then, the SWD adds these IDs to the new column. Third, for each column (except for the column which stores IDs) in the table, the SWD constructs a binary tree. Each node of a binary tree contains a split value. Assuming that the binary tree on a certain column is T, if a table record has a value in that column that is the same as the split value of a leaf node in T, then the ID of this table record is associated with the leaf node in T. The algorithm $Index$ recalls the sub-algorithm $Encoding$ to process all the split values of nodes in T. Then, the algorithm $Index$ can obtain the secure binary tree index, denoted by T^*.

For the sake of illustration, we assume that this table has n columns (except for the column which stores IDs). The binary trees on these columns are $T_1, T_2, \ldots, T_n$ respectively. The SWD executes the algorithm $Index$ n times for $T_1, T_2, \ldots, T_n$ respectively. Then, the SWD can obtain the secure binary tree indexes $T_1^*, T_2^*, \ldots, T_n^*$. Next, the SWD recalls the algorithm $SE.Enc$ to encrypt all the table records. Finally, the SWD stores the ciphertexts of these table records and $T_1^*, T_2^*, \ldots, T_n^*$ in the cloud. We use Example 2 to illustrate the execution process above.

Example 2: As shown in Figure 2, the SWD organizes all the collected personal health data into table records and stores them in a table. Then, the SWD adds a new column to the table to record the identification of each table record (as shown in the left-hand table of Figure 2). Next, the SWD uses the secure encryption algorithm $SE.Enc$ to encrypt all the table records (as shown in the right-hand table of Figure 2) and uploads the ciphertexts and their identifications to the cloud. Subsequently, for each column in the original table (excluding the added column for identifications), the SWD creates a binary tree. Each node in these binary trees contains a split value, and each leaf node is associated with a set of identifications. In a certain column, if a table record's value is equal to the split value of a leaf node in the binary tree, the identification of that table record is stored in the set that is associated with that leaf node. Furthermore, the SWD uses the algorithm $Index$ to process all the binary trees $T_1, T_2, \ldots$ (as shown in the left-hand side binary trees of Figure 2) and generate secure binary tree indexes $T_1^*, T_2^*, \ldots$ (as shown in the right-hand side secure binary tree indexes of Figure 2). Finally, the SWD uploads these secure binary tree indexes to the cloud.

Token Generation Algorithm - $Token(SK, Q) \rightarrow token_Q$: The algorithm takes the secret key SK and a queried request Q as inputs. It calculates the query token $token_Q$ of Q as output. The

Figure 2. Secure binary tree indexes construction and personal health data encryption



ID	A	B	C	D	E	F	G	H	I	J
1	a1	b1	c1	d1	e1	f1	g1	h1	i1	j1
2	a2	b2	c2	d2	e2	f2	g2	h2	i2	j2
3	a3	b3	c3	d3	e3	f3	g3	h3	i3	j3
4	a4	b4	c4	d4	e4	f4	g4	h4	i4	j4
5	a5	b5	c5	d5	e5	f5	g5	h5	i5	j5
6	a6	b6	c6	d6	e6	f6	g6	h6	i6	j6
7	a7	b7	c7	d7	e7	f7	g7	h7	i7	j7
8	a8	b8	c8	d8	e8	f8	g8	h8	i8	j8
9	a9	b9	c9	d9	e9	f9	g9	h9	i9	j9
...	...	...	...	...	...	...	...	...	...	...

ID	Ciphertexts
1	SE.Enc(<a1,b1,c1,d1,e1,f1,g1,h1,I1,J1>)
2	SE.Enc(<a2,b2,c2,d2,e2,f2,g2,h2,I2,J2>)
3	SE.Enc(<a3,b3,c3,d3,e3,f3,g3,h3,I3,J3>)
4	SE.Enc(<a4,b4,c4,d4,e4,f4,g4,h4,I4,J4>)
5	SE.Enc(<a5,b5,c5,d5,e5,f5,g5,h5,I5,J5>)
6	SE.Enc(<a6,b6,c6,d6,e6,f6,g6,h6,I6,J6>)
7	SE.Enc(<a7,b7,c7,d7,e7,f7,g7,h7,I7,J7>)
8	SE.Enc(<a8,b8,c8,d8,e8,f8,g8,h8,I8,J8>)
9	SE.Enc(<a9,b9,c9,d9,e9,f9,g9,h9,I9,J9>)
...	...

SWD runs the algorithm $Token$ to generate the query token $token_Q$, and sends $token_Q$ to the ENT. The details of the algorithm $Token$ are as follows:

1. The ENT sends the query range Q to SWDW. In formal terms, the query range Q is $[p_{i_1}, q_{i_1}] \times [p_{i_2}, q_{i_2}] \times \ldots \times [p_{i_k}, q_{i_k}]$, where $[p_{i_j}, q_{i_j}]$ is the query range on the i_j-th column, $1 \le i_1 < i_2 < \ldots < i_k \le n$ and n is the maximum number of columns in the table. The query range Q represents ENT wants to search for the SWDW's personal health data within the range $[p_{i_1}, q_{i_1}]$ in the i_1-th column, the range $[p_{i_2}, q_{i_2}]$ in the i_2-th column, …, the range $[p_{i_k}, q_{i_k}]$ in the i_k-th column simultaneously.
2. If the SWDW agrees to the ENT's query range Q, then the SWD executes the algorithm $Token$ to generate the query token $token_Q$.
3. For each range $[p,q] \in \{[p_{i_1}, q_{i_1}], [p_{i_2}, q_{i_2}], \ldots, [p_{i_k}, q_{i_k}]\}$, the algorithm $Token$ encodes the lower limit p to its binary value, denoted by c_p. Then, the algorithm $Token$ randomly selects a number r_p and calculates its binary value, denote by c_{r_p}. Note that, c_p and c_{r_p} have the same length l, where l is a sufficiently large positive integer. If the length of c_p and c_{r_p} is less than l, the algorithm $Token$ pads 0s at the high-order positions of c_p and c_{r_p}. The algorithm $Token$ pads c_{r_p} after c_p, i.e., $c_p \mid c_{r_p}$, where $\mid$ represents the concatenation operation of two binary values. By using the same method, the algorithm $Token$ converts the upper limit q to $c_q \mid c_{r_q}$, where c_q is the binary value of q and c_{r_q} is the binary value of the random number r_q.
4. The algorithm $Token$ calculates the 1-encoding of $c_p \mid c_{r_p}$, denoted by $S^1_{c_p|c_{r_p}}$. Then, the algorithm $Token$ uses the hash functions $h_1, h_2, \ldots, h_k$ in the Bloom filter and the second part secret key $sk_2 = (sd_1, sd_2, \ldots, sd_k)$ to process the binary values in $S^1_{c_p|c_{r_p}}$. Specifically, the algorithm $Token$ calculates $t_p = \{< h_1(s, sd_1), h_2(s, sd_2), \ldots, h_k(s, sd_k) > \mid s \in S^1_{c_p|c_{r_p}}\}$. By

using the same method, the algorithm $Encoding$ calculates $t_q = \{< h_1(s, sd_1), h_2(s, sd_2), \ldots, h_k(s, sd_k) >| s \in S^1_{c_q|c_{r_q}}\}$.

5. For the query range $Q = [p_{i_1}, q_{i_1}] \times [p_{i_2}, q_{i_2}] \times \ldots \times [p_{i_k}, q_{i_k}]$, the algorithm $Token$ can calculate the query token $token_Q$ of Q, which is $token_Q = \{< t_{p_{i_1}}, t_{q_{i_1}} >, < t_{p_{i_2}}, t_{q_{i_2}} >, \ldots, < t_{p_{i_k}}, t_{q_{i_k}} >\}$. The SWDW sends the query token $token_Q$ to the entities.

Search Algorithm - $Search(token_Q, \{T_1^*, T_2^*, \ldots, T_n^*\}) \to I^*$: The algorithm takes a query token $token_Q$ and the secure binary tree indexes $\{T_1^*, T_2^*, \ldots, T_n^*\}$ as inputs. It outputs the retrieved ciphertexts as search results I^*. The cloud executes the algorithm $Search$ to retrieve ciphertexts and then returns them to the ENT as response. The specific steps of the query process are as follows:

1. After receiving the query token $token_Q$ from the ENT, the algorithm $Search$ extracts each sub-token from $token_Q$. We suppose $< t_p, t_q >$ is a sub-token, where:

$$< t_p, t_q > \in token_Q = \{< t_{p_{i_1}}, t_{q_{i_1}} >, < t_{p_{i_2}}, t_{q_{i_2}} >, \ldots, < t_{p_{i_k}}, t_{q_{i_k}} >\}$$

and T^* is a secure binary tree index, where $T^* \in \{T_1^*, T_2^*, \ldots, T_n^*\}$. Note that, the sub-token $< t_p, t_q >$ and security binary tree index T^* must satisfy that $< t_p, t_q >$ and T^* correspond to the same column of the table.

To provide a clear and detailed introduction to the query process, we suppose the sub-token $< t_p, t_q >$ represents the range $[p, q]$, the bit array C_{sv} represents the binary encoding of the split value sv, and C_{sv} is stored in a node of T^*. The algorithm $Search$ executes the following steps to determine whether the split value sv is within the range $[p, q]$, or less than the minimum value p of $[p, q]$, or greater than the maximum value q of $[p, q]$. Please note that:

$$t_p = \{< h_1(s, sd_1), h_2(s, sd_2), \ldots, h_k(s, sd_k) >| s \in S^1_{c_p|c_{r_p}}\}$$

$$t_q = \{< h_1(s, sd_1), h_2(s, sd_2), \ldots, h_k(s, sd_k) >| s \in S^1_{c_q|c_{r_q}}\}$$

(see token generation algorithm) and $C_{sv} = A_{sv}$ where A_{sv} is a bit array which is calculated by using the Bloom filter and the values in the set:

$$V_{sv} = \{h_1(s, sd_1), h_2(s, sd_2), \ldots, h_k(s, sd_k) \mid s \in S^0_{c_{sv}|c_r}\}$$

(see encoding algorithm). Based on the characteristics of 0-1 encoding, we can know that (a) if $S^0_{c_{sv}|c_r} \cap S^1_{c_p|c_{r_p}} \neq \varnothing$, there is $sv < p$; (b) if $S^0_{c_{sv}|c_r} \cap S^1_{c_p|c_{r_p}} = \varnothing$ and $S^0_{c_{sv}|c_r} \cap S^1_{c_q|c_{r_q}} \neq \varnothing$, there is $p < sv < q$; (c) if $S^0_{c_{sv}|c_r} \cap S^1_{c_q|c_{r_q}} = \varnothing$, there is $q < sv$. Since the comparison between the range $[p, q]$ and the split value sv has been transformed into the operation of determining whether there is an intersection between several sets, Bloom filters can be used to further process the intersection operation between several sets. Thus, it is easy to know that if $S^0_{c_{sv}|c_r} \cap S^1_{c_p|c_{r_p}} \neq \varnothing$, there exists a tuple in:

$$t_p = \{< h_1(s, sd_1), h_2(s, sd_2), \ldots, h_k(s, sd_k) > | \; s \in S^1_{c_p|c_{r_p}}\}$$

such that all the bits at $h_1(s, sd_1)$, $h_2(s, sd_2)$, …, $h_k(s, sd_k)$ positions of the binary array C_{sv} are 1. Thus, it is easy to know that if there exists a tuple in:

$$t_p = \{< h_1(s, sd_1), h_2(s, sd_2), \ldots, h_k(s, sd_k) > | \; s \in S^1_{c_p|c_{r_p}}\}$$

that all the bits at $h_1(s, sd_1)$, $h_2(s, sd_2)$, …, $h_k(s, sd_k)$ positions of the binary array C_{sv} are 1, there is $S^0_{c_{sv}|c_r} \cap S^1_{c_p|c_{r_p}} \neq \varnothing$, i.e., $sv < p$. Using the same method, the following determinations can be made. If there does not exist a tuple in:

$$t_p = \{< h_1(s, sd_1), h_2(s, sd_2), \ldots, h_k(s, sd_k) > | \; s \in S^1_{c_p|c_{r_p}}\}$$

such that all the bits at $h_1(s, sd_1)$, $h_2(s, sd_2)$, …, $h_k(s, sd_k)$ positions of the binary array C_{sv} are 1, and there exists a tuple in:

$$t_q = \{< h_1(s, sd_1), h_2(s, sd_2), \ldots, h_k(s, sd_k) > | \; s \in S^1_{c_q|c_{r_q}}\}$$

such that all the bits at $h_1(s, sd_1)$, $h_2(s, sd_2)$, …, $h_k(s, sd_k)$ positions of the binary array C_{sv} are 1, there is $p < sv < q$. If there does not exist a tuple in:

$$t_q = \{< h_1(s, sd_1), h_2(s, sd_2), \ldots, h_k(s, sd_k) > | \; s \in S^1_{c_q|c_{r_q}}\}$$

such that all the bits at $h_1(s, sd_1)$, $h_2(s, sd_2)$, …, $h_k(s, sd_k)$ positions of the binary array C_{sv} are 1, there is $q < sv$.

2. The algorithm $Search$ uses the sub-token $< t_p, t_q >$ to perform top-down judgment on the nodes in the security binary tree index T^*. Starting from the root node of T^*, if $sv < p$, the algorithm $Search$ performs the judgment on the right sub-tree; if $p < sv < q$, the algorithm $Search$ performs the judgment simultaneously on the left and right subtrees; if $q < sv$, the algorithm $Search$ performs the judgment on the left sub-tree. By performing layer-by-layer judgment, the leaf nodes of T^* are eventually obtained, and their split values fall within the range $[p, q]$. As each leaf node of T^* is associated with an identification set (see Index construction algorithm), the algorithm $Search$ can obtain an identification set $S_{<t_p, t_q>}$ for the sub-token $< t_p, t_q >$.

3. For the sub-tokens in:

$$token_Q = \{< t_{p_{i_1}}, t_{q_{i_1}} >, < t_{p_{i_2}}, t_{q_{i_2}} >, \ldots, < t_{p_{i_k}}, t_{q_{i_k}} >\}$$

the algorithm $Search$ can obtain several corresponding identification sets, which are $S_{<t_{p_{i_1}}, t_{q_{i_1}}>}$, $S_{<t_{p_{i_2}}, t_{q_{i_2}}>}$, …, $S_{<t_{p_{i_k}}, t_{q_{i_k}}>}$ respectively. Then, the algorithm $Search$ calculates the intersection of these sets, which is $\bigcap S_{<t_p, t_q>}$, where:

$$< t_p, t_q > \in token_Q = \{< t_{p_{i_1}}, t_{q_{i_1}} >, < t_{p_{i_2}}, t_{q_{i_2}} >, \ldots, < t_{p_{i_k}}, t_{q_{i_k}} >\}$$

4. The algorithm $Search$ extracts the ciphertexts in the table stored in the cloud according to the identifications in $\bigcap S_{<t_p, t_q>}$. These ciphertexts are as the search results I^* for the query range $Q = [p_{i_1}, q_{i_1}] \times [p_{i_2}, q_{i_2}] \times \ldots \times [p_{i_k}, q_{i_k}]$. Finally, the cloud returns I^* to the ENT.

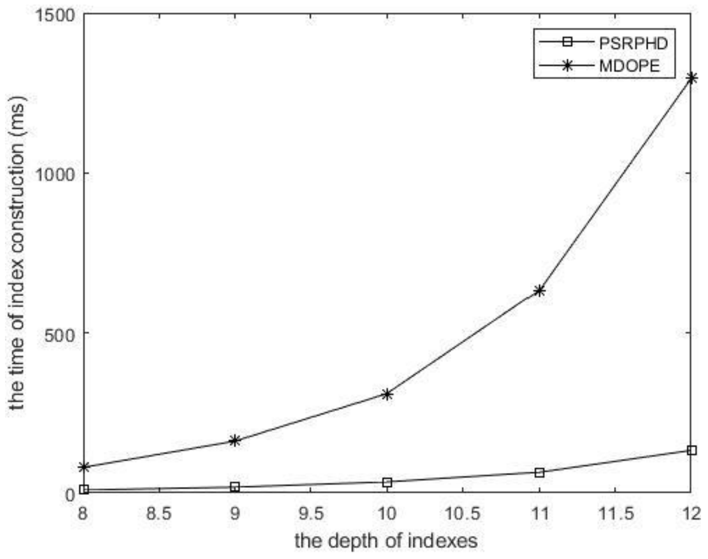
Decryption Algorithm - $Dec(SK, I^*) \to I$: The algorithm takes the ciphertexts I^* as inputs. It outputs the plaintext I through decrypting the ciphertexts with the first part of the secret key sk_1, i.e., $I = SE.Dec(I^*, sk_1)$. When the ENT receives the ciphertexts I^*, it requests the decryption key sk_1 from the SWDW, then runs the algorithm Dec to decrypt the ciphertexts in I^*, and finally obtain the plaintext search results I.

EXPERIMENTS

In our experiments, we compare the MDOPE scheme (Zhan et al., 2022) and the PPSRMPHD scheme. Usually, data collected by a SWD are transferred to the paired smartphone through a Bluetooth connection. Then, the smartphone encrypts the data before uploading to the cloud. Thus, in our experiments, we establish a smartphone emulator on a personal computer (AMD Ryzen 5 2500U CPU and 8G Random Access Memory) and implement these two schemes by using Java language. To achieve fairness in experimental comparisons, the experimental parameter settings for MDOPE and PPSRMPHD are provided below. In MDOPE and PPSRMPHD, each node of the tree index contains only one split value. We construct multiple datasets of 2-dimensional data for experimental comparisons. The ranges of these datasets are $[0,127] \times [0,127]$, $[0,255] \times [0,255]$, $[0,511] \times [0,511]$, $[0,1023] \times [0,1023]$ and $[0,2047] \times [0,2047]$ respectively. In PSRPHD, as the two-dimensional data should be transformed into two columns of table records, we build two secure binary tree indexes for these two columns, respectively.

Index Construction: As shown in Figure 3, when the depth of index increases from 8 to 12, the construction times of indexes in MDOPE and PPSRMPHD increase exponentially. This is because when the depth of the indexes increases, the total number of index nodes increases exponentially (the indexes in MDOPE and PPSRMPHD are tree structures). This results such that the construction time also increases exponentially. In our experimental settings, as the maximum range is $[0,2047]$, the encoding length is set to 12 bits, which is large enough for the range $[0,2047]$. In the index of MDOPE, the split value of each node is first transformed to a 12-bit binary value, and then padded with an additional 14-bit binary value (2 bits for data

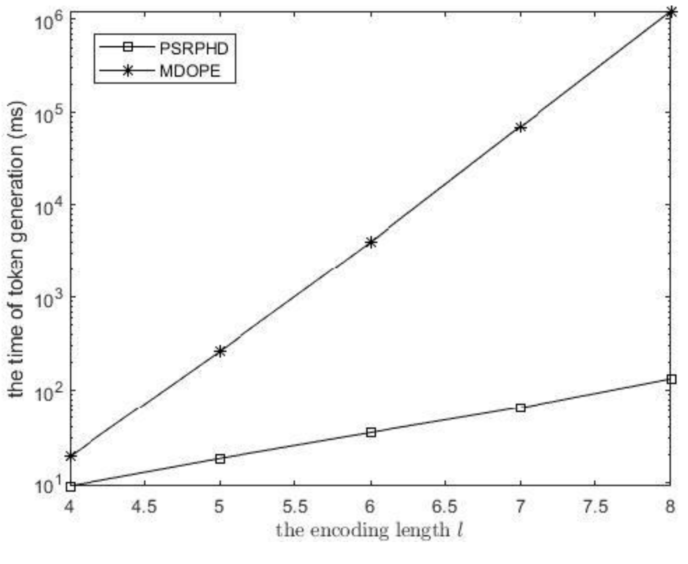
Figure 3. Index construction



comparison and 12 bits for randomization) after it. Next, the 26-bit (26=12+14) binary value is transformed to its prefix encoding. In PPSRMPHD, the index construction algorithm only generates a 24-bit binary value (12 bits for encoding the data in the range $[0, 2047]$ and 12 bits for randomization). As the encoding length of MDOPE is larger than that of PPSRMPHD and the prefix encoding used in MDOPE is more complex than the 0-1 encoding used in PPSRMPHD, the construction time of the index in MDOPE is more time consuming than the construction time of the index in PPSRMPHD.

Token Generation: As shown in Figure 4, in MDOPE and PPSRMPHD, when the encoding length increases one bit, the scale of the query range doubles, and the computation cost of the query

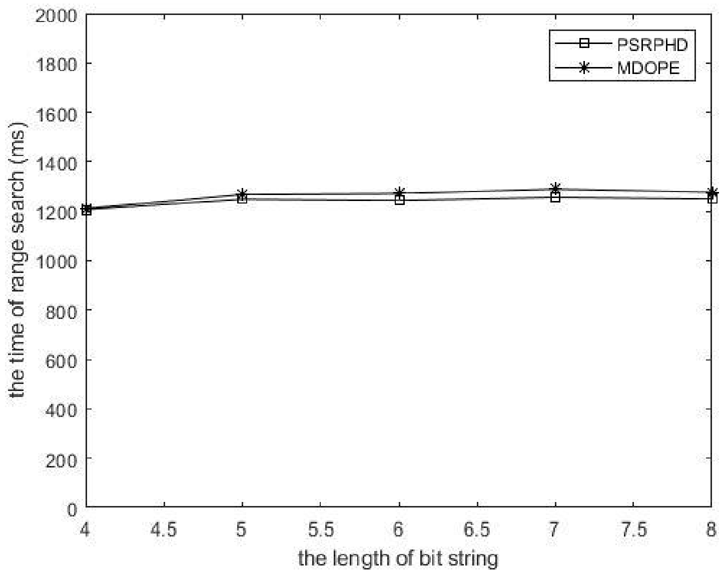
Figure 4. Query token generation



token generation also doubles. Therefore, the times of the token generation algorithms in MDOPE and PPSRMPHD increase exponentially. In MDOPE, a query range $[a,b]$ is first transformed into two binary values c_a and c_b whose lengths are l, where l increases from 4 to 8. As the length of c_a and c_b is l, the maximum range denoted by $[a,b]$ is $[0,2^l-1]$. Then, MDOPE pads c_a and c_b with two different binary values whose lengths are $l+2$ (2 bits for data comparison and l bits for randomization). Next, MDOPE obtains two binary values c_a' and c_b' whose lengths are $2l+2$ ($2l+2=l+(l+2)$). There are at most 2^{2l+2} different binary values between c_a' and c_b'. Finally, MDOPE merges all these different binary values into a few binary values as the prefix encoding of $[a,b]$. As the merging process for these different binary values is very time-consuming, the search token generation algorithm in MDOPE is relatively inefficient. In PSRPHD, a query range $[a,b]$ is first transformed into two binary values c_a and c_b whose lengths are l. Then, PPSRMPHD pads c_a and c_b with two different binary values whose lengths are l. Next, PPSRMPHD obtains two binary values c_a' and c_b' whose lengths are $2l$ ($2l=l+l$). Finally, PPSRMPHD only calculates the 1-encoding of c_a' and c_b' respectively to generate the search token of $[a,b]$. As the merging process in MDOPE is very time-consuming and the search token generation process in PPSRMPHD only requires to process two binary values, the token generation in PPSRMPHD is very efficient.

Search: As shown in Figure 5, when the number of data is fixed at 10,000 and the encoding length l increases from 4 to 8, the processes of range query in MDOPE and PPSRMPHD are very efficient. During the process of range query, each search token needs to be compared with the index nodes in a top-down manner, and the total number of index node comparisons is a crucial factor affecting the search efficiency. In our experiments, the indexes in MDOPE and PPSRMPHD are tree structures and set to the same depth. Thus, the total numbers of index node comparisons in MDOPE and PPSRMPHD are almost the same. Additionally, as the process of index node comparisons only involves comparing hash values with binary arrays, the search algorithm has

Figure 5. Search



a very high efficiency. Therefore, the search times of MDOPE and PPSRMPHD are almost the same and very efficient.

CORRECTNESS AND SECURITY ANALYSIS

Theorem 1: The scheme PPSRMPHD satisfies the correctness property defined in Definition 1.

Proof 1: We suppose that $Q = [p_{i_1}, q_{i_1}] \times [p_{i_2}, q_{i_2}] \times \ldots \times [p_{i_k}, q_{i_k}]$ is a query request and $[p_{i_j}, q_{i_j}]$ is the query range on the i_j-th column, where $i_j = i_1, i_2, \ldots, i_k$. And we suppose that the values of a table record d in the i_1-th, i_2-th, …, i_k-th columns are d_{i_1}, d_{i_2}, …, d_{i_k}, respectively. If $d \in Q$, there are $d_{i_1} \in [p_{i_1}, q_{i_1}]$, $d_{i_2} \in [p_{i_2}, q_{i_2}]$, …, $d_{i_k} \in [p_{i_k}, q_{i_k}]$. According to the 0-1 encoding, there are $S^0_{c_{d_{i_1}} | c_{r_{d_{i_1}}}} \cap S^1_{c_{p_{i_1}} | c_{r_{p_{i_1}}}} = \varnothing$ and $S^0_{c_{d_{i_1}} | c_{r_{d_{i_1}}}} \cap S^1_{c_{q_{i_1}} | c_{r_{q_{i_1}}}} \neq \varnothing$; $S^0_{c_{d_{i_2}} | c_{r_{d_{i_2}}}} \cap S^1_{c_{p_{i_2}} | c_{r_{p_{i_2}}}} = \varnothing$ and $S^0_{c_{d_{i_2}} | c_{r_{d_{i_2}}}} \cap S^1_{c_{q_{i_2}} | c_{r_{q_{i_2}}}} \neq \varnothing$; … ; $S^0_{c_{d_{i_k}} | c_{r_{d_{i_k}}}} \cap S^1_{c_{p_{i_k}} | c_{r_{p_{i_k}}}} = \varnothing$ and $S^0_{c_{d_{i_k}} | c_{r_{d_{i_k}}}} \cap S^1_{c_{q_{i_k}} | c_{r_{q_{i_k}}}} \neq \varnothing$. Thus, for the i_j-th column ($i_j = i_1, i_2, \ldots, i_k$), if the split value sv_{i_j} in a leaf node of the corresponding secure binary tree $T^*_{i_j}$ is covered by the queried range $[p_{i_j}, q_{i_j}]$ of Q, all the IDs of table records whose values in the i_j-th column equal to sv_{i_j} can be found. If ranges $[p_{i_1}, q_{i_1}]$, $[p_{i_2}, q_{i_2}]$, …, $[p_{i_k}, q_{i_k}]$ are used simultaneously for searching, all IDs of table records that satisfy the query request Q can be retrieved. Finally, all the ciphertexts in these table records are returned as the search results. In summary, the PPSRMPHD scheme is correct.

Theorem 2: The scheme PPSRMPHD satisfies the security property defined in Definition 2.

Proof 2: Due to the utilization of a secure encryption scheme SE, the security of personal health data collected by SWD can be ensured. The security of split values in the secure binary tree index can be analyzed in the following ways. The split values are processed by using the padding, the 0-1 encoding and the Bloom filter techniques. In the scheme PPSRMPHD, $u = u_1 u_2 \ldots u_n$ represents the binary value obtained by padding random number r after the split value sv (see encoding algorithm); $v = v_1 v_2 \ldots v_n$ represents the binary value obtained by padding r_p (or r_q) after the lower limit p (or the upper limit q) (see token generation algorithm). If the member in the intersection of $S^0_{c_u | r}$ and $S^1_{c_p || r_p}$ (or $S^1_{c_q || r_q}$) is z, where the length of z is m, it can deduced that $v_1 = u_1$, $v_2 = u_2$, …, $v_{m-1} = u_{m-1}$, $x_m \neq y_m$. As a result, the cloud is only aware of the leakage function $F(u, v) = position_{diff}(u, v)$. Thus, the PPSRMPHD scheme satisfies the security property defined in Definition 2.

CONCLUSION

In this paper, we propose a Privacy-Preserving Storage and Retrieval Method for Personal Health Data (PPSRMPHD) scheme, which is designed to safeguard motion data (such as motion trajectory and status) and physiological data (such as heart rate and blood pressure) collected by SWDs. In the PPSRMPHD, the SWD transmits the collected data to the paired smartphone through a Bluetooth connection. The paired smartphone, which encrypts the collected data, builds secure binary tree indexes and, finally, outsources the ciphertexts and secure binary tree indexes to the cloud. The secure binary tree indexes are built by using Bloom filters and 0-1 encoding techniques. After obtaining approval from the SWDW, the SWD generates query tokens for the ENT. The ENT can use these query tokens to perform efficient queries in the cloud. By utilizing the 0-1 encoding techniques and

Bloom filters, the PPSRMPHD can transform the comparisons of ciphertexts into verifications of the presence of 1s at specific positions in a bit array. Therefore, this approach significantly improves the efficiency of the query process. Moreover, due to the low computational complexity of 0-1 encoding and Bloom filters, the PPSRMPHD scheme exhibits high efficiency in index construction and token generation. In summary, the PPSRMPHD is capable of achieving secure cloud storage and retrieval of personal health data collected by SWDs.

ACKNOWLEDGMENT

This research was supported by the National Natural Science Foundation of China (No. 61962029, 62262033, 62062045, 62362042); the Jiangxi Provincial Natural Science Foundation of China (No. 20202BAB212006, 20224BAB202012, 20202ACBL202005); the Science and Technology Research Project of Jiangxi Education Department (No. GJJ201832, GJJ211815); the Hangzhou Philosophy and Social Sciences Planning Project (No. M22JC067); the Zhejiang Provincial Natural Science Foundation of China (No.LR23F020001); the Hubei Natural Science Foundation Innovation and Development Joint Fund Project under Grant (No.2022CFD101); the Xiangyang High-tech Key Science and Technology Plan Project under Grant (No.2022ABH006848); the Hubei Superior and Distinctive Discipline Group of "New Energy Vehicle and Smart Transportation."

REFERENCES

Agrawal, R., Kiernan, J., Srikant, R., & Xu, Y. (2004, June). Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on management of data* (pp. 563-574). doi:10.1145/1007568.1007632

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, *13*(7), 422–426. doi:10.1145/362686.362692

Boldyreva, A., Chenette, N., Lee, Y., & O'Neill, A. (2009). Order-preserving symmetric encryption. In *Advances in cryptology-EUROCRYPT 2009: 28th annual international conference on the theory and applications of cryptographic techniques* (pp. 224-241). doi:10.1007/978-3-642-01001-9_13

Boldyreva, A., Chenette, N., & O'Neill, A. (2011). Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Advances in cryptology–CRYPTO 2011: 31st annual cryptology conference* (pp. 578-595). Academic Press.

Cui, Z., Lu, Z., Yang, L. T., Yu, J., Chi, L., Xiao, Y., & Zhang, S. (2023). Privacy and Accuracy for Cloud-Fog-Edge Collaborative Driver-Vehicle-Road Relation Graphs. *IEEE Transactions on Intelligent Transportation Systems*, *24*(8), 8749–8761. doi:10.1109/TITS.2023.3254370

David, H. A., & Nagaraja, H. N. (2004). *Order statistics*. John Wiley & Sons.

Dyer, J., Dyer, M., & Xu, J. (2017). Order-preserving encryption using approximate integer common divisors. In Data privacy management, cryptocurrencies and blockchain technology: ESORICS 2017 international workshops, (pp. 257-274). doi:10.1007/978-3-319-67816-0_15

Guo, J., Wang, J., Zhang, Z., & Chen, X. (2018, September). An almost non-interactive order preserving encryption scheme. In *International conference on information security practice and experience* (pp. 87-100). doi:10.1007/978-3-319-99807-7_6

Gupta, P., & McKeown, N. (2001). Algorithms for packet classification. *IEEE Network*, *15*(2), 24–32. doi:10.1109/65.912717

Hacigümüş, H., Iyer, B., Li, C., & Mehrotra, S. (2002, June). Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD international conference on management of data* (pp. 216-227). doi:10.1145/564691.564717

Hore, B., Mehrotra, S., Canim, M., & Kantarcioglu, M. (2012). Secure multidimensional range queries over outsourced data. *The VLDB Journal*, *21*(3), 333–358. doi:10.1007/s00778-011-0245-7

Hore, B., Mehrotra, S., & Tsudik, G. (2004, August). A privacy-preserving index for range queries. In *Proceedings of the thirtieth international conference on very large data bases* (pp. 720-731). Academic Press.

Krendelev, S. F., Yakovlev, M., & Usoltseva, M. (2014, September). *Order-preserving encryption schemes based on arithmetic coding and matrices. 2014 federated conference on computer science and information systems.*

Lee, Y. (2014). Secure ordered bucketization. *IEEE Transactions on Dependable and Secure Computing*, *11*(3), 292–303. doi:10.1109/TDSC.2014.2313863

Lin, H. Y., & Tzeng, W. G. (2005). An efficient solution to the millionaires' problem based on homomorphic encryption. In *Applied cryptography and network security: Third international conference* (pp. 456-466). Academic Press.

Mei, Z., Zhu, H., Cui, Z., Wu, Z., Peng, G., Wu, B., & Zhang, C. (2018). Executing multi-dimensional range query efficiently and flexibly over outsourced ciphertexts in the cloud. *Information Sciences*, *432*, 79–96. doi:10.1016/j.ins.2017.11.065

Peng, Y., Li, H., Cui, J., Zhang, J., Ma, J., & Peng, C. (2017). hOPE: Improved order preserving encryption with the power to homomorphic operations of ciphertexts. *Science China. Information Sciences*, *60*(6), 1–17. doi:10.1007/s11432-016-0242-7

Popa, R. A., Redfield, C. M., Zeldovich, N., & Balakrishnan, H. (2011, October). CryptDB: Protecting confidentiality with encrypted query processing. In *Proceedings of the twenty-third ACM symposium on operating systems principles* (pp. 85-100). doi:10.1145/2043556.2043566

Quan, H., Wang, B., Zhang, Y., & Wu, G. (2018). Efficient and secure top-k queries with top order-preserving encryption. *IEEE Access : Practical Innovations, Open Solutions*, *6*, 31525–31540. doi:10.1109/ACCESS.2018.2847307

Teranishi, I., Yung, M., & Malkin, T. (2014). Order-preserving encryption secure beyond one-wayness. In *Advances in cryptology–ASIACRYPT 2014: 20th international conference on the theory and application of cryptology and information security* (pp. 42-61). doi:10.1007/978-3-662-45608-8_3

Wang, P., & Ravishankar, C. V. (2013, April). Secure and efficient range queries on outsourced databases using Rp-trees. In *2013 IEEE 29th international conference on data engineering (ICDE)* (pp. 314-325). doi:10.1109/ICDE.2013.6544835

Wong, W. K., Cheung, D. W. L., Kao, B., & Mamoulis, N. (2009, June). Secure kNN computation on encrypted databases. In *Proceedings of the 2009 ACM SIGMOD international conference on management of data* (pp. 139-152). doi:10.1145/1559845.1559862

Xiao, L., & Yen, I. L. (2012, March). Security analysis for order preserving encryption schemes. In *2012 46th annual conference on information sciences and systems (CISS)* (pp. 1-6). Academic Press.

Zeng, J., Che, J., Xing, C., & Zhang, L. J. (2018). Leadsrobot: A sales leads generation robot based on big data analytics. In *Big Data–BigData 2018: 7th International Congress, held as part of the Services Conference Federation, SCF 2018, Seattle, WA, USA, June 25–30, 2018, Proceedings* (pp. 277-290). Springer International Publishing. doi:10.1007/978-3-319-94301-5_21

Zeng, J., Yang, L. T., Ma, J., & Guo, M. (2015). HyperspaceFlow: A system-level design methodology for smart space. *IEEE Transactions on Emerging Topics in Computing*, *4*(4), 568–583. doi:10.1109/TETC.2015.2501846

Zhan, Y., Shen, D., Duan, P., Zhang, B., Hong, Z., & Wang, B. (2022). MDOPE: Efficient multi-dimensional data order preserving encryption scheme. *Information Sciences*, *595*, 334–343. doi:10.1016/j.ins.2022.03.001

Zhuolin Mei received the B.E. degree from Wuhan University of Technology in 2009, and the Ph.D. degree from Huazhong University of Science and Technology in 2017. His research interests include access control, data encryption, ciphertext search, and cloud computing. He is currently a Lecturer with the School of Computer and Big Data Science, Jiujiang University, China.

Jing Yu received the M.Sc. degree from Huazhong University of Science and Technology in 2012 and the Ph.D. degree from Wonkwang University in 2021. She is currently a lecturer at Jiujiang University. Her research interests include Social Responsibility, Privacy-Preserving and Data Analysis.

Jinzhou Huang received the PhD degree in computer system architecture from Huazhong University of Science and Technology (HUST), in 2015. He is currently an associate professor in the School of Computer Engineering, Hubei University of Arts and Science, China. His research interests include online social networking, intelligent transportation, internet of things, computer networking and information security.

Bin Wu received the Ph.D. degree from the Huazhong University of Science and Technology in 2017. His research interests include privacy preserving, big data security, cloud security, information security, and information query.

Zhiqiang Zhao received the B.Sc. degree in computer science and technology in 2002 from Hunan University, Changsha, China. He received the Ph.D. degree in the School of Computer Science and Technology, Huazhong University of Science and Technology. He is currently an associate professor with the School of Mathematics and Computer Science, Ningxia Normal University. His research interests include visual tracking, computer vision, and image processing.

Caicai Zhang received the B.E. degree from Changchun University in 2009, and the Ph.D. degree from Huazhong University of Science and Technology in 2016. Her research interests include uncertain data management, machine learning and cloud computing. She is currently a Lecturer with the School of Modern Information Technology, Zhejiang Institute of Mechanical & Electrical Engineering, China.

Jiaoli Shi received the Ph.D. degree from the School of Computing, Wuhan university, in 2017. Since 2012, she has been an Assistant Professor at the school of computer and big data science, Jiujiang University. Her research interests include Cloud security, ICN security, SDN and CDN. Dr. Jiaoli twice won the third prize of Doctoral Forum of School of Computer Science, Wuhan University for Excellence in 2014 and 2015.

Xiancheng Wang received a B.S. degree from Shandong University of Science and Technology, China, in 2005, and received an M.S. degree from Huazhong University of Science and Technology, China, in 2009, and recieved an Ph.D degree from Kunsan National University, South Korea, in 2021. He is currently a lecturer with the School of Computer and Big Data Science, Jiujiang University, and his research interests include Microgrid, Optimization method, and Blockchain.

Zongda Wu is a full professor at Shaoxing University. He received his Ph.D. degree in Computer Science from Huazhong University of Science and Technology (HUST) in 2009. From 2019, he worked as a postdoctoral research fellow at Nanjing University. As the first author, he has published more than 70 papers on many journals (such as TSC, TVT, TETC, JASIST, KIS, INS and WWW) and conferences (such as CIKM, ICDM and IJCNN). His research interests are primarily in the area of information retrieval and user privacy.