Highly-Scalable Searchable Symmetric Encryption

Presenter: Jacob Chen

Searchable symmetric encryption

- Queries are only limited to a single keyword
- In order to be practical, a system should support conjunctive search
- Symmetric encryption should be scalable
- All systems leak information
- A practical system must relax absolute privacy

Searchable symmetric encryption (cont'd)

- Complexity should ideally be proportional to the number of matches of the least frequent term
- FHE and ORAM are too costly

Contributions

- Developed the first non-generic sub-linear SSE schemes supporting conjunctive keyword search
- Assumes the adaptive adversary model
- Server cannot deduce anything beyond the precisely defined *leakage profile*
  - leakage profiles includes total size of database
  - Access patterns (intersection of two sets of results and queries)
  - There is never direct exposure of plaintext data or searched values

Contributions (cont'd)

- Complexity of the search protocols is independent of the number of documents in the database
- Extends to support answers for Boolean queries

SSE Syntax

- A database has *d* documents
- Each document has a set of keywords W_i
- Output from the SSE protocol are indices, ind (identifiers)
- Client program uses ind to retrieve encrypted documents

SSE Syntax (cont'd)

- A SSE Scheme π consists of EDBSetup and a Search Protocol
- EDBSetup takes input DB and outputs secret key K along with EDB
- K is given to the client
- EDB is given to server

T-Sets

- A T-set is a tuple set
- T-set associates a list of fixed-size data tuples with each keyword in the database
- Σ = (TSetSetup, TSetGetTag, TSetRetrieve)
- (TSet, K_T) = TSetSetup(T)
- s-tag = TSetGetTag(K_T, w)
- TSetRetrieve(TSet, s-tag) returns T[w]

T-Sets (cont'd)

Index	Strings
w_1	ind_1, ind_2, ind_3, …
w_2	ind_4, ind_5,ind_6, …
w_3	ind_7, ind_8, ind_9, …

Single-Keyword SSE Scheme

PRF → K_S

For each w in the database, find its corresponding document ind

Database → $<w_i, ind_1, ind_2, ...>$

Initialize **T** to be an empty array

For each w in W, build **T[w]** as follows:

Initialize **t** as an empty list

K_S, w → F → K_e

For all ind in DB(w), in random order

Enc(K_e, ind) → e

t = [**t**, e]

T[w] = t

T → TSetSetup → Tset, K_T

Give KS, KT to client, and give server TSet

Client takes as input key (K_S, K_T) and keyword w to query

KT, w → TSetGetTag → s-tag

Server

Server has TSet and s-tag

TSetRetrieve → t

Client

Client Side:

K_S, w → F → K_e

For each e in **t**

Dec → ind

SSE Schemes

- The server's work grows linearly with the number of documents
- Scheme does not support unstructured data
- We want to develop a search protocol that scales with the size of the smallest $DB(w_i)$ among the conjunctive terms w_i
- Naive solution is to run a single keyword SSE scheme for each conjunctive word

Basic Cross Tag protocol

PRF → K_S, K_X

For each w in the database, find its corresponding document ind

Database → <w_i, ind_1, ind_2, ...>

Initialize **T** to be an empty array,
Initialize **Xset** to an empty set

For each w in W, build **T[w]** and **Xset** as follows:

Initialize **t** as an empty list

K_S, w → F → K_e

K_X, w → F → XTrap

For all ind in DB(w), in random order

Enc(K_e, ind) → e

t = [**t**, e]

Xtrap, ind → f → XTag

Xset = [**Xset**, XTag]

T[w] = t

T → TSetSetup → Tset, K_T

Give K_S, K_X, K_T to client, and give server Tset, Xset

Client takes as input (K_S, K_T, K_X) and keywords w_1, w_2, … w_n to query

K_T, w_1 → TSetGetTag → s-tag

K_S, w_1 → F → K_e

For each i = 2, 3, … , n

K_X, w_1 → F → Xtrap $_i$

Server

K_e Tset s-tag → TSetRetrieve → t

For each ciphertext e in t, server computes:

Dec → ind

Xset ← in ← f ← Xtrap $_i$

Client

Basic Cross Tag protocol cont'd

- The server's work in BXT scales with $n * |DB(w_1)|$
- This protocol improves substantially security wise in terms of leakage
- Vulnerable to a simple attack: server can use xtrap and save it for later use
- Performance and privacy improves with lighter s-terms

Oblivious Cross-tags Protocol

PRF → K_S PRFp → K_I, K_X, K_Z

For each w in the database, find its corresponding document ind

Initialize **T** to be an empty array,
Initialize **Xset** to an empty set

For each w in W, build **T[w]** and **Xset** as follows:

Initialize **t** as an empty list

K_S, w → F → K_e

K_X, w → F → XTrap

For all ind in DB(w), in random order, init a counter c = 0

K_I, ind → PRFp → xind K_Z, w||c → PRFp → Z

xind* z → Y^-1 Enc(K_e, ind) → e

t = [**t**, (e,y)]

Xind, K_X, w → g → XTag

Xset = [**Xset**, XTag]

T[w] = t

T → TSetSetup → Tset, K_T

Give K_S, K_X, K_T K_I, K_Z to client, and give server Tset, Xset

Client takes as input (K_S, K_T, K_X , K_I, K_Z) and keywords w_1, w_2, … w_n to query

K_T, w_1 → TSetGetTag → s-tag

For c = 1, 2, … until server says stop

For i = 2,..., n

K_Z, w_1, c, K_X, w_i → G' → xtoken[c,i]

xtoken[c]

Server

TSet S-tag Xset xtoken[c]

TSetRetrieve → **t**

For c = 1, …, |t|

Retrieve (e,y) from the c-th tuple of **t**

For i = 2, …, n

If xtoken[c,i]^y belongs to Xset

e → client

When last tuple is reached, send STOP

Client

Retrieves e from server

K_S, w_1 → F → Ke

ind → DEC

Oblivious Cross-tags Protocol

- Single round of communication
- Achieves better security