POIS Project

Privacy Preserving Keyword Searches on Remote Encrypted Data

Team 61

Likhith Asapu - 2020114015

Nitin Rajasekar - 2020101117

Aakash Terala - 2020111023

Agenda.

Introduction
- Goal
- Challenges
- Previous Approaches

Preliminaries

Basic Ideas on
- Scheme 1
- Scheme 2
- Secure Update

Demo of the implementation

Rudimentary Goal

Challenges

- As the files are encrypted, there is no straightforward method for the server to perform keyword search (without the decryption key).

- The server cannot simply send all the encrypted files to the user, as the device would likely have limited bandwidth and storage.

- The User would prefer to keep the keyword he is interested secret.

All that the server may learn is this - ' The encrypted files returned to the user share some keyword.

Previous approaches

Encrypting each word:

An approach recommended by a previous paper was to encrypt the file word by word, meaning that each word would have the same encryption, and thus be searchable. However, this approach is vulnerable to classical frequency analysis.

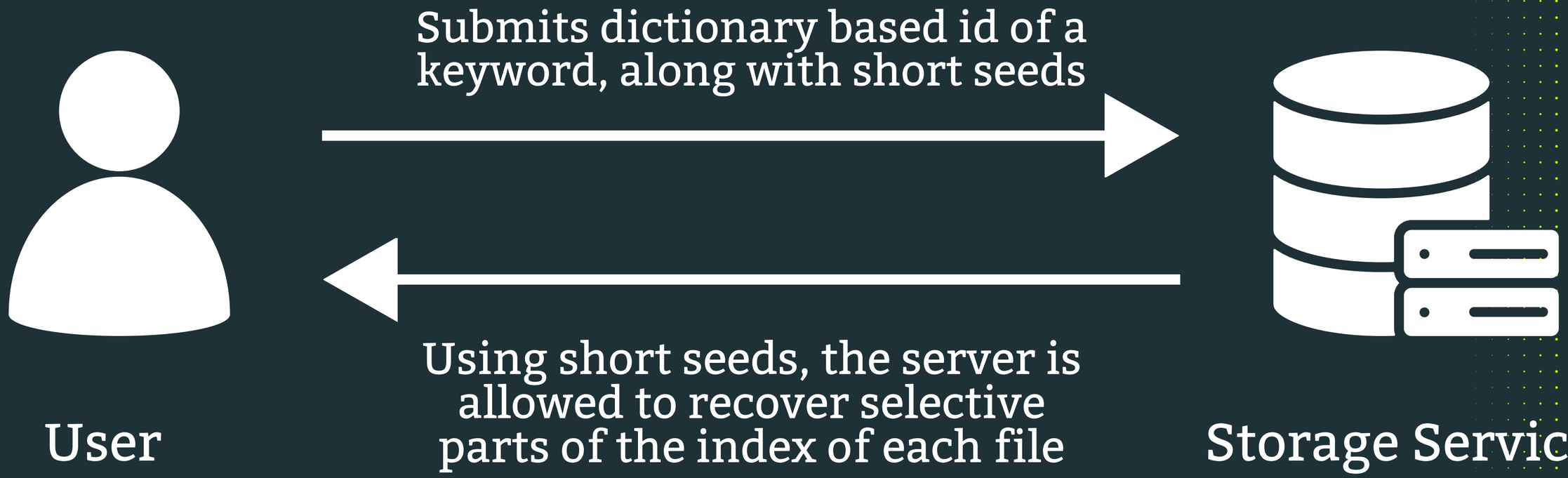
Secure multi-party computation:

Secure multi-party computation (MPC) allows multiple parties to compute a function on their private inputs without revealing their inputs to each other. This approach can be used to perform keyword search on encrypted data. However, MPC requires significant computational overhead and is not practical for large-scale applications.

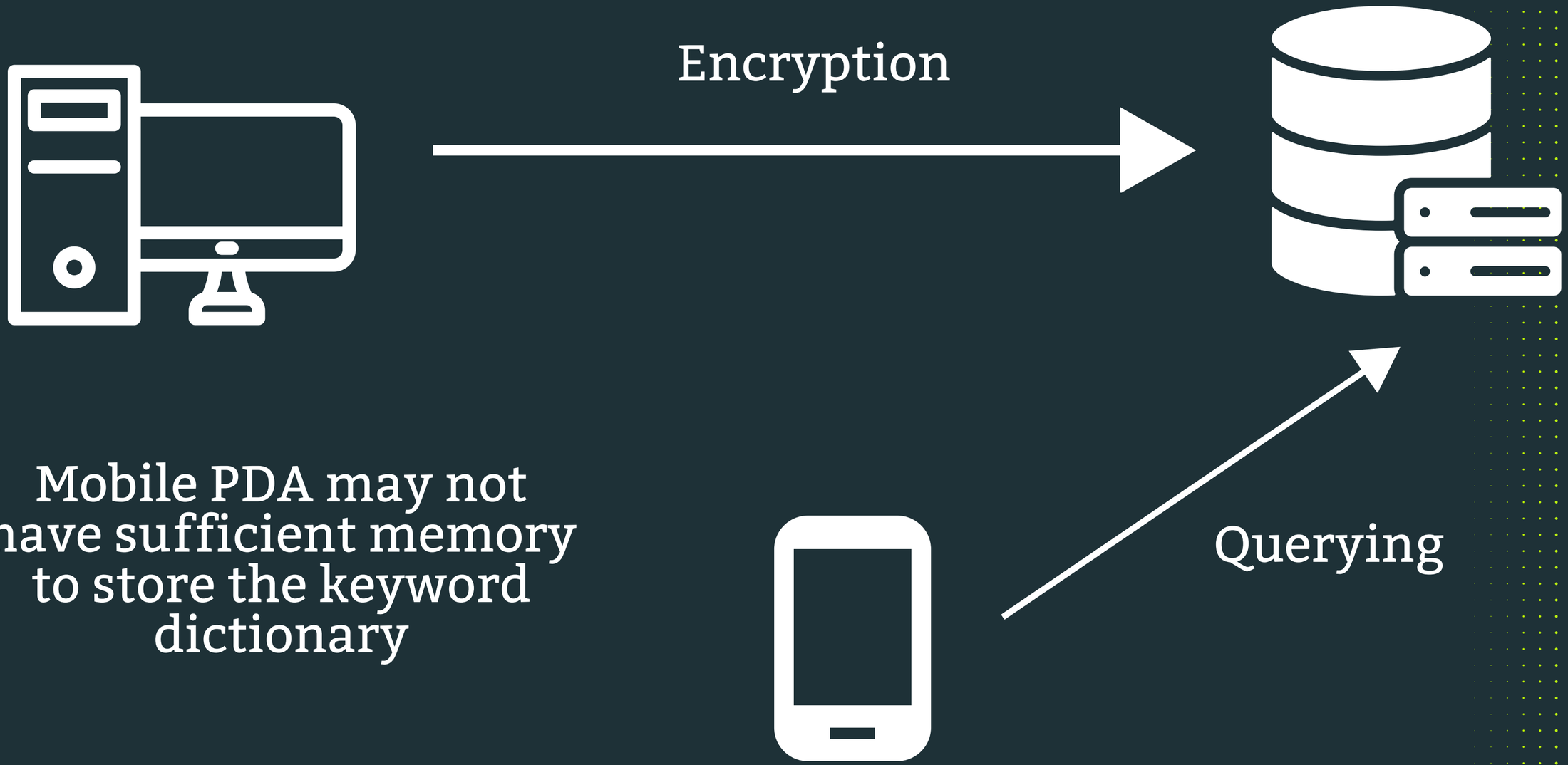
Basic concept

The solution makes use of

- a dictionary which maps each keyword to a unique number
- a keyword index for each file, which indicates which keywords are present within it



Additional complication



Mobile PDA may not have sufficient memory to store the keyword dictionary

Shortcomings

The most used PDA Device in 2004 was the Motorola RAZR V3 - it had 5.5 MB of internal memory.

The paper performed some analysis on what the typical size of a dictionary might be - assuming a reasonable number of keywords (using the Merriam-Webster), and 256 bytes per keyword search, the size of the dictionary would come to 2 megabytes.

A typical mobile device would generally not possess such storage capacity, meaning an improvised scheme would have to be devised.

Pseudorandom Generator (PRG)

- To create a PRG, we first need to determine a hardcore predicate. For our implementation we used DLP (Discrete Log Problem).
- We first define a constructor, $G_1(s) =< g^s \bmod p, msb(s) >$ where g is any arbitrary number, p is a large prime number, s is unique seed.
- We decide msb(s) by,

$$msb(s) = \begin{cases} 0 & \text{if} \quad s < \dfrac{p-1}{2} \\ 1 & \text{otherwise} \end{cases}$$

- We then use this constructor to construct our final PRG, by following these steps,

1. Let $s \in \{0,1\}^n$ be the seed, and denote $s_0 = s$.
2. For every $i = 1, \ldots, p(n)$, compute $(s_i, \sigma_i) = G_1(s_{i-1})$, where $\sigma_i \in \{0,1\}$ and $s_i \in \{0,1\}^n$.
3. Output $\sigma_1, \ldots, \sigma_{p(n)}$

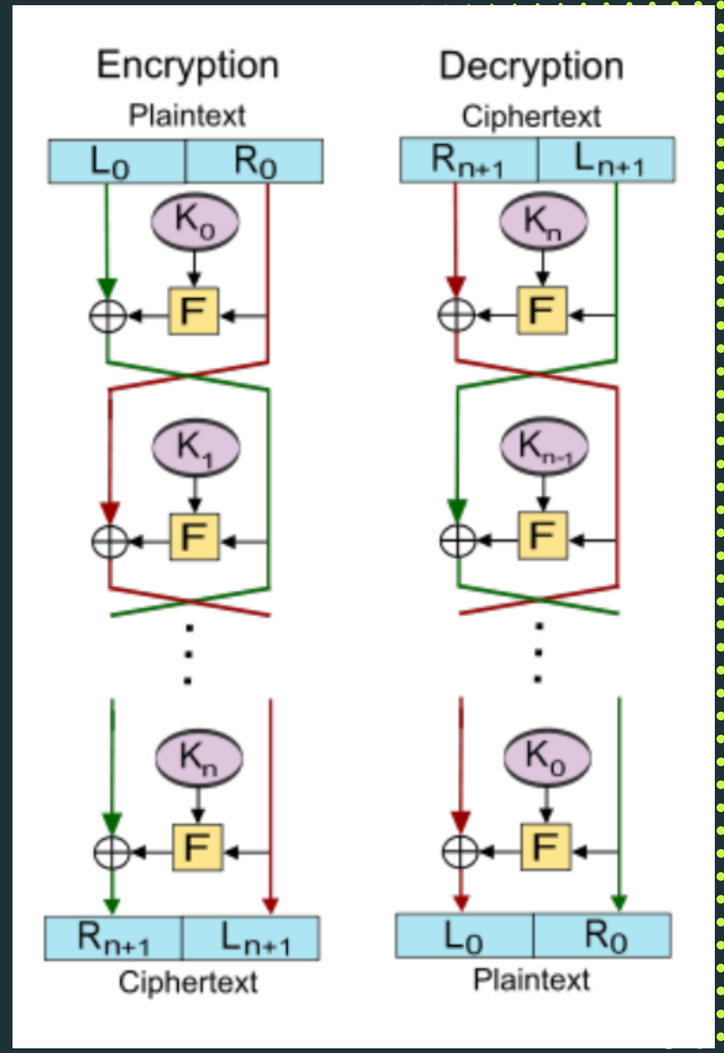
Pseudorandom Function (PRF)

- To construct PRFs, we make use of a secure PRG. For our implementation we use the previously constructed PRG.

- PRFs use a key, k, as a parameter while generating output, this key is kept secret.
- Using key k, and input binary string x of length n, we define PRF F as, $F_k(x_1x_2\cdots x_n) = G_{x_n}(\cdots(G_{x_2}(G_{x_1}(k)))\cdots)$
- Here, G is the PRG we implemented earlier, and G0(k) is defined as first n bits of the output of G and similarly G1(k) is defined as the second n bits of the output of G.

Pseudorandom Permutation (PRP)

- A PRP is essentially a PRF that is a bijection (one-to-one or invertible).
- To construct a PRP, we followed the Luby-Rackoff construction, which says that we can construct a strong PRP using PRFs by using the concept of Feistel Networks/Cipher.
- As shown, the Feistel Cipher uses a PRF F, and a sequence of keys for each round to generate a final output.
- We perform 3 rounds of the Feistel Cipher which provides a strong PRP.



Scheme 1

- The user chooses two random keys *s* and r, where $s, r \in \{0,1\}^t$ Here, t is the security parameter.
- For each file, we the user creates the aforementioned keyword index. Based on the keyword dictionary, the bits of the index (for a particular file) are set or 0 or 1 depending on whether it contains the concerned keyword.

  For instance, if the file contains keywords 2 and 3, but not 1 or 4 (numbering based on the dictionary), then the index for that file would be 0110 .
- However, if the user directly sends the created index to the server, the server may be able to directly figure exactly which keyword is present in which file..

Scheme 1

- This is where a PRP *P* is utilized. If the file contains a keyword *i*, it is not the position *i*, but the position $P_s(i)$ that is set to 1 in the keyword index.
  Here, s is the secret key chosen at the beginning.

- This method would still have its weaknesses. For instance, if there are *x* keywords present in a file, the keyword index would have *x* bits set to 1. It would also be able to tell if two files share the same keywords, and thus gain information about file structure.

- Two PRPs *F* and *G* are then utilized.
  F takes *r* as the key, and the keyword index *i* as input.
  The output of *F* is then used as the key for *G*, with *j*, the file index, being *G*'s input.

Scheme 1

- This is done for every keyword - so if there are n keywords present, a string of length n would be obtained as the output of *G*. Same as the length of the keyword index.
- The bits of this string are then XORed with the bits of the keyword index.
- So finally this string, generated after processing with the PRP and PRF, is sent by the user to the server along with the encrypted files.
- From this information, the server has no way of knowing which files are present within which file, or what the keywords are.
- The user stores the secret keys *r*, *s*, and the keyword dictionary on his device.

Scheme 1 - Retrieval

- If the user is interested in files with the keyword whose index is λ in the dictionary, then they send $p = P_s(\lambda)$ and $f = F_r(p)$.
- With p, the server can now get past the PRP barrier of the keyword index, but it can only do so for that specific keyword which the user is interested in.
  As it does not have the key s, it cannot do so for a keyword of any other index.
- Similarly, with f, the server can now get past the PRF barrier of the keyword index, but it can only do so for that specific keyword which the user is interested in.
- So to determine whether keyword i is present in file j, the server computes $K_j[p] \oplus G_f(j)$.
- If the value so computed is 1, the file is sent to the User - else it is not.

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.

Scheme 2

- Similar to Scheme 1, the user begins by choosing two random keys s and r, where $s, r \in \{0,1\}^t$
- 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 ,φ1 = Φτ (wi_1), φ2 = Φτ (wi_2).. and so on, where P(i_j) = j, where j is the index of the keyword in the dictionary. This is done for all keywords. The pairs (j, φj) are stored on the server.

Scheme 2

- 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 φi 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.

Scheme 2 - Retrieval

s

- 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.
- To retrieve a keyword w_λ the user sends to the server $\Phi_\tau(w_\lambda)$
- The server then sends back, to the user, the value p such that the pair (p, φp) exists on the server.
- The user then, as in scheme 1, sends F (p, φ) to the server. The server can perform the same XOR operations as in scheme 1 to access the keyword index for that particular word.
- 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.

Secure Update

- This section tells us how to securely submit new files to server S.
- The User U can follow all the steps in the first phase of Scheme 1 or Scheme 2 to submit a new set of files, but some additional care is necessary.
- The issue arises when U uses old pseudorandom seeds. In such a case, for any keyword that U queried, S can learn whether the newly added files contain the unknown keyword, as S already knows the corresponding pseudorandom seed.
- This means that the new files suffer from a-priori information leakage before any query.

Secure Update

- The solution to this problem is to use different pseudorandom seeds for each set of files (between each retrieval)
- This is achieved by first choosing a truly random seed, which is then used to generate several pseudorandom seeds rθ for each set of files.
- When U makes a query, for each rθ, generate another pseudorandom seed corresponding to the keyword index, and send all of them to S, who then decodes the encrypted index accordingly.