POIS Project Report
Team 61

Likhith Asapu 2020114015
Nitin Rajasekar 2020101117
Aakash Terala 2020111023

Abstract

This paper considers the problem of performing keyword searches on encrypted data stored on a remote server. It proposes a scheme that allows a user to encrypt their files and later retrieve some of them based on specific keywords, without revealing the keywords or compromising the security of the files. The scheme is efficient and incremental, and does not require a public-key cryptosystem. The paper analyzes the security and performance of the scheme and compares it to previous work on the same problem.

1 Problem Setting

The problem of Privacy Preserving Keyword Searches on Remote Encrypted Data (PPSED for short) is defined in this section, and will hereafter use PPSED to denote this problem.

The formal definition of PPSED is as follows:
PPSED is a multi-round protocol between a remote file server S and a user U. The server S has a set of n encrypted files $\zeta = E_1(m_1), E_2(m_2), ..., E_n(m_n)$ where for each i ϵ [n], E_i is an encryption function and m_i is a file. The user U has decryption algorithms $D_1, D_2, ..., D_n$ such that $D_1(E_1(m_1) = m_1, D_2(E_2(m_2) = m_2, ..., D_n(E_n(m_n) = m_n$. Moreover, in each round j ϵ N, U prepares a keyword $w_j \ \epsilon \ \{0,1\}^*$.

An implementation of PPSED with security parameter t must satisfy the following :

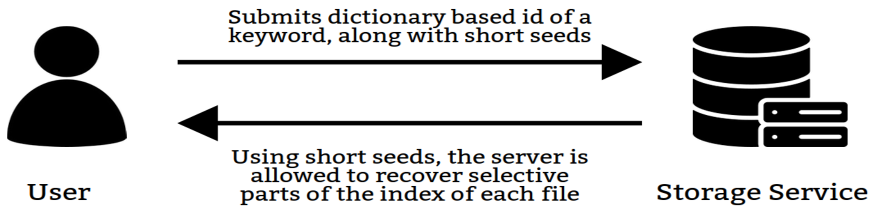
1. Correctness: In round j, for i ϵ [n], if w_j is a keyword of m_i, U can obtain $E_i(m_i)$
2. Limits on the bandwidth and storage space:
   - In round j, the number of bits sent from S to U is $\sum_{i \epsilon I_j} |E_i(m_i)| + O(1)$, where $I_j = \{i|$ i ϵ [n], w_j is a keyword of $m_i\}$.
   - The number of bits stored on U is O(t).

- The number of bits sent from U to S is O(t) per keyword search.

3. Security Requirement:

   - For k ϵ N, let C_k be all the communications S receives from U before round k, and let $C_k^* = \{\zeta,\ Q_0 \equiv \phi, Q_1, ..., Q_{k-1}\}$, where for each j ϵ [k-1], Q_j is an n-bit string such that for i ϵ [n], $Q_j[i] = 1$ if and only if w_j is a keyword of m_i.
   - For k ϵ N, for any PPT algorithm A, any $\Delta_k = \{m_i, ..., m_n,\ w_0 \equiv \phi, w_1, ..., w_{k-1}\}$, any function h, there is a PPT algorithm A^* such that the following value is negligible in t:

$$|Pr[A(C_k, 1^t) = h(\Delta_k)] - Pr[A^*(C_k^*, 1^t) = h(\Delta_k)]$$


Submits dictionary based id of a keyword, along with short seeds
Using short seeds, the server is allowed to recover selective parts of the index of each file
User
Storage Service

2 Scheme 1

Let s and r be two random keys that the user picks from the set 0,1t, where t is a parameter that determines the security level.
The user builds a keyword index for each file, as described before. The index is a sequence of bits (0 or 1) that indicate which keywords from the dictionary are in the file.
For example, if the file has keywords 2 and 3, but not 1 or 4 (according to the dictionary order), then the index for that file would be 0110 . But, if the user just sends the index as it is to the server, the server might be able to figure out which keyword is in which file... In this approach, a PRP P with a secret key s is used. The keyword index has a 1 at the position P(i) if the file contains the keyword i, instead of the position i itself.
However, this approach has some drawbacks. For example, the number of 1s in the keyword index would reveal how many keywords are in the file. It would also be possible to infer if two files have common keywords, and thus learn something about the file structure.
To address this, two PRPs F and G are applied. F takes r as the key and i as the input, where i is the keyword index. The output of F is then used as the key for G, with j as the input, where j is the file index.
For each of the n keywords, G produces a bit of length n after applying the PRP and PRF. This bit has the same length as the keyword index.
The keyword index and the bit are XORed together. The user sends this bit and the encrypted files to the server. The server cannot tell which files are in

which file, or what the keywords are. The user keeps the secret keys r, s, and the keyword dictionary on his device. The server sends the file to the user if the result is 1, otherwise it does not.

Retrieval:
The user sends $p = P(\lambda)$ and $f = F(p)$ to the server if they want to find files that contain the keyword with the index λ in the dictionary.
The server can use p to bypass the PRP obstacle of the keyword index, but only for that specific keyword that the user wants. It cannot do that for any other keyword index because it does not have the key s.
The server can also use f to overcome the PRF obstacle of the keyword index, but again only for that specific keyword that the user wants. The server calculates $K[p] \oplus G(j)$ to check if keyword i is in file j.

2.1 Theorem:

Scheme1 is a correct implementation of PPSED where S sends $\sum_{i \epsilon I_j} |E_i(m_i)|$ total bits. U stores 2t bits plus a dictionary of constant size, and U sends (d+t) bits per keyword search.

2.2 Proof:

In the following, by "the view of S" we mean all the communications S receives from U. Let ζ denote $\{E_1(m_1), E_2(m_2), ..., E_n(m_n)\}$. Next, let

$$I(a) = \{I_1[a], I_2[a], ..., I_n[a]\},$$
$$M(a) = \{M_1[a], M_2[a], ..., M_n[a]\},$$
$$G(a) = \{G_a(1), G_a(2), ..., G_a(n)\}$$

and let $M = \{M(1), M(2), ..., M(2^d)\}$. Moreover, let λ_v denote the dictionary index of the keyword in round v, and define $p_v = P_s(\lambda_v)$ and $f_v = F_r(p_v)$. In addition, let C_v denote the view of S before round v, so we have

$$C_1 = \{\zeta, M\}, C_2 = \{\zeta, M, p_1, f_1\}, C_3 = \{\zeta, M, p_1, p_2, f_1, f_2\}, ...$$

Consider the ideal case which meets our security requirement perfectly: U records in advance a set of linked lists such that each file index is associated with a list of all the keywords of the corresponding file. In this case, the only message U needs to send in round v is the n-bit string Q_v such that for j ϵ [n], $Q_v[j] = 1$ if and only if m_j contains the keyword in round v (and S has to send back $E_j(m_j)$ back). SO if we let C_V^* denote the view of S before round v in the ideal case, we have

$$C_1^* = \{\zeta\}, C_2^* = \{\zeta, Q_1\}, C_3^* = \{\zeta, Q_1, Q_2\}, ...$$

Observe that $Q_v = I(p_v)$ for v ϵ $[2^d]$.

Our goal is to prove the following (for k ϵ $[2^d + 1]$): for any PPT algorithm A, any $\Delta_k = \{m_i, ..., m_n,\ w_0 \equiv \phi, w_1, ..., w_{k-1}\}$, any function h, there is a PPT algorithm A^* such that the following value is negligible in t:

$$\rho = |Pr[A(C_k, 1^t) = h(\Delta_k)] - Pr[A^*(C_k^*, 1^t) = h(\Delta_k)]$$

Intuitively, suppose A^* on input C_k^* an generate a view C_k' that is indistinguishable from C_k. Then A^* can simulate running A with C_k' to give the desired result (that is, that ρ is negligible in t). We shall follow this intuition.
For k = 1, A^* just needs to choose M' from $\{0,1\}^{n2^d}$ uniformly at random, and feeds A with $\{\zeta$, M'$\}$. We claim A^* is as desired as otherwise the pair (A, A^*) is a PPT distinguisher for pseudo-random bits and truly random bits. For k ¿ 1, the strategy of A^* is as follows:

- A^* chooses $f_1', f_2', ..., f_{k-1}'$ uniformly at random from $\{0,1\}^t$, and chooses $s' = (p_1', p_2', ..., p_{k-1}')$ uniformly at random from $S = \{s | s \subset \{1, 2, ..., 2^d\}, |s| = k-1\}$.
- A^* computes $M' = \{M'(1), M'(2), ..., M'(2^d)\}$ in the following way;
  - For i ϵ $[2^d]$, $i \neq p_1', p_2', ..., p_{k-1}'$, choose M'(i) uniformly at random from $\{0,1\}^n$.
  - For i ϵ [k-1], set $M'(p_i') = Q_i \oplus G(f_i')$
- A^* feeds A with $C_k' = \{\zeta, M', p_1', p_2', ..., p_{k-1}', f_1', f_2', ..., f_{k-1}'\}$.

We explain why this strategy works as follows. First, recall $Q_v = I(p_v)$ for v ϵ [k-1], and consider the following imaginary case: for each i ϵ $[2^d]$, $i \neq p_1, p_2, ..., p_{k-1}$, U does not generate M(i) according to Scheme1; instead, U chooses M(i) from $\{0,1\}^n$ uniformly at random. Clearly, in this case, the only difference between the generation of C_k' and the generation of C_k comes from the employment of truly randomness in place of pseudo-randomness. Specifically, C_k' is generated using truly random p_j' and f_j' for j ϵ [k-1], yet C_k is generated using pseudo-random p_j and f_j for j ϵ [k-1]. So we claim ρ must be negligible in t in this case as otherwise the pair (A, A^*) can be used to invalidate either P_K or F_K.
Next, consider the real case (that U does follow every step of Scheme1). An observation is for each i ϵ $[2^d]$, $i \neq p_1, p_2, ..., p_{k-1}$, M(i) remains pseudo-random before round k. However, since this is the only difference between the real case and the imaginary case, we claim ρ must be negligible in t in the real case as otherwise the pair (A, A^*) can be used to invalidate G_K. In consequence, we have proven the desired security guarantee.

2.3 Analysis:

We examine the practicability of the above scheme with realistic parameters. First, if we set d = 18, the storage overhead on server is 32 kilobytes per file. Note the latest Merriam-Webster's Collegiate Dictionary contains only 225,000 definitions. So even if U adds new words by himself, $2^1 8$ could be a reasonable

upper-bound in practice for the number of all the distinct words in U's dictionary as well as in his documents. Second, notably only a few bits are sent from U per keyword search. If we set t = 2030, for example, only 256 bytes are required. Clearly, the scheme is independent of the encryption method chosen for the remote files, so it works for different file formats (including compressed files, multimedia files, etc.), as long as a keyword index on the corresponding content can be built. Moreover, only pseudo-random functions (and permutations) are used in the construction of the scheme. As mentioned earlier, these functions can be implemented efficiently by heuristic algorithms.
Although we assume the availability of a dictionary on U's mobile device, the assumption is not far-fetched as most of today's mobile devices are equipped with built-in electronic dictionaries (or can store one on a memory card). Actually, if we estimate the average length of a keyword by 2^3 ASCII characters, a dictionary only amounts to $(2^{18})(2^3)(8) = 2$ megabytes, which can be improved further using compression.

3 Scheme 2

As mentioned earlier, it is unlikely that a mobile PDA device, which a user would use for mail retrieval, would have enough memory capacity to comfortably store the dictionary.
This causes difficulties as, in this case, the user would be unable to send the index of the keyword to the server, as they would not have the dictionary available to them.
The paper thus proposes a second scheme for this scenario.

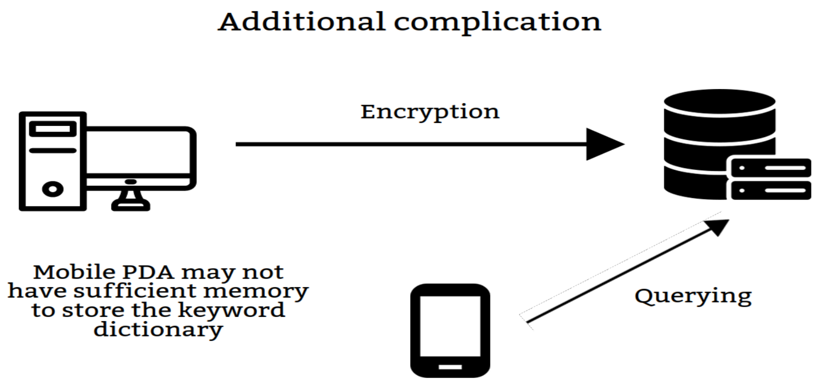
Similar to Scheme 1, the user begins by choosing two random keys s and r. Again similar to Scheme 1, the user creates the keyword index for each file, after running the keyword positions through the PRP P.

Here, the user utilizes an additional PRP, Φ, and chooses another secret key, τ.

Next the user sends $\phi_1 = \Phi_\tau(w_i)$, $\phi_2 = \Phi_\tau(w_i)$.. and so on, where $P(i) = j$, where j is the index of the keyword in the dictionary. This is done for all keywords. The pairs (j, ϕ_j) are stored on the server.

Again similar to Scheme 1, the same process with the PRFs is followed, except that instead of taking only i as the input, F takes the concatenation of i with Φ as input, for each keyword.
The same XOR operation involving G as Scheme 1 is performed. The user then stores the key r, s, and τ onto his mobile device. The user need not store the dictionary on his device in this case.
Unlike Scheme 1, this time the user does not have the luxury of being able to pass the index of the keyword he is interested in, as he/she does not have access to the dictionary.Essentially, the primary difference to scheme 1 is that here, there is an additional permutation and communication involved as the server needs to point the user to what the position of the keyword is, without knowing

it itself.


Additional complication
Encryption
Querying
Mobile PDA may not have sufficient memory to store the keyword dictionary

3.1 Theorem:

Scheme2 is a correct implementation of PPSED where S sends $\sum_{i\epsilon I_j} |E_i(m_i)|+d$ total bits. U stores 3t bits, and U sends $(w_m ax + t)$ bits per keyword search.

3.2 Proof:

We first prove U's security, employing some of the notation in the proof of the previous Theorem. Let $\tilde{C_k}$ denote the view of S bfeore round k in Scheme2. It suffices to prove the following: For k ϵ N, for any PPT algorithm $\tilde{A}$, any $\Delta_k = \{m_i, ..., m_n,\ w_0 \equiv \phi, w_1, ..., w_{k-1}\}$, any function h, there is a PPT algorithm A such that the following value is negligible in t:

$$|Pr[\tilde{A}(\tilde{C_k}, 1^t) = h(\Delta_k)] - Pr[A(C_k, 1^t) = h(\Delta_k)]$$

Recall C_k is the view of S before round k in Scheme1. In other words, we ask everything about Δ_k that can be computed given $\tilde{C_k}$ can also be computed given C_k. In other words, the information leakage of Scheme2 is essentially no worse than that of Scheme1.
Since the retrieval phase is interactive, we know U's ongoing action depends on S's message, namely p. So we must consider the case that S might dishonestly send an arbitrary p' $\neq$ p to U. However, let us start from the simplified case that S always sends the correct p to U.
In the simplified case, we can assume w.l.o.g. that U always sends back p, along with f, to S. Note this does not jeopardize U's security since U learns p from S, while the difference between $\tilde{C_k}$ and C_k now comes from $\{\varphi_j\}_{j\epsilon[2^d]} + \{\varphi = \varphi_p\}$. An observation is A can simulate each φ_j by flipping coins and can simulate φ by setting φ to be the simulated φ_p, in that each φ_j represents t pseudo-random bits and p is known to A (as p ϵ C_k is part of the input to A). So all A needs to do is to feed $\tilde{A}$ with C_k and the simulated results.
Next, let us consider the case that S might be dishonest. Note if S sends p'

$\neq$ p to U, then the returning message from U, namely $f' = F^*_k(p', \varphi)$, cannot be used for decryption and represents nothing more than t pseudo-random bits. Hence we can assume w.l.o.g. that S always simulate f' by flipping t coins and discarding f' from his view ($\tilde{C_k}$) in this case. Accordingly, it is enough to prove that for all k, for any PPT algorithm $\tilde{A}$, any $\Delta_k = \{m_i, ..., m_n,\ w_0 \equiv \phi, w_1, ..., w_{k-1}\}$, any function h, there is a PPT algorithm A such that the following value is negligible in t:

$$|Pr[\tilde{A}(\tilde{C_k}, 1^t) = h(\Delta_k)] - Pr[A(C_k, 1^t) = h(\Delta_k)]$$

where $c_k \subset C_k$ is the reduced C_k defined as follows: c_k is constructed by mimicking S's dishonest behavior to discard the corresponding f from C_k. Clearly, A just needs to do the same simulations as in the simplified case and feeds $\tilde{A}$ with c_k and the simulated results. Hence, we have finished the security proof by describing this PPT algorithm A.
There server S must send an additional d bits (namely p) beyond the files themselves. The on-mobile-device dictionary is replaced by a small storage overhead of t bits (namely the key to Φ_k). Moreover, we claim the correctness follows the fact that Φ_k is injective, and the user-side communication complexity can be easily verified.

3.3 Analysis:

If we estimate the maximal length of a word by 24 ASCII characters, we have $w_{max} = (2^4)(8) = 128$. Hence the encrypted dictionary amounts to $(2^{18})(128)$ = 4 megabytes per user. The server-side storage overhead is the same with Scheme1. On the other hand, the communication complexity changes only slightly: S needs to send additional d bits per keyword search, while U now needs to send (w_{max} + t) bits per keyword search.