ISSN: 2229-6948(ONLINE)
DOI: 10.21917/ijct.2016.0184

STRENGTHENING ENCRYPTION SECRECY FOR PRIVATE SEARCH USING FULLY HOMOMORPHIC ENCRYPTION

J. Ramya[1] and M. Saravanan[2]
Department of Computer Science and Engineering, K. Ramakrishnan College of Engineering, India
E-mail: jramyakrce@gmail.com
Department of Computer Science and Engineering, Sri Ramakrishna College of Engineering, India
E-mail: srisaravana.c@gmail.com

Abstract

A system for private searching on streaming data, allows client to send the encrypted search query to the remote server. The server uses the encrypted query on a stream of documents and returns the matching documents to the client without revealing the features of the query. A novel technique for private searching on streaming data is proposed which is based on keyword frequency that is the number of times that a keyword appears in the document is required to be higher or lower than a given threshold. This form of query searching can help the client in locating more related documents. Using fully homomorphic encryption techniques, the server can perform search for retrieving the related documents even though the search query is in encrypted form. Our scheme provides number of vital benefits for the client. They provide provable secrecy for encryption, in the view that the untrusted server cannot gather any information about the plaintext. They also reinforce hidden queries, so that the client may ask the remote server to search for a secret word without revealing the word to the server.

Keywords:

Homomorphic Encryption, Private Searching on Streaming Data, Trusted Security, Query Search, Efficient Retrieval

1. INTRODUCTION

The objective of encryption is to ensure confidentiality of data in communication and storage processes. The crucial problem crop ups when there is a constraint for computing publicly over untrusted server with private data and while ensuring the privacy.

To evade this situation, the client transmits only an encrypted version of the data to the untrusted server to process. The server will perform the computation on this encrypted data without recognizing anything about its real value. Finally, it will dispatch the result back to the client, and the client will decrypt it. The decrypted result will be equal to the expected computed value if acted upon the original data. This is where Homomorphic cryptosystems can be used, since this system facilitates computations on encrypted data.

Ostrovsky and Skeith [3] gave a basic solution for private searching on streaming data using the concept of public/private key obfuscation. The basic idea can be briefly explained as follows,

Consider the public dictionary of possible keywords is $D = \{w_1, w_2 \ldots, w_{|D|}\}$. To search for documents containing one or more of keywords $Kw = \{k_1, k_2 \ldots, k_{|Kw|}\} \subset D$, the client generates a public/private key pair and constructs a program P, composed of an encrypted dictionary (D) from Kw and a buffer B which will store matching documents. Then the client dispatches the program P to a public server, where P filters a streaming document and stores the encryptions of matching documents in the buffer B. After the buffer B returns, the client decrypts the buffer and retrieves the matching documents. The searching criterion is kept classified, because both the keywords and the buffer are in encrypted form.

Consider a scenario for a cloud service governing electronic medical records (EMR), where devices continuously collect vital health information, and stream them to a server who then computes some statistics and apparently decide on the course of treatment.

The volume of the data comprised is large and thus the patient presumably does not want to store and manage all this data in the neighborhood, she may prefer the assistance of cloud storage and computation. To protect patient privacy, all the data is uploaded in encrypted form, and thus the cloud must perform operations on the encrypted data in order to return encrypted alerts, predictions, or summaries of the results to the patient.

The existing solutions for private searching on streaming data have not considered keyword frequency, the number of times that keyword is used in a document. That is, a novel private query, which explores for documents based on keyword frequency, such that a number of times that a keyword appears in a matching document is required to be higher or lower than a given threshold.

For example, find documents containing keywords $\{k_1, k_2, \ldots, k_n\}$ such that the frequency of the keyword k_i in the document is higher than ft_i.

2. PRELIMINARIES

2.1 FULLY HOMOMORPHIC ENCRYPTION ALGORITHM

Dijk et al. [2] proposed the fully Homomorphic encryption algorithm and explained as follows:

To protect privacy, all the data is uploaded in encrypted form, and thus the cloud must perform operations on the encrypted data in order to return encrypted alerts, predictions.

2.2 KeyGen(k)

Takes the security parameter k and outputs a pair of secret key and public key.

1. Takes the security parameter k.
2. Determines the parameters η, ρ, γ, τ satisfying certain conditions.
3. Choose a random odd η bit integer p from
4. $(2\mathbb{Z}+1) \cap (2^{\eta-1}, 2^{\eta})$ as the secret key s_k.

5. Arbitarily choose $q_0,q_1,\ldots q_\tau$ from (1,2^γ/p), to the condition that q_i is odd and re-label $q_0,q_1,\ldots q_\tau$ so, that q_0 is largest.
6. Randomly choose $r_0,r_1,\ldots r_\tau$ from $\mathbb{Z} \cap (2^{-\rho},2^\rho)$.
7. Calculate $x_0 = q_0p+2r_0$ and $x_i = (q_ip+2r_i)\%\ x_0$
8. The Public Key, $p_k = <x_0,x_1,\ldots,x_\tau>$.

Encrypt(p_k,m):

Takes input as public key, p_k and message to be encoded, m and returns the cipher texts, c.

1. Choose m as $m \in \{0,1\}$
2. Choose a random subset $S \subset \{1,2,..,\tau\}$ and a random integer r from $(2^{-\rho},2^\rho)$.
3. Generate the cipher texts using Eq.(1),

$$c = (m) = m + 2r + \sum_{i \in S} x_i \bmod x_0 \quad (1)$$

Decrypt(s_k,c):

Takes input as secret key and cipher text and returns the original message m.

$$m' = (c \bmod s_k) \bmod 2 \quad (2)$$

2.3 FULLY HOMOMORPHIC ENCRYPTION PROPERTIES

Some of the fully Homomorphic encryption properties are listed below,

In the Goldwasser–Micali cryptosystem [11], add each component of cipher texts, and the decrypted result is equivalent to the XORed values of the plaintext.

$$(x_1) + (x_2) = (x_1 \oplus x_2).$$

In the ElGamal cryptosystem, multiply each component of cipher texts, and the decrypted result is equivalent to the multiplication of the plaintext values.

$$(x_1) \,.\, (x_2) = (x_1x_2).$$

In Paillier cryptosystem [12], multiply each component of cipher texts, and the decrypted result is equivalent to the addition of the plaintext values.

$$(x_1) \,.\, (x_2) = (x_1) + (x_2), \text{ where } x_1, x_2 \in \{0, 1\}.$$

2.4 BINARY ADDITION

For a positive integer S expressed in binary form where $S = (s_1, s_2,\ldots, s_l)$. Assume $(S_1) = ((a_1), (a_2),\ldots, (a_l))$ and $(S_2) = ((b_1), (b_2),\ldots. (b_l))$, we can construct (S_1+S_2) as follows.

Consider $(a_1, a_2, \ldots a_l) + (b_1, b_2,\ldots b_l) = (d_0, d_1, d_2,\ldots d_l)$ where d_0 is the carry bit. On the basis of binary integer addition [8],

$$c_{i-1} = a_ib_i \forall ((a_i \oplus b_i)c_i)$$

$$d_i = a_i \oplus b_i \oplus c_i, \quad (3)$$

for $i = 1,2,\ldots l$, then $(d_0) = (c_0)$ and $S = (S_1 \boxplus S_2)$

2.5 INTEGER COMPARISON

Consider two positive integers S_1 and S_2. It is possible to compare by S_1 and S_2, by calculating the 2's complement of -S_2 as $\overline{-S_2}$.

If $S_1 \geq S_2$, then MSB of $\overline{S_1} + \overline{-S_2}$ is 0 and 1 otherwise.

We can compare S_1 and S_2 by computing,

$$\left(\overline{S_1} + \overline{-S_2}\right) = (s_{12}) + (s_{22}) + 1,\ldots, (s_{11}) + (s_{21}) + 1 + 1,$$

$$= \left(\overline{S_1}\right) \boxplus \left(\overline{-S_2}\right) \quad (4)$$

3. PROPOSED METHOD

In Fig.1, the client will take a security parameter k and generates a pair of public and secret keys (p_k, s_k) by executing KeyGen(k) function and constructs a program P which holds an encrypted dictionary (D), an encrypted search query (s_q) and the public key p_k. The program P is sent to the server.

The public server takes the stream of documents S, and searches for the document containing the search query s_q, $s_q \subset S_i$. The server stores up to m encrypted matching document in a buffer B, and finally outputs an encrypted buffer B to the client.

The client will decrypt the buffer B using secret key s_k.

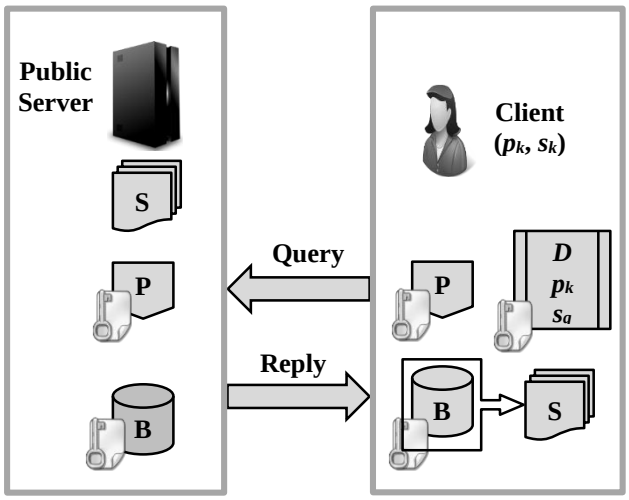

Fig.1. Architecture of private stream searching system

Table.1. Notations

Symbols	Explanation
D	Dictionary of potential keywords
|D|	Number of potential keywords in dictionary
K_w	Set of keywords
w_i	A word present either in dictionary or document
k_i	A keyword
S	Set of documents in streaming data
B	Buffer to store matching documents
C	Set of cipher texts
$<p_k, s_k>$	Public and Secret key pair
$\boxplus$	Homomorphic addition of integers
ft_i	Frequency threshold of keyword k_i
$f(k_i)$	Frequency of the keyword k_i in document S
$\hat{C}$	Set of common words in the document S and the dictionary D and their frequencies in S.

For set of keywords $K_w = \{k_1, k_2,\ldots, k_3\}$, disjunctive threshold query can be stated as,

$$(f(k_1) \geq ft_1) \vee (f(k_2) \geq ft_2) \vee \ldots.. \vee (f(k_n) \geq ft_n),$$

where, $f(k_i)$ is the frequency of the keyword k_i and ft_i is the frequency threshold.

Our scheme for disjunctive threshold query consists of 5 algorithms like Key generation, Keyword frequency construction, Word collection, Keyword frequency comparison and Buffer decryption. Our scheme is formally explained as follows,

Key Generation: Client executes this function in fully Homomorphic encryption algorithm and returns a pair of public key p_k and secret key s_k.

Frequency Threshold Construction: This algorithm takes an encrypted dictionary D, set of keywords K_w and document to be searched S and outputs ft, which holds frequency threshold for each word in dictionary.

Assume that the public dictionary $D = \{w_1, w_2\ldots w_{|D|}\}$, Keywords $K_w = \{k_1, k_2,\ldots, k_{|Kw|}\} \subset D$, $p = [\log_2|S|]$ where $|S|$ denotes the maximal number of words in the document S. Calculate the frequency of each and every word in the dictionary using Eq.(5),

$$(D) = \{(w_1), (w_2)\ldots (w_{|D|})\}$$

where, $(w_i) = (ft_i)$

$$ft_i = \begin{cases} \text{frequency threshold for } k_j & \text{if } w_i = k_j \in K \\ 2^p - 1 & \text{if } w_i \notin K \end{cases} \quad (5)$$

Because the document S contains at most $2^p - 1$ words, the frequency of any word in S is less than $2^p - 1$. The algorithm for frequency threshold comparison is explained in Algorithm 1.

Algorithm 1: ***Frequency Threshold Construction***

Name	:	FrequencyThreshold(D,K,S)
Input	:	D – set of words in dictionary, K_w – set of keywords, S – document to be searched
Output	:	et – encrypted value of frequency threshold for each word in dictionary

Begin
P = (log(S.length)/log2);
for each word in D as i
for each word in K_w as j
if(D_i == k_j)
ft_i = Compute the frequency of k_j;
else
$ft_i = 2^p$-1;
for each word in D as i
ft_i = 2'sComp(ft_i);
et_i = Encrypt(p_k,ft_i);
return et;
End

Word Collection: This function inputs a dictionary D and a document S and outputs a set of common words in the document S and the dictionary D and their frequencies in encrypted form,

$$\hat{C} = \{w_i, f(w_i) \mid w_i \in S \cap D\} \quad (6)$$

where, $f(w_i)$ is the frequency of w_i in the document S.

Then the frequencies $f(w_i)$ of w_i are encrypted. That is, the encryption of $f(w_i) = \{a_1, a_2,\ldots a_l\}$ denoted as,

$$(f(w_i)) = \{d(0), (a_1), (a_2),\ldots (a_l)\}$$

The algorithm for word collection is explained in Algorithm 2.

Algorithm 2: ***Word Collection***

Name	:	WordCollection(D,S)
Input	:	D – set of words in dictionary, S – document to be searched
Output	:	$\hat{C}$ – Encrypted set of common words in the document S and the dictionary D and their frequencies in S.

Begin
for each word in D as i
for each word in S as j
if(D_i == S_j)
count+=1; /* *count calculates the frequency* */
if(count!=0)
$\hat{C}_{i0} = D[i]$;
$\hat{C}_{i1}$ = count;
end for
end for
for each word in $\hat{C}$ as i
$\hat{C}_{i1}$= Encrypt(H_{i1})
return $\hat{C}$;
End

Keyword Frequency Comparison: For each word w_i in $\hat{C}$, the system homomorphically compares the frequency ($f(w_i)$)and the frequency threshold $\left(\overline{-ft_i}\right)$.

$$(f(w_i) + \overline{-ft_i} = (f(w_i)) \boxplus \overline{-ft_i}$$
$$= ((c_{i0}), (c_{i1}), (c_{i2}),\ldots., (c_{id})).$$

From this (c_{i0}) is extracted.

In 2's Complement system, if $c_{i0} = 0$, then $f(w_i) \geq ft_i$ and otherwise $f(w_i) < ft_i$. The encrypted values of c_{i0} are stored in the buffer B and is sent to the client.

The algorithm for keyword frequency comparison is explained in Algorithm 3.

Algorithm 3: ***Keyword Frequency Comparison***

Name	:	KeywordFrequencyComparison(D,Kw,S)
Input	:	D – set of words in dictionary, k_w – set of keywords, S – document to be searched
Output	:	Returns Buffer B

Begin
et = ThresholdAssignment(D,Kw,S);
$\hat{C}$ = WordCollection(D,S);
for each word in $\hat{C}$ as i
for each word in et as j
if($\hat{C}_i$ == et_i)
z = BinaryAddition($\hat{C}_{i1}$,ft_{j1})
B[flag] = z[1];
flag++;
end if
end for

```
  end for
return B;
End
```

Buffer Decryption: The client receives the buffer B from the server. The encrypted values in B are decrypted using the secret key s_k. It computes the values c, using Eq.(7),

$$c \vee = B_i \oplus 1. \quad (7)$$

If c value is 0, then the document is the matching document. Else if nonzero, then the document is not a matching document. The algorithm for buffer decryption is explained in Algorithm 4.

Algorithm 4: ***Buffer Decryption***

Name	:	BufferDecryption(B,*sk*)
Input	:	*B* – buffer, *sk* – secret key
Output	:	Returns whether *S* is matching document or not

```
Begin
  for each value in B as i
Bi = decrypt(Bi, sk);
cV = Bi ⊕ 1
end for
if (c == 0)
return S is matching document
else
return S is not a matching document
End
```

4. PERFORMANCE EVALUATION

The client performs only two operations. It can perform encryption and decryption step. In other words the client needs to encrypt the frequency of each keyword in the phase of the filter program generation and to decrypt the buffer B to retrieve the matching documents after the buffer returns.

Complexity

The computation complexity of the client is $O(|K_w|)$ encryptions to generate the program P where $|K_w|$ is the total number of the keywords. To decrypt the buffer B, it holds $O(|m|)$ decryptions where m is the total number of matching documents.

The server executes two operations. It performs word collection and frequency comparison steps. Specifically, after receiving the filter program P, the server processes each document S_i. At Word collection step, consider $\mu = |S_i \cap D|$ = number of common words in both document S and dictionary D. It takes $O(\mu)$ complexity. At Frequency comparison step, it takes $O(\mu)$ complexity. So, The total complexity at server side = $O(\mu)$.

4.1 EXPERIMENTAL RESULTS

4.1.1 Encryption and Decryption Algorithm Results:

Key Generation:

Consider the chosen values for the parameters ρ, γ, $\rho = 1$, $\gamma = 9$, $\tau = 5$. The system arbitrarily picks a 4-bit odd integer 13 as a secret key s_k. It randomly choses an odd integer in the range (1, 39) for the parameters q_0, q_1, q_2, q_3, q_4, q_5 and re-label q_0, q_1, q_2, q_3, q_4, q_5 so, that q_0 is largest. Randomly chooses r_0, r_1, r_2, r_3, r_4, r_5 from the range (-1, 2). Calculates $x_0 = q_0p + 2r_0$ = ((37*13) + (2*2)) = 485 and $x_i = (q_ip + 2r_i)$ % x_0.x_1 = ((37*13) + (2*2)) % 485 = 0 and so on. The key generation results are listed in Table.2.

Encryption:

The system has chosen a message to be encoded as 0. It picks a random subset S as 5 and a random integer r from (-1, 2). It generates cipher texts using Eq.(1),

$$c = 0 + 2 * 0 + (0+403+277+247+121) = 1048.$$

Checks whether the generated cipher text satisfies the condition, (*c % sk*) *% 2 == m*. (1048 % 13) % 2 = 0. The encryption results are listed in Table.2.

Decryption:

The function decrypts the cipher text using the secret key 13 and returns the decrypted result. The value of the plain text will be equal to the decrypted result.

4.1.2 Frequency Threshold Computation:

In this algorithm, the frequency of each and every word in the dictionary is computed. The function takes dictionary D, keywords K_w which consists of set of keywords that are present in the dictionary and the document to be searched S holds more number of words. Consider the partial dataset values for dictionary D, Keyword K_w and Document to be searched S as,

D = {1100, 0111, 1110, 0011, 1000, 1001, 0001, 0010, 0100, 1111}

K = {1100, 0000, 1110, 1111};

M = {0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 0000, 0001, 0010, 1000, 1000, 1111, 1111, 0001, 1110, 0000, 1100, 0011, 0011, 1111, 1100, 1110, 0000, 1100}

It computes the frequency threshold, if the word in a dictionary D contains in keyword K_w (i.e) w_i in $D \in K_w$, then the frequency of the word w_i is counted and stores in ft_i. If not, then 2^p-1 value is assigned to the frequency of the word w_i.

The word in dictionary w_1 = 1101 contains in the keyword. So, the value of ft_i = 3 and the word in dictionary w_2 = 111 does not belongs to the word in keyword. So the value of $ft_i = 2^p$-1 = 5. It converts the ft_i values to binary form and calculates the two's complement for the number. The resulted two's complement values are encrypted in bit by bit fashion. The sample results to calculate the frequency threshold is listed in Table.3.

Table.2. Encryption and Decryption Results

Key generation Chosen values are $\eta = 4$, $\rho = 1$, $\gamma = 9$, $\tau = 5$ Secret Key: 13 Before Relabeling… q[0]:9, q[1]:21, q[2]:37, q[3]:19, q[4]:31, q[5]:37 in the range (1, 39) After Relabeling the parameters, so that q[0] is largest… q[0]:37, q[1]:37, q[2]:31, q[3]:21, q[4]:19, q[5]:9 r[0]:2, r[1]:2, r[2]:0, r[3]:2, r[4]:0, r[5]:2 in the range (-1, 2) The Public keys are < 485, 0, 403, 277, 247, 121>
Encryption: Assume m = 0 The Cipher Text is 1048
Decryption: m = 0

4.1.3 Word Collection:

In word collection step, the function collects the set of common words in both dictionary and document S with their frequencies. It computes the frequencies ($f(w_i)$) of those common words in *S*. It converts the $f(w_i)$ values to binary form. The binary values are encrypted in bit by bit fashion. The function outputs the words in $D \cap S$ and their frequencies in encrypted form. The sample result to collect the common word in *S* and *D* and the calculation of their frequencies as cipher texts are listed in Table.4.

Table.3. Frequency threshold Computation Results

Word	*Frequency* (*ft*)	*-ft*	(*-ft*)
1100	11	0101	98,31,237,47
111	11	0101	4,15,98,101
1110	101	1011	234,98,15,47
11	10	0110	15,101,237
1000	101	1011	47,30,234,101
1001	101	1011	15,98,234,47
1	101	1011	31,4,47,237
10	101	1011	234,98,15,47
100	101	1011	101,237,15,234
1111	11	0101	30,101,78,63

Table.4. Word Collection Step Results

Word in $D \cap S$	*Frequency*	(*f*)
1100	11	237,47
111	1	15
1110	10	101,30
11	11	47,679
1000	10	234,98
1	11	234,101
10	10	659,98
100	01	15
1111	11	101,47
1100	11	237,47

4.1.4 Frequency Threshold Comparison:

In this step, the function performs frequency comparison and returns a buffer *B* which holds the MSB bits of the compared results. For each word w_i in $\hat{C}$, the function homomorphically compares the frequency $\hat{C} = (f(w_i))$ and the frequency threshold $et = \left(\overline{-ft_i}\right)$. The system compares for the word 1100, then *f*(1100) is 0011 and *ft*(1100) is 0011. Two's complement of *ft* is -*ft*(1100) = 0101. The cipher texts generated for these values are (*f*(1100)) = <4, 30, 234, 101>, (-*ft*(1100)) = <98, 31, 237, 47> . The function converts the cipher text values to binary and performs binary summation. The MSB of the result are stored in the buffer *B*. Finally, it returns the buffer *B* to the client. The result for keyword frequency comparison is listed in Table.5.

Table.5. Keyword Frequency Comparison Results

***f*(1100)=0011, *t*(1100)=0011**
(*f*(1100)) = (0011) = < 4, 30, 234, 101> ,
(-*ft*(1101)) = (0101) = < 98, 31, 237, 47>
(*f*(1100))) = < 100, 11110, 11101010, 1100101>
(-*ft*(1101)) = < 1100010, 11111, 11101101, 101111>
Sum[ ] = 1100110, 111101, 111010111, 10010100
Add Sum[0] value to the Buffer *B*
***f*(111)=0001, *t*(111)=0011**
(*f*(111)) = (0001) = < 4, 237, 30, 15>,
(-*ft*(111)) = (0101) = < 4, 15, 98, 101>
(*f*(111)) = <100, 11101101, 11110, 1111>
(-*ft*(111)) = <100, 1111, 1100010, 1100101 >
Sum[ ] = 1000, 11111100, 10000000, 1110100
Add Sum[0] value to the Buffer *B*

4.1.5 Buffer Decryption:

The client receives the buffer *B* from the server. The encrypted values in *B* are decrypted using the secret key. It computes the values c_0 using Eq.(6).

If c_0 value is 0, then the document is the matching document, otherwise the document is not a matching document. The result for buffer decryption is listed in Table.6.

Table.6. Buffer Decryption Results

B[0] = 102 Decrypt the value (102) = (102%7)%2 B_1=0 *B*[1]=8 Decrypt the value(8) = (8%7)%2 B_2=1	*B*[2] = 501 Decrypt the value(501) = (501%7)%2 B_3=0 *B*[3]=96 Decrypt the value(96) = (96%7)%2 B_4=1
$c_0 \vee = B_i \oplus 1$ c_0=(0 ⊕ 1) ∨ (1 ⊕ 1) ∨ (0 ⊕ 1) ∨ (1 ⊕ 1) c_0=1	
As c_0 = 1, so the given document *m* is the matching document	

4.1.6 Matching Document Computation Without Encryption:

Consider the function, which performs matching document computation without encrypting the frequencies and returns a buffer *B* which holds the MSB bits of the compared results. For each word w_i in $\hat{C}$, the function homomorphically compares the frequency $\hat{C} = f(w_i)$ and the frequency threshold $et = \overline{-ft_i}$ (w_i). The system compares for the word 1100, then *f*(1100) is 0011 and *ft*(1100) is 0011. Two's complement of *ft* is -*ft*(1100) = 1101. The function performs binary summation. The MSB of the result are

stored in the buffer *B*. Finally, it returns the buffer *B* to the client. The result for keyword frequency comparison without encryption is listed in Table.7.

Table.7. Matching Document Computation without Encryption Results

***f*(1100)=11 *ft*(1100)=11** Two's Complement of *ft*=1101 Sum *f*(1100) + -*ft*(1100) = 00000 Sum[0] = 0, Add Sum[0] value to the Buffer *B*
***f*(111)=11 *ft*(111)=101** Two's Complement of *ft*=1011 Sum *f*(111) + -*ft*(111) = 11100 Sum[0] = 1, Add Sum[0] value to the Buffer *B*
***f*(1110)=10 *ft*(1110)=10** Two's Complement of *ft*=1110 Sum *f*(1110) + -*ft*(1110) = 00000 Sum[0] = 0, Add Sum[0] value to the Buffer *B*
***f*(11)=11 *ft*(11)=101** Two's Complement of *ft*=1011 Sum *f*(11) + -*ft*(11) = 11110 Sum[0] = 1, Add Sum[0] value to the Buffer *B*
***f*(1000)=10 *ft*(1000)=101** Two's Complement of *ft*=1011 Sum *f*(11) + -*ft*(11) = 11101 Sum[0] = 1, Add Sum[0] value to the Buffer *B*
Collect all the Buffer value *B*={0,1,0,1,1} $c_0 \vee = B_i \oplus 1$ $c_0=(0 \oplus 1) \vee (1 \oplus 1) \vee (0 \oplus 1) \vee (1 \oplus 1) \vee (1 \oplus 1)$ $c_0=1$ So, The given *m* is the matching document

The matching document computation is carried out in both encrypted text and plain text. Thus, using fully Homomorphic encryption the searching can be performed even in encrypted domain.

5. CONCLUSION

On the account of the fully Homomorphic encryption techniques, disjunctive threshold queries based on keyword frequency has been presented. It has been believed that the protocols for private threshold queries based on keyword frequency will be made practical with the performance improvement of fully homomorphic encryption techniques in the future.

Privacy becomes a major concern and it is securing higher attention among the users. In cloud computing, it will become viable only if privacy of users is completely protected. For example, Google Alerts [15] is a service offered by the Google, the service sends emails to the user when it finds new results such as web pages, newspaper articles, or blogs that match the user's search criteria. In this the search criteria should be kept classified to Google. This should evade the situation of AOL search data leak [16].

By this private searching, it is viable for a user to construct a filter program according to the frequencies of some classified keywords and submit it to Google, which executes the program on all latest Web and news pages. The program lists the relevant pages to the user as its discovery conferring to the search criteria specified by them. Whereas the program is executed by Google, the search criteria of the user can be kept confidential to Google.

REFERENCES

[1] Craig Gentry, "Fully Homomorphic Encryption Scheme", Ph.D Thesis, Department of Computer Science, Stanford University, 2009.

[2] Marten Van Dijk, Craig Gentry, Shai Halevi and Vinod Vaikuntanathan, "Fully Homomorphic Encryption over the Integers", *Proceedings of 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Vol. 6110, pp. 24-43, 2010.

[3] Rafail Ostrovsky and William E. Skeith, "Private Searching on Streaming Data", *Journal of Cryptology*, Vol. 20, No. 4, pp. 397-430, 2007.

[4] John Bethencourt, Dawn Song and Brent Water, "New Construction and Practical Applications for Private Stream Searching (Extended Abstract)", *Proceedings of IEEE Symposium on Security and Privacy*, pp. 132-139, 2006.

[5] J. Bethencourt, D. Song and B. Waters, "New Techniques for Private Stream Searching", *ACM Transactions on Information and System Security*, Vol. 12, No. 3, pp. 1-32, 2009.

[6] N.P. Smart and F. Vercauteren, "Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes", *Proceedings of 13th International Conference on Practice and Theory in Public Key Cryptography*, Vol. 6056, pp. 420-443, 2010.

[7] Damien Stehle and Ron Steinfeld, "Faster Fully Homomorphic Encryption", *Proceedings of 16th International Conference on the Theory and Application of Cryptology and Information Security*, Vol. 6477, pp. 377-394, 2010.

[8] David Money Harris, and Sarah L. Harris, "*Digital Design and Computer Architecture*", Morgan Kaufmann, 2007.

[9] Caroline Fontaine and Fabien Galand, "A Survey of Homomorphic Encryption for Nonspecialists", *Journal of Information Security*, Vol. 1, pp. 41-50, 2009.

[10] Craig Gentry, "Computing Arbitrary Functions of Encrypted Data", *Communications of the ACM*, Vol. 53, No. 3, pp. 97-105, 2009.

[11] Shafi Goldwasser and Silvio Micali, "Probabilistic Encryption and How to Play Mental Poker Keeping Secret all Partial Information", *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, pp. 365-377, 1982.

[12] Pascal Paillier, "Public Key Cryptosystems Based on Composite Degree Residuosity Classes", *Proceedings of International Conference on the Theory and Application of Cryptographic Techniques*, Vol. 1592, pp. 223-238, 1999.

[13] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky and Giuseppe Persiano, "Public Key Encryption with Keyword Search", *Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques*, Vol. 3027, pp. 506-522, 2004.