Advanced Topics in Cryptography

Lecture 13: Search on encrypted data and on data streams.

Benny Pinkas

Related papers

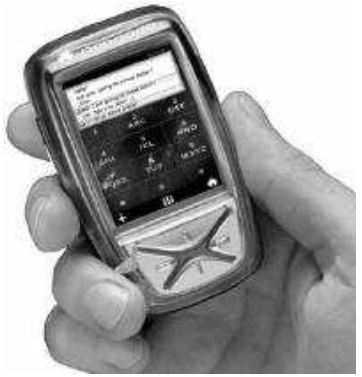
- Secure indexes, Eu-Jin Goh. http://crypto.stanford.edu/~eujin/papers/secureindex/index.html

- Private searching on streaming data, R. Ostrovsky and W. Skeith.

Search on encrypted data

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 (note that we should only provide security for the remote user, not for the server)
  - 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 for index based search
- Index based search requires O(1) search time
- The search will be based on
  - Bloom filters – efficient tests for group membership
  - PRFs

Bloom filters

- An m bit array B represents a set $S=\{S_1,\ldots,S_n\}$
- Uses r independent hash functions
  - $h_1,\ldots,h_r : \{0,1\}^* \rightarrow [1,m]$

- Initially, all bits in the array B are set to 0.
- Insertion: to insert a word x
  - Compute $i_1=h_1(x),\ldots,i_r=h_r(x)$
  - Set $B[i_1]=1, \ldots, B[i_r]=1$ (if B[i] is already set to 1, it is not changed)
- Testing: to test if a word x is in the set
  - Compute $i_1=h_1(x),\ldots,i_r=h_r(x)$
  - Output Yes iff $B[i_1] \wedge \ldots \wedge B[i_r] = 1$.

Bloom filters

- If $x \in S$ then the test always returns “Yes”
- False positives:
  - If x is not in S then the test might return “Yes” if all B[i] corresponding to x were set by other words in S.
  - The false positives rate depends on the relation between r,n and m.

Bloom filters - parameters

- Bounding the probability of false positives
  - Inserting n documents to an m bit Filter, using r functions.
  - The probability that a bit remains 0 is $(1-1/m)^{rn} \approx e^{-rn/m}$
  - The probability of a false positive is therefore $(1-e^{-rn/m})^r$
  - Assume that n and m are fixed. The above probability is minimized with $r=(m/n)\cdot \ln 2$, and is 2^{-r}. Therefore r=log(1/false error probability).

- Bloom filters have many other nice properties
  - Easy to add items
  - Hard to remove items
  - Easy to merge Bloom filters

Keyword search using Bloom filters

- First (insecure) approach
  - The user uses a PRF F, with different keys k1,…,kr which it keeps secret.
  - For every word w, define WD(w) = $\langle F_{k1}(w),\ldots,F_{kr}(w)\rangle$
  - For a document D containing words $w_1,..,w_L$, set to 1 the bits in B corresponding to the indices in $WD(w_1),\ldots,WD(w_L)$.
  - Send the documents and the array B to the server.
  - Search:
    - Given a search word x, compute and send WD(x)= $\langle F_{k1}(x),\ldots,F_{kr}(x)\rangle$ to the server.
    - The server checks each document, to see if in the corresponding Bloom filter B all these bits are set to 1.

Problems with the previous solution

- A word x results in the same set of indices WD(x) for each document in which it appears.
- The server cannot know which words appear in each document, but it can check for similarity between documents.

The construction

- Initialization: choose a master key $\langle k1,\ldots,kr \rangle$
- Trapdoor: For a word w, search is enabled using the trapdoor $T_w = \langle F_{k1}(w),\ldots,F_{kr}(w) \rangle$
- Build index: given a document D with identifier D_{ID} and words $w_1,\ldots,w_t$, do for each w_i
  - Compute $x_1=F_{k1}(w_i)$,…, $x_r=F_{kr}(w_i)$
  - Compute indices $y^D_1=F_{x1}(D_{ID})$,…, $y^D_r=F_{xr}(D_{ID})$
  - Set the bits $y^D_1,\ldots,y^D_r$ in the Bloom filter B to 1.
  - Let u be an upper bounds on the number of words in a document, and v be the number of unique words in D. Choose with repetition (u-v)r random bits in B, and set to 1.
- Search for w:
  - Provide T_w to the server. For every D, server uses T_w to compute $y^D_1,\ldots,y^D_r$ and check if they are all 1.

Overhead

- r is set to be log(1 / false positive probability)
  - r=10 results in error probability of 1/1024
- $r=(m/n)\cdot \ln 2$
  - Therefore, for a document with 2000 unique words, need to set m to about 28000 bits, or 3.6 kilobytes.
- The PRF can be implemented using very efficiently (say, using HMAC-SHA1).

Properties

- Easy addition of words to documents
- Short trapdoor (k_1,…,k_r)
- The document itself can be compressed and then encrypted (unlike the search methods we learned last week)
- Variable length words

Private searching on streaming data

- Motivation:
  - The intelligence community monitors data (e.g., email messages) using important keywords.
  - The keywords themselves are confidential
  - They therefore need to gather all the data, and do the filtering in house.
  - It is more efficient to let the ISPs do the filtering, but this would reveal the keywords to the ISPs.
  - Drawbacks of the current practice:
    - Communication, processing.
    - Privacy.

Requirements

- The search algorithm is run by an untrusted party
  - The search algorithm must be kept secret
  - The results must be kept secret
  - For efficiency, the size of the query must be independent of the size of the stream, but it might depend on the size of the dictionary of possible search words.

Scenario

- The client generates a public key and a private key.
- The client sends a “search program” to a server.
- The server applies the program to a data stream.
- The output of the program is an encrypted buffer which is sent to the client.
- The client uses the private key to learn the output.

- Note that the size of the encrypted output buffer must not reveal the number of matches.

Basic scheme based on Paillier's homomorphic encryption

- Key generation: generate keys for Paillier's scheme.
- Filter generation:
  - Assume that the size of each document can fit as a plaintext for the encryption scheme (no problem, see below).
  - The filter consists of
    - A buffer with 2cm blocks, each with two encryptions, $\langle E(1),E(1)\rangle$.
    - For each word w in the (English) dictionary, $e_w=E(1)$ if w is a searchword, and $e_w=E(0)$ otherwise.
  - Search:
    - Let $d_1,..,d_L$ be the words in document D.
    - Compute $v = \Pi_{i=1...L}\ e_{di}$. (The result is E(j), where j is the number of occurrences of searchwords in D.)
    - Compute E(vD), and multiply E(v),E(vD) into c random locations in the buffer.

Basic scheme

- Decryption:
  - Decrypt B one block at a time.
    - If the block is of the form (0,0) then disregard it
    - If it is of the form (s,sD), then retrieve document D.

- Problems:
  - The document should fit as a plaintext for Paillier's scheme
    - Luckily, the Damgard-Jurick variant of Paillier uses a ciphertext of size n^{d+1} to encrypt plaintexts of size n^d.
  - Collisions:
    - What if two documents are mapped to the same block, which is now of the form (s+s',sD+s'D')?