Search on Encrypted Data

Ziad El Bizri
November 22\textsuperscript{nd}, 2004
Motivation

- Goals
  - Store data on data storage servers (mail servers, file servers) while preserving privacy and security
  - Provide functionalities: add a document, retrieve documents, search for documents containing specific keywords

- Mechanism: store data in an encrypted form on the server side, and provide a search mechanism over the encrypted data

- Server is not trusted!!!
Practical Techniques for Searches on Encrypted Data

\cite:
- Dawn Song, David Wagner and Adrian Perrig
- IEEE Symposium on Security and Privacy 2000

Objectives:
- Mechanisms are provably secure
- Search in $O(N)$ time, where $N$ is the total number of documents (linear search)
- Supports controlled searching (the server cannot search for a word not provided by the user), hidden queries (the word that is searched for is unknown to the server) and query isolation (server only learns the search results)
Provable Security

- Different from perfect secrecy in the information theoretic sense
- Provable security is concerned with the resources needed by an attacker to break the cryptosystem, i.e. computational security
- We wish to compute the advantage of an attacker
- Formally
  - $A: \{0, 1\}^n \rightarrow \{0, 1\}$ is an algorithm
  - $X$ and $Y$ are random variables distributed in $\{0, 1\}^n$
  - $Adv A = | Pr[A(X) = 1] - Pr[A(Y) = 1] |$ (Distinguishing probability)
Primitives

- Pseudorandom generator $G$ (e.g. stream cipher)
  - $G: K_G \_ S$ is a $(t, e)$-secure pseudorandom generator if every algorithm $A$ with running time at most $t$ has advantage $\text{Adv } A = |\Pr[A(G(U_{Kg})) = 1] – \Pr[A(U_S) = 1]| < e$, where $U_{Kg}$ and $U_S$ are randomly distributed over $K_G$ and $S$ respectively.

- Pseudorandom function $F$ (e.g. hash function)
  - $F: K_F \times X \_ Y$ is a $(t, q, e)$-secure pseudorandom function if every algorithm $A$ making at most $q$ oracle queries with running time at most $t$ has advantage $\text{Adv } A = |\Pr[A^Fk = 1] – \Pr[A^R = 1]| < e$, where $R$ represents a random function selected uniformly from all the maps from $X$ to $Y$. 
Primitives (2)

- Pseudorandom permutation $E$ (e.g. block cipher)
  - $E: K_E \times Z \rightarrow Z$ is a $(t, q, e)$-secure pseudorandom permutation if every algorithm $A$ making at most $q$ oracle queries with running time at most $t$ has advantage $\text{Adv}_A = |\Pr[A^{E_k, E_k^{-1}} = 1] - \Pr[A, \psi^{-1} = 1]| < e$, where $\psi$ represents a random permutation selected uniformly from all the set of all bijections on $Z$
  - Note that the attacker is given an oracle for encryption as well as decryption _adaptive chosen-plaintext/ciphertext attack model_

- Intuition: $(t, q, e)$-security represents resistance to attacks that use at most $t$ offline work and at most $q$ adaptive chosen-text queries
Set up

- Alice wants to encrypt a document which is the sequence of words $W_1, \ldots, W_l$
- For now assume that all words are of equal length $n$ (e.g. use padding for shorter words, and split longer words)
- She has the following primitives
  - $G: K_G \rightarrow X^l$ is a pseudorandom generator for some $l$ and $X = \{0, 1\}^{n-m}$
  - $F: K_F \times X \rightarrow Y$ is a pseudorandom function, $X = \{0, 1\}^{n-m}$ and $Y = \{0, 1\}^m$
  - $E: K_E \times Z \rightarrow Z$ is a pseudorandom permutation, $Z = X \times Y = \{0, 1\}^n$
Scheme I: The Basic Scheme

- **Step by step**
  - Alice generates a sequence of pseudorandom values $S_1, \ldots, S_l$ using $G$ where each $S_i$ is $n-m$ bits long.
  - She computes $T_i = [S_i, F_{k_i}(S_i)]$
    She outputs the ciphertext $C_i = W_i \oplus T_i$

- **Properties**
  - The key $k_i$ can be the same or chosen independently for every word.
  - If $F$ and $G$ are secure, then the sequence of $T_i$ is a secure pseudorandom generator.
Scheme I (2)
Scheme I (3)

- **Search**
  - If Alice wants to search for the word $W$, she can tell Bob (the server) the word $W$ and the $k_i$ corresponding to each location $l$ in which $W$ may occur.
  - Bob then searches the document in the ciphertext by checking whether $C_i \oplus W_i$ is of the form $s$, $F_{k_i}(s)$ for some $s$.

- **Properties**
  - Limited controlled search: Bob can only search regions of the text for which Alice has provided the keys $k_i$.

- **Problems**
  - Alice should know in advance the positions at which $W$ may appear and ends up revealing all keys $k_i$ to Bob.
Scheme II: Controlled Searching

- Alice uses an additional pseudorandom function $f: K_F \times \{0, 1\}^* \rightarrow K_F$ keyed with $k'$, independently of $F$
  - Now she uses $k_i = f_{k'}(W_i)$
  - If Alice wants Bob to search for $W$, she reveals $W$ and $f_{k'}(W)$ and Bob will not be able to learn anything on locations $i$ where $W_i \neq W$

- Discussion:
  - The generation of $k_i$ is flexible
  - Still the main problem is that Alice has to reveal $W$ to Bob
Scheme III: Support for Hidden Searches

- Alice wants to search for a word $W$ but is not ready to reveal it to Bob
  
  - She should preencrypt each word $W$ of the clear text using a deterministic encryption algorithm $E_{k''}$ (e.g. ECB encryption of words) such that $X_i = E_{k''}(W_i)$
  
  - Generate $T_i = [S_i, F_k(S_i)]$
    
    Output the ciphertext $C_i = X_i \oplus T_i$

- Search
  
  - Alice computes $X = E_{k''}(W_i)$ and $k = f_k(X)$, and sends $[X, k]$ to Bob

- All the desired properties are satisfied
Scheme III (2)
Scheme IV: The Final Scheme

- Problem with Scheme III
  - Given the ciphertext only, Alice cannot decrypt arbitrary words
  - Recall: $C_i = X_i \oplus T_i$ and $k_i = f_{k'}(X_i)$, she needs to know $X_i$ before computing $T_i$ and decrypting $C_i$

- Solution
  - Split $X_i$ into $L_i, R_i$ where $L_i$ is $n$-m bits long, and $R_i$ is m bits long
  - Now use $k_i = f_{k'}(L_i)$
  - To decrypt a random entry, Alice obtains $L_i = C_i \oplus S_i$ (the first $n$-m bits), and is able to generate $k_i$ and finally she can recover $R_i$
Scheme IV (2)
Discussion

- Easy to add new entries
- Only a set of master keys (very small key management overhead)
- If wildcards are used, all possible combinations must be generated and queried
- Having variable length words, allow statistical attacks by the server
- An additional set of indices to keywords can be encrypted and stored to increase efficiency is search
Efficient Tree Search in Encrypted Data

\cite

- R. Brinkman, L. Feng, J. Doumen, P.H. Hartel and W. Jonker
- 2nd International Workshop on Security in Information Systems ,
  April 2004

Objectives

- Algorithm that is suitable for XML databases
- Use the structure of XML documents to handle queries more efficiently
- Implement both search algorithm (linear and tree search) and compare their relative efficiency
Main Idea

- Parse the XML document and save additional fields on top of encrypted data
  - Pre: the number of open tags
  - Post: the number of closed tags
  - Parent: the pre value of the parent element

- XPath properties are simple to represent
  - v is a child of v': v.parent = v'.pre
  - v is a descendant of v': v'.pre < v.pre AND v'.post > v.post
  - v is following v': v'.pre < v.pre AND v'.post < v.post
Example

<table>
<thead>
<tr>
<th></th>
<th>pre</th>
<th>post</th>
<th>parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;a&gt;</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&lt;b&gt;</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>&lt;/b&gt;</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;c&gt;</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>d=”…”&gt;</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>&lt;e/&gt;</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>&lt;/c&gt;</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;/a&gt;</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:
- Ascendants
- Following siblings
- Previous siblings
- Descendants

Pre-

Post-

---

November 22, 2004

CS590T - Presentation
Encryption

- Do a one pass over the whole document to compute all the additional fields \((pre, post, parent)\)
- Use the same method as described in the previous paper, but this time store \(pre, post, parent, C_{tag}, C_{value}\)

Discussion: doesn’t this lead to attacks in which the structure of the document is leaked to the server? Is this of any concern?
Search

- Given a search query /$tag_1$/$tag_2$[$tag_3$ = 'value']
- Perform a linear search on the first tag
- Do a SQL subquery to return all the entries that satisfy $\text{tag}_2$ is a descendant of $\text{tag}_1$ (use the pre, post and parent information)
- Perform a new linear search to find $\text{tag}_2$
- Do this iterative process over and over again until the desired result is found
  - Not all space is searched, however search time is highly dependent on the structure of the XML document!!