Advanced Topics in Cryptography

Lecture 12: Search on encrypted data.

Benny Pinkas

An announcement

- Seminar talk, this Wednesday:

  Hovav Shacham

  New paradigms in signature schemes
- Abstract:
  - Groups featuring a computable bilinear map are particularly well suited for signature-related primitives.
  - For some signature variants the only construction known is based on bilinear maps.
  - Bilinear-map-based constructions are simpler, more efficient, and yield shorter signatures.
  - The talk describes three constructions and their applications: short signatures, aggregate signatures, group signatures.

Related papers

- Practical Techniques for Searches on Encrypted Data . D. Song, D. Wagner and A. Perrig.
- Public Key Encryption with Keyword Search. D. Boneh, G. Di Crescenzo, R. Ostrovsky and G. Persiano.

Search on encrypted data

- Today, mail and file servers must be fully trusted
  - They can store encrypted data, but users must download all data, or know which part of the data to download.
- Motivation: Store encrypted data on a remote server, while being able to perform searches in order to decide which parts to download.
- Applications:
  - Searching on encrypted e-mails on mail servers, searching on encrypted files on file servers, searching on encrypted databases.
- Why is this hard?
  - Computations on encrypted data are often hard
  - Usual tradeoffs: security and functionality

Scenario

Search query

Download relevant emails

Remote user stores documents on remote server.

User can access each doc, but wants to minimize communication.

Each document is divided to words.

Remote user searches for docs which contain a specific word.

It receives these words, while server learns nothing.

Desired properties

- Security
  - Secrecy: encryption scheme is provably secure
  - Controlled search: server cannot search for arbitrary words
  - Query isolation: a search for one word does not leak information about other words
  - Hidden queries: a search does not reveal the search words
- Efficiency
  - Low computation overhead
  - Low space and communication overhead
  - Low management overhead

Two approaches to search

- Even without taking security into account, there are two possible ways to do search
  - Search sequentially over all stored data
  - Use an index

- We will see today a solutions based on sequential search

Reminder: pseudo-random functions

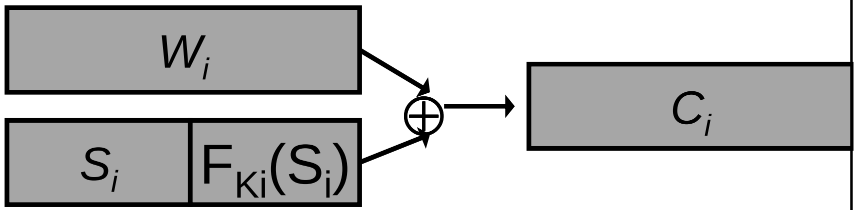
- A function F is a pseudo-random function if
  - F is keyed by a key k. A specific instantiation is $F_k()$.
  - The key k is chosen uniformly at random by Alice.
  - Bob does not know k.
  - Bob may ask Alice to compute $F_k(x)$ for values of x of his choice
  - Still, this does not give him any advantage in distinguishing $F_k(y)$ from random, for a value y different than all x for which he learned $F_k(y)$.
- A pseudo-random function can be instantiated using any block cipher.

Basic scheme

- A document is composed of equal length words (each *n* bits long) $W_1, W_2, \ldots, W_L$.
- Alice (remote user) uses only symmetric primitives:
  - A pseudo-random function F
  - A pseudo-random permutation E
- To encrypt a word W_i
  - Use a pseduo-random generator (stream cipher) to generate *n-m* bit long pseudo-random strings S_i.
  - Use a key k_i to compute an *m* bit string $F_{Ki}(S_i)$ from S_i.
  - To encrypt W_i compute pad $T_i = \langle S_i, F_{Ki}(S_i) \rangle$.
  - Ciphertext is $C_i = W_i \oplus T_i$, and it is stored on server.
  - The pad looks pseudo-random to the server.
  - To decrypt, Alice computes $S_1, \ldots, S_L$, and then $T_1, \ldots, T_L$.

The basic scheme

- To encrypt a word W_i
  - Compute an *m* bit string $F_{Ki}(S_i)$.
    - E.g. use same key $k=k_i$ for all i
  - Compute pad $T_i = \langle S_i, F_k(S_i) \rangle$.
  - Ciphertext is $C_i = W_i \oplus T_i$, and it is stored on server.



- Search:
  - Alice reveals the search word W and the key k to the server.
  - $\forall$i the server computes $W_i \oplus C_i$ and checks if it is of the form $\langle S_i, F_k(S_i) \rangle$. (If there is a match it returns the document to Alice.)
  - Note that $F_k(S_i)$ must be long enough to prevent false alarms.
  - Note that the search word W is revealed to the server.

- If Alice wants to enable search in specific locations only, she can use different k_i values and reveal only those corresponding to these location.

Controlled search

- Once the server learns k, it can search for any word it wishes in the locations which use this k ☹ We need to prevent the server from conducting arbitrary searches.
- Solution:
  - Alice uses an additional key k', and a pseudo-random function, to define $k_i = F_{k'}(W_i)$.
  - k' is never revealed to the server.
  - The pad is now $T_i = \langle S_i, F_{K_i}(S_i) \rangle$, and the ciphertext is $C_i = W_i \oplus T_i$.
  - When Alice wants to search for W, she reveals $F_{k'}(W_i)$ and W (but not k') to the server.
    - For any $W_j \neq W$, the server cannot distinguish $F_{k'}(W_j)$ from random, and cannot search for W_j.
  - There's actually a problem here, we'll discuss it later..

Hidden searches

- Problem: The previous scheme revealed W to the server.
- Basics of a new solution:
  - Alice picks a random key *k''* which she will keep secret
  - She encrypts each word and obtains $X_i=E_{k''}(W_i)$
  - She repeats the previous procedure, but now with X_i instead of W_i. She keeps k' and k'' to herself.
  - When she wants to search for W_i, she provides $k_i=F_{k'}(X_i)$ and X_i to the server (instead of $F_{k'}(W_i)$ and W_i).
- Problem (in decrypting the message):
  - The encrypted word is $X_i \oplus T_i$, where $T_i= \langle S_i,F_{Ki}(S_i)\rangle$, and $k_i=F_{k'}(E_{k''}(W_i))$.
  - To decrypt, Alice can compute S_i and therefore compute the first *n-m* bits of T_i and of X_i. But she doesn't know W_i and therefore cannot compute k_i, and the last m bits of S_i.

The final scheme

- Alice picks random k,k',k''
  - $X_i=E_{k''}(W_i)$. Define $X_i=L_i \mid R_i$, where $|L_i|=n-m$, $|R_i|=m$.
  - $k_i=F_{k'}(L_i)$
  - $T_i=\langle S_i, F_{Ki}(S_i) \rangle$
  - Ciphertext is $C_i = X_i \oplus T_i$
- To search for W, Alice provides k_i and $E_{k''}(W_i)$.

- To decrypt
  - Alice computes S_i.
  - Retrieves L_i by xoring S_i with the first n-m bits of C_i.
  - Computes $k_i=F_{k'}(L_i)$ and then $F_{Ki}(S_i)$.
  - Xors the result with the last m bits of T_i.

Public key encryption with keyword search

- Suppose that Bob sends encrypted email messages to Alice.
  - Messages are encrypted with Alice's public key. *They were not generated by Alice.*
  - Each message is accompanied by some encrypted keywords.
  - Messages and keywords are stored on a remote mail server.
  - Alice would like to search for messages which include specific keywords, and retrieve these messages alone.

The scenario

- Alice's public key is PK. She has a trapdoor T.
- We use a primitive which is denoted as PEKS (Public Key Encryption with Keyword Search).
- Bob sends a message to Alice
  – Wants to send a message msg with keywords $W_1,\ldots,W_k$.
  – Sends E_{PK}(msg), PEKS(PK,W_1),…,PEKS(PK,W_k)
- Alice wants to search for messages with keyword W
  – Alice computes a trapdoor T_W, as a function of T and W, T_W=Trapdoor(T,W).
  – She sends T_W to the server.
- The server has encrypted keywords of the form PEKS(PK,W'). It computes Test(PK,S,T_w), which outputs "yes" iff W=W'.

Security for PEKS

- PEKS(PK,W) must not reveal information about W unless T_W is available.
- The attacker
  - Adaptively asks for the trapdoor keys of many $T_{W'}$
  - Sends to Alice two words W_0,W_1
  - Receives a challenge PEKS(PK,W_b), where $b \in_R \{0,1\}$
  - Can ask for more T_W values, but not for T_{W_0} or T_{W_1}.
  - Must find b with probability significantly better than ½.

PEKS implies identity based encryption

- Given a PEKS, we can build an IBE system which encrypts single bits
  - The public parameter is the public key PK of the PEKS. The master private key is the trapdoor T.
  - The IBE key for identity X is the pair $d_X = \langle T_{X|0}, T_{X|1} \rangle$
  - To encrypt a bit *b*, using identity X, compute CT=PEKS(PK, X|b).
  - To decrypt CT, output 0 if Test(PK,CT,d_0)="yes", and output 1 if Test(PK,CT,d_1)="yes".

- Therefore, building searchable public key encryption is at least as hard as IBE.

Construction

- Using
  - a bilinear map $e : G_1 \times G_1 \rightarrow G_2$ of groups of prime order p
  - and hash functions $H_1:\{0,1\}^* \rightarrow G_1$, $H_2:G_2 \rightarrow \{0,1\}^{\log p}$.

- The construction:
  - Key generation: The secret key is $\alpha \in_R [1,p]$. The public key is $PK=\langle g,h=g^\alpha \rangle$, where g is a generator of G_1.
  - $PEKS(PK,W) = \langle\ g^r, H_2(e(H_1(W),h^r)\ \rangle$
  - $Trapdoor(\alpha, W) = T_W = (H_1(W))^\alpha$
  - $Test(PK, S, T_W)$: Let $S=\langle A,B \rangle$. Output "yes" iff $H_2(e(T_W,A))=B$.
- Correctness? Security? (why use $H_2()$, which is modeled as a random oracle?)

A simple PEKS from trapdoor permutations

- Source-indistinguishable public key encryption
  - A public key encryption scheme is source-indistinguishable if, given encryptions with public keys PK_1 and PK_2 of a random message m, it is impossible to decide which encryption corresponds to which key.
  - Source indistinguishable public key encryption can be constructed from trapdoor permutations.

A simple PEKS from trapdoor permutations

- Let Σ be the dictionary of all possible keywords.
  - For each $W \in \Sigma$ generate a new public/private key pair PK_W/SK_W
  - PEKS(PK,W): pick a random $m \in \Sigma$, and output $\langle m, Enc(PK_W, m) \rangle$. Namely, encrypt m using the public key PK_W.
  - Trapdoor: The trapdoor W is $T_W = SK_W$.
  - Test(PK,S,T_W): Let $S = \langle A,B \rangle$. Decrypt B with the key T_W. Output "yes" iff the result equals A.
- Security?
- Overhead? Quite high.

Reducing the public key size

- Reducing the size of the public key to be linear in the number of search queries, rather than in the size of the dictionary.