Encrypted Domain Keywords Search

Ee-Chien Chang

School of Computing

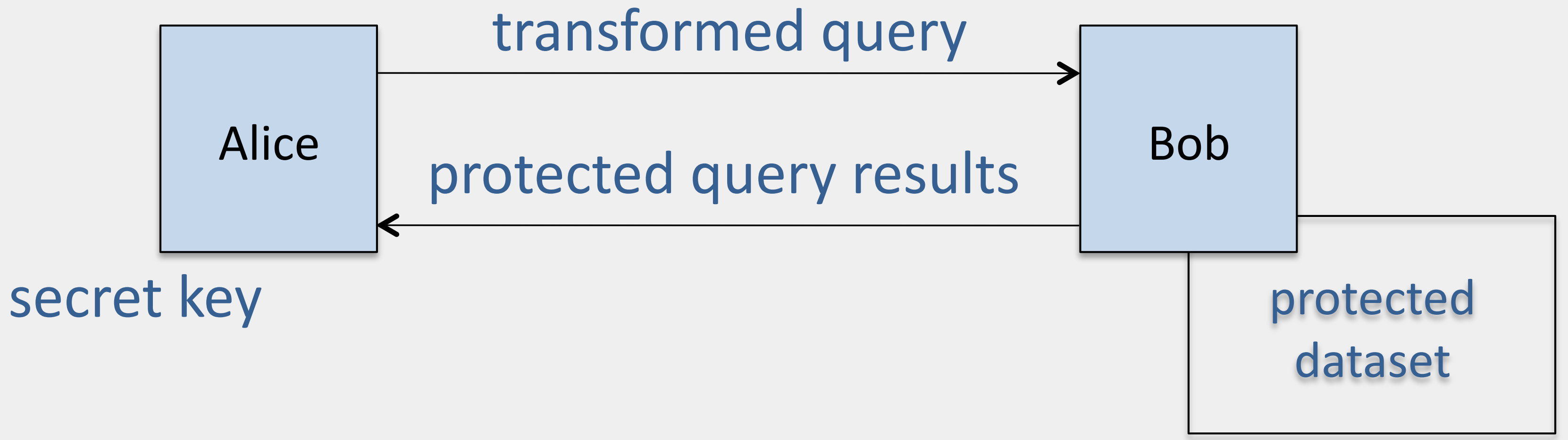
National University of Singapore

Encrypted Domain keywords Search

1. Motivation & Overview

Problem

- **Alice** (user) has a set of documents $\boldsymbol{D}=\{d_1, d_2, \ldots, d_n\}$ that are to be stored by **Bob** (server). Periodically, Alice wants to retrieve all documents that contains a query word w.
- Alice does not fully trust Bob. Ideally, she wants to hide the query, query results and the dataset from Bob.
- The query, query-result, and the dataset $\boldsymbol{D}$, have to be protected by some mechanisms.



Performance Requirements

- Alice wants a solution that requires low communication and computation cost per query, and low storage overhead.

- The communication requirement rules out the following meaningless solution:
  - the whole dataset is encrypted using a well-accepted cipher (for e.g. AES) and stored in Bob;
  - for each query, Bob sends the the encrypted dataset back to Alice. Alice decrypts the dataset and then searches for the results.

Security Requirements

- It is important to clearly specify the ***security requirements*** (also called as ***security model*** or ***adversary model***)
- The security requirements are typically formulated by describing the types of attacks to be prevented.
- Types of attacks are typically described by specifying
  - the attackers' goals,

    e.g. to find the secret key,
  - information accessible by the attackers, and

    e.g. a randomly selected plaintext and the corresponding ciphertext
  - attackers' computing resources and capabilities

    e.g. polynomial time algorithm

Our Security Requirements

- Storage privacy:
  - Server is unable to get information on the dataset ***D***.

    (The above is an over-simplified description. Is it ok to reveal the size of each document in the dataset? Is it ok to reveal the number of documents in the dataset? Strong requirements usually take the form of "***semantic security***". In the context of cryptosystem, a semantically secure scheme ensures that any adversary is unable to distinguish the ciphertexts of two known plaintexts.)

- Query privacy
  - Server is unable to get information of the query.

    (Again, the above is an over-simplified description. Is it ok to reveal which part of the dataset a query has accessed?)

Server’s capability

- Curious-but-honest server: We consider ***curious-but-honest*** server. That is, the server will carry out the protocol and computation honestly, but may retain knowledge derived during the process for malicious purpose. (*Question: is this assumption reasonable?*)

- Malicious server: We will not consider malicious server (i.e. dishonest server in contrast to the “curious-but-honest” model) in this lecture. In general, it is theoretically possible to have both secure and efficient (communication and storage overhead) mechanisms using the “*fully homomorphic encryption*”, but the computation cost of such generic mechanisms is extremely high (Two generic constructions [1,3]).

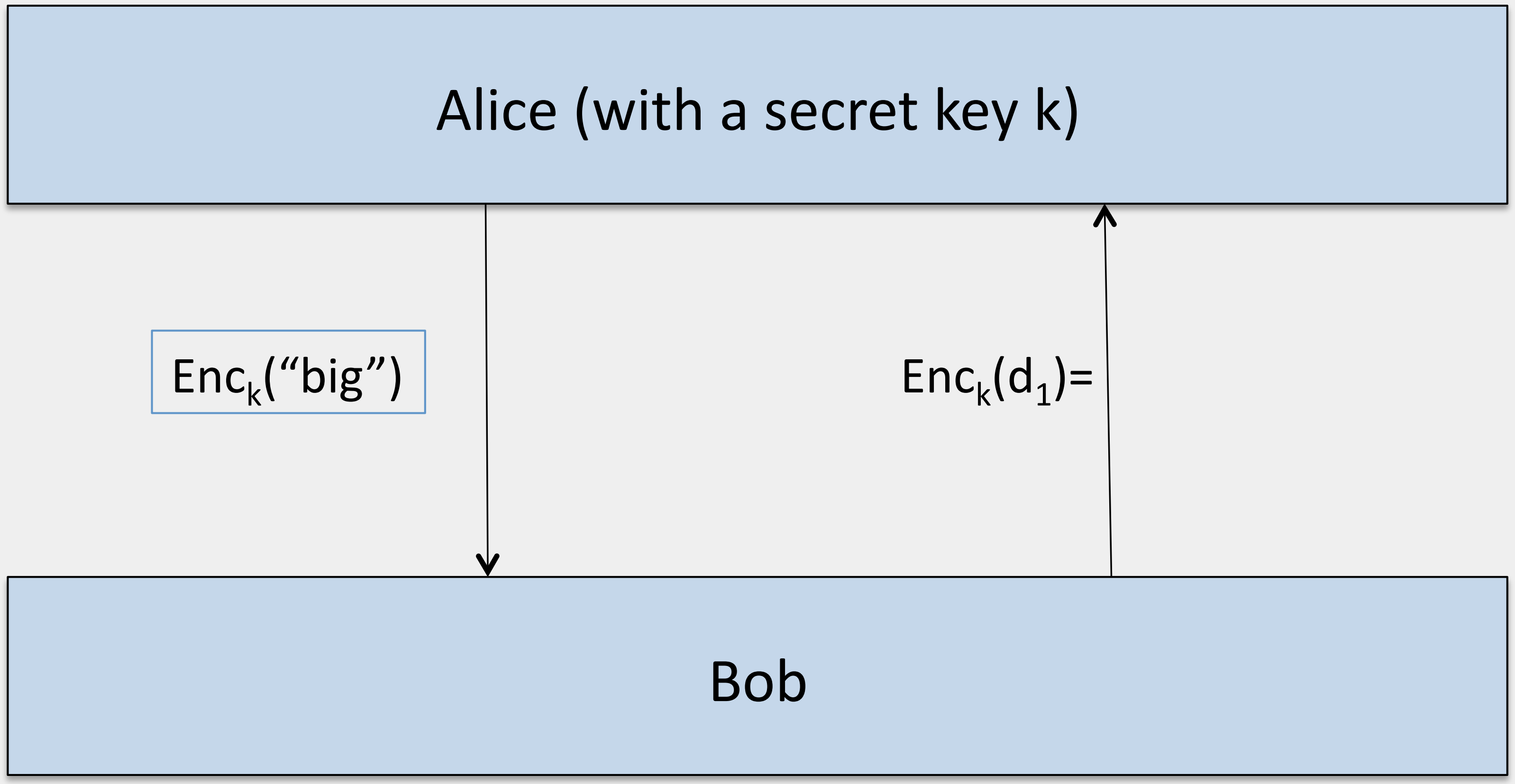
2. A simple illustrating scheme that leaks too much

- Each document is represented as a set of words, $d = \{ w_1, w_2, \ldots, w_m \}$

  For e.g. *d* ={ “cryptography”, “make”, “the”, “world”, “safer” }

  Each word is padded to 256 bits by adding spaces, and then encrypted using a well-accepted deterministic cipher, say AES, using a secret key ***k***.

  To search for a word *w*, Alice sends the ciphertext $\mathrm{Enc}_k(w)$ to Bob. Bob then returns all documents that contain $\mathrm{Enc}_k(w)$.



$\mathrm{Enc}_k(d_1)=\{\ \boxed{\mathrm{Enc}_k(\text{"big"})}\ \ \boxed{\mathrm{Enc}_k(\text{"cat"})}\ \ \boxed{\mathrm{Enc}_k(\text{"runs"})}\ \}$

$\mathrm{Enc}_k(d_2)=\{\ \boxed{\mathrm{Enc}_k(\text{"cat"})}\ \ \boxed{\mathrm{Enc}_k(\text{"dog"})}\ \ \boxed{\mathrm{Enc}_k(\text{"lion"})}\ \}$

What's wrong?

- storage privacy?
  - For the example in the previous slide, Bob knows that d_1 and d_2 have exactly one common word.
  - Use statistical test, Bob may able to infer the ciphertext of common words like "the", "is", etc
- query privacy?
  - Suppose Alice has issued two queries. Bob will know whether these two queries are the same.

What’s wrong?

- The method is still slow. The whole dataset has to be scanned for each query.

4. Secure index with Bloom Filter (E.J. Goh [4])

Incorporating ***Bloom filter*** and a technique by Song et al[7].

Background – Bloom filter

A Bloom filter is a representation for membership testing. It represents a set (*document*) $\boldsymbol{d} = \{w_1, \ldots, w_m\}$ of m items (*words*) using an array of t bits.

The filter uses $\boldsymbol{r}$ hash functions $H_1, H_2, \ldots, H_{\boldsymbol{r}}$ to determine the bit-values in the array. Each hash function maps an item to an integer in $\{1, 2, \ldots, \boldsymbol{t}\}$.

For the set ***d***, the ***j**-th* bit of the representing array is 1 *iff* there is an item ***w*** in ***d***, and an ***i*** such that

$$H_{\boldsymbol{i}}(\boldsymbol{w}) = \boldsymbol{j}\ .$$

wiki gives a good description of Bloom Filter http://en.wikipedia.org/wiki/Bloom_filter

Example

Suppose t=10, r=2.

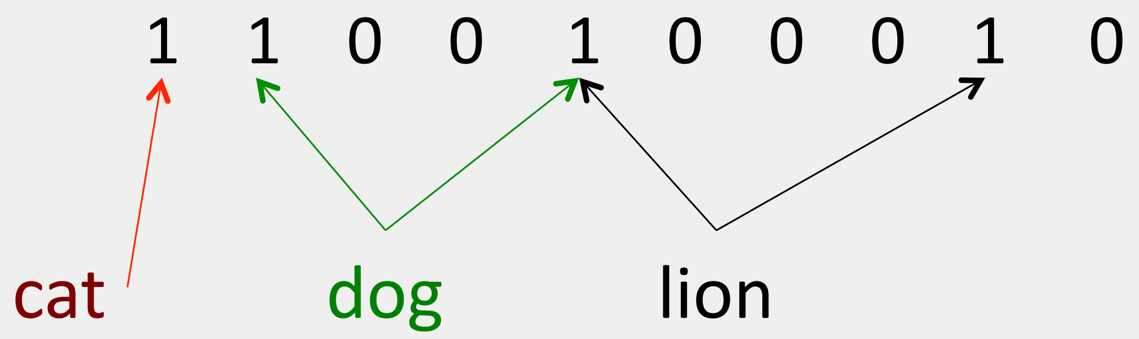
Consider d= { “dog”, “cat”, “lion”}.

If

H_1(“dog”) = 5 H_1(“cat”) = 1 H_1(“lion”) = 5

H_2(“dog”) = 2 H_2(“cat”) = 1 H_2(“lion”) = 9

Then, the array is



Searching using Bloom Filter

Bloom filter is designed to speedup membership test.

Given an query (*word*) ***w*** and a set ***d*** (*document*), determine whether ***w*** is in ***d.***

This can be done by first computing the hash of w

$$h_1 = H_1(\boldsymbol{w}),\ h_2 = H_2(\boldsymbol{w}),\ \ldots,\ h_r = H_r(\boldsymbol{w}),$$

and then check whether all the $h_1, h_2, \ldots, h_r$ –th bits in the array are 1. If so, declare that ***w*** is in ***d***.

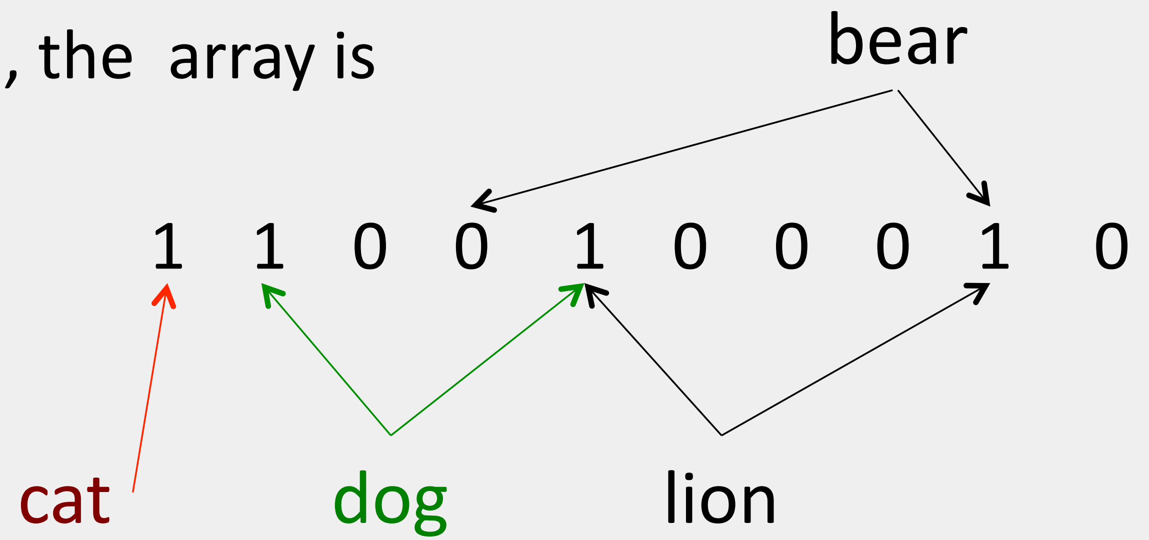
Example

Consider two words “bear” and “panda”

H_1(“panda”) = 2 H_1(“bear”) = 4

H_2(“panda”) = 9 H_2(“bear”) = 9

Then, the array is



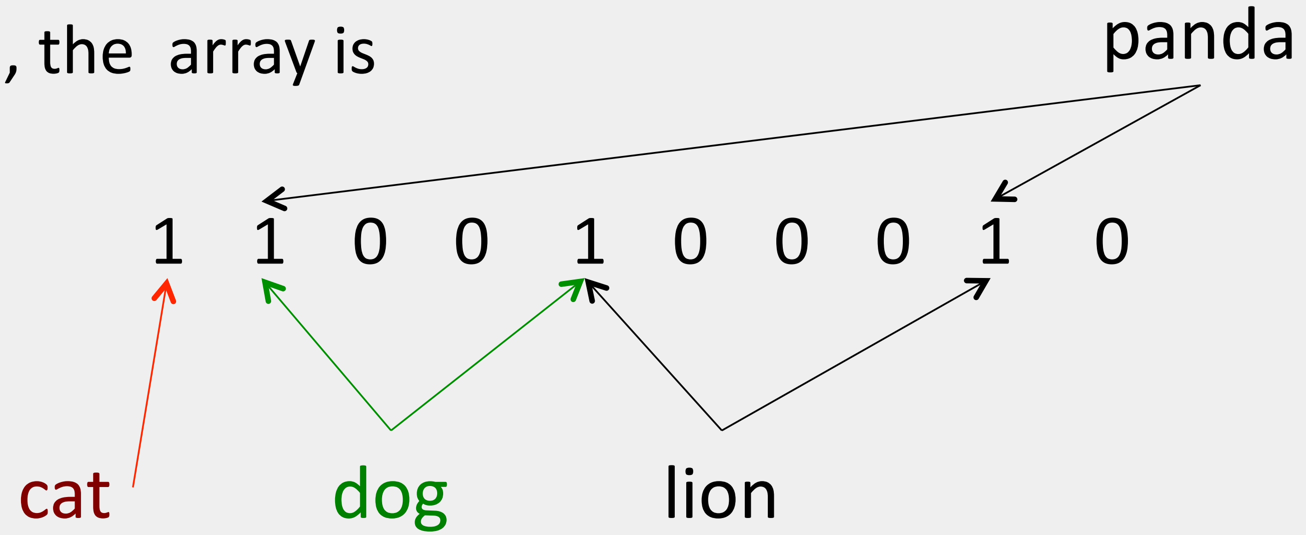
Example

Consider two words "bear" and "panda"

H_1("panda") = 2 H_1("bear") = 4

H_2("panda") = 9 H_2("bear") = 9

Then, the array is



Performance of Bloom Filter

- The membership checking can be done in O(***r***) time.

  The storage space required is ***t*** bits.

- The false accept rate is non-zero, i.e. there are chances that ***w*** is not in ***d***, but wrongly declared to be in.

  The parameters, ***r*** *(i.e. number of hash funcitons)*, ***t*** *(i.e. size of array)* can be adjusted to tradeoff false accept rate, the searching time and storage size, for a given ***m*** *(i.e. number of items)*.

  See http://en.wikipedia.org/wiki/Bloom_filter for details of the tradeoff.

Bloom filter for a collection of sets

Recall that our original problem is for a collection of documents:

> Given an query (*word*) ***q*** and a collection of sets (*documents*), find all the sets (*documents*) ***d*** that contains ***q***.

Bloom filter can be extended to a collection of $\overset{n}{\cancel{m}}$ sets, represented as an ***n*** by ***t*** bits array, one set per row.

d_1 : 0 0 1 1 0 … 0 1

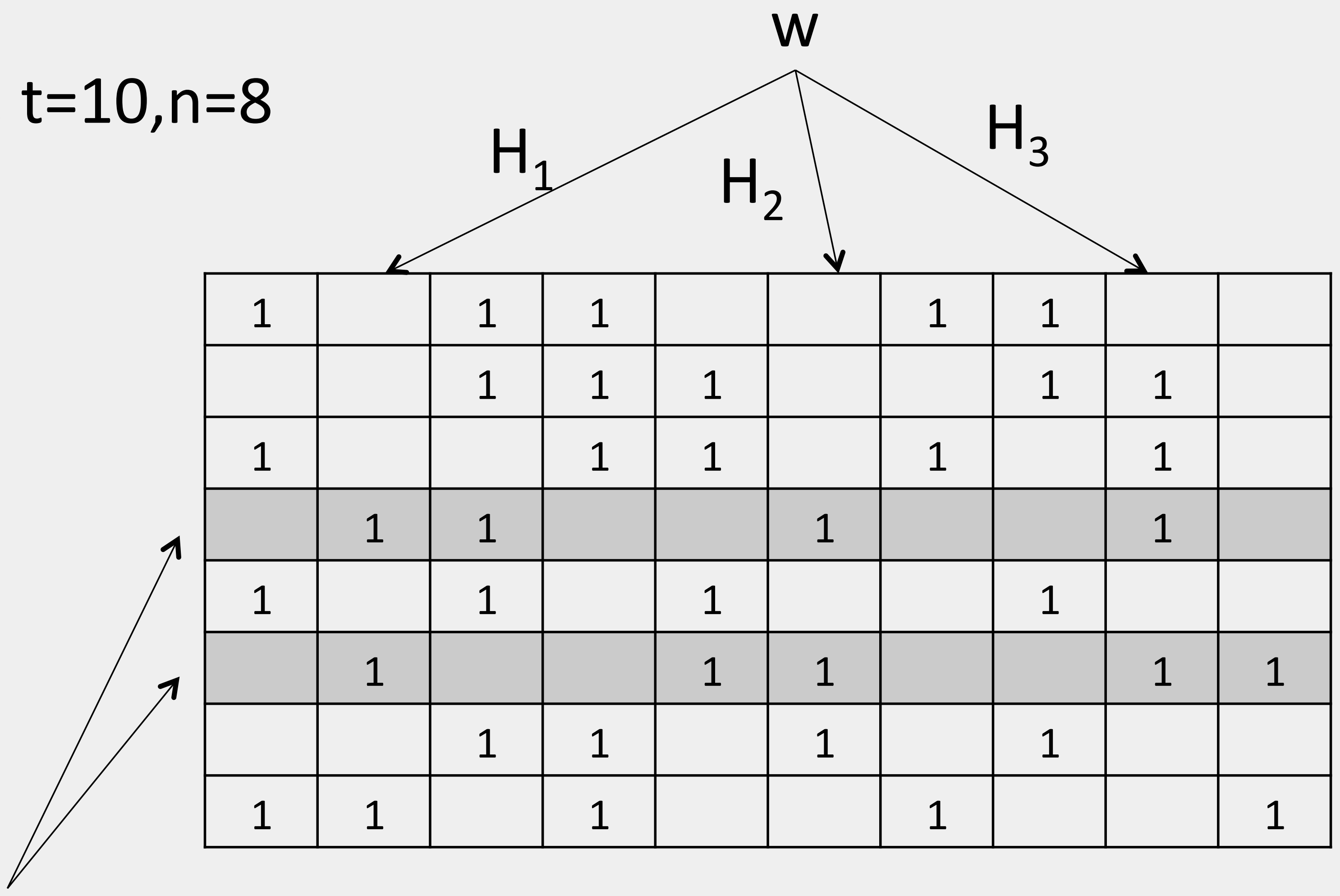
d_2 : 0 1 0 1 0 … 1 0

…

d_n: 0 1 0 1 1 … 0 0

Each search is done by checking membership row by row.

r=3, t=10,n=8



1		1	1			1	1		
		1	1	1			1	1	
1			1	1		1		1	
	1	1			1			1	
1		1		1			1		
	1			1	1			1	1
		1	1		1		1		
1	1		1			1			1

each of these 2 documents likely contain w.

Secure index (Goh [ ])

The scheme consists of two components:

- **Setup phase**: Given a secret key ***k***, and the set of documents, Alice builds a Bloom Filter using "hidden" hash functions. This structure is then sent to Bob, and removed from Alice's storage. Alice keeps the key.

- **Query phase**: Given a query word ***w*** and the secret key ***k***, Alice computes and sends some "*trapdoor*" keys to Bob. With the trapdoor keys, Bob can compute the "hidden" hash functions and thus can conduct the BF search.

Setup phase – input and parameters

Given a collection of n documents

$$\boldsymbol{D} = \{ d_1, d_2, \ldots, d_n \}$$

where each document is a set of *m* words, and each document is associated with an unique identity *id.* (note that for simplicity, we assume that every document has exactly *m* unique words).

- Alice determines the desired Bloom Filter parameters r, t, and chooses a secure *pseudo-random* function (for e.g. base on a well accepted cryptographic hash like SHA2). The parameters and choices of functions are made public, i.e. they are not secret.
- Alice generates a secret key k=(k_1, k_2, …, k_r), which consists of r subkeys.

Setup phase- building the BF

Given a word w and the subkeys, let's call

$$f(w, k_1), \quad f(w, k_2), \ldots, \quad f(w, k_r)$$

the ***trapdoor keys*** for w.

- For a document ***d*** with the identity ***id,*** Alice constructs the Bloom Filter by using the following *r* hash functions:

$$H_1(w) = f(\,id,\ x_1\,) \quad \text{where } x_1 = f(w, k_1)$$
$$H_2(w) = f(\,id,\ x_2\,) \quad \text{where } x_2 = f(w, k_2)$$
$$\ldots$$
$$H_r(w) = f(\,id,\ x_r\,) \text{where } x_r = f(w, k_r)$$

- The constructed BF (which is an *n* by *t* bits array) are then sent to Bob. The identities of of the documents are also sent to Bob.

- Remark: The *j*-th hash function for the document *id* is

$$H_j(w) = f(\, id, \underset{\overset{\uparrow}{f(\text{ document id, trapdoor key })}}{\,} x_j \,) \quad \text{where } x_j = f(w, k_j)$$

The hash functions are different for different documents.

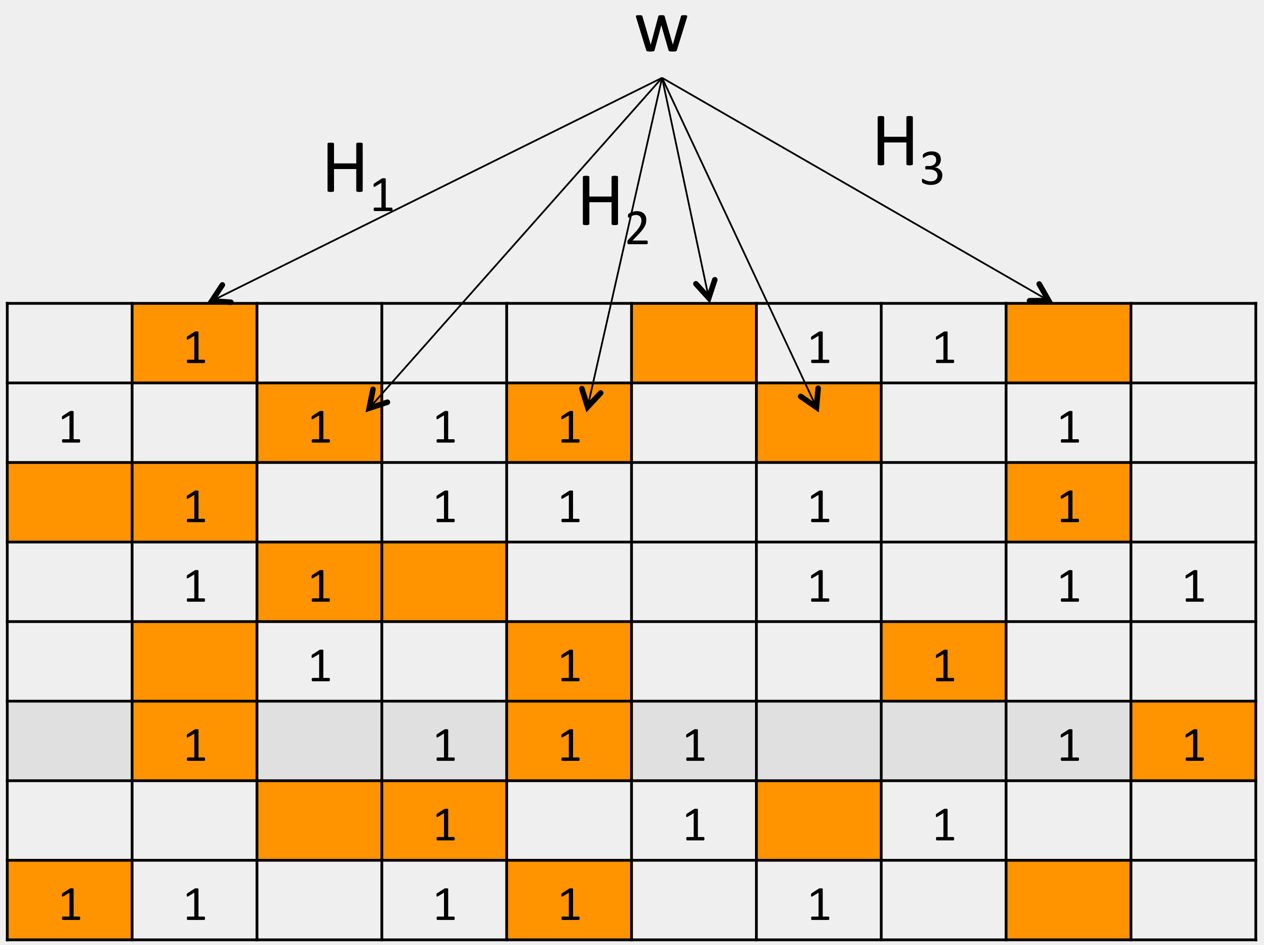
Query phase

- Given a query word w. Alice computes the trapdoor keys and sends them to Bob.

$$x_j = f\,(\,w, k_j\,) \quad \text{for } j = 1,2,\ldots, r.$$

- From the trapdoor keys, Bob can construct the hash functions for each document, and thus can perform the search.

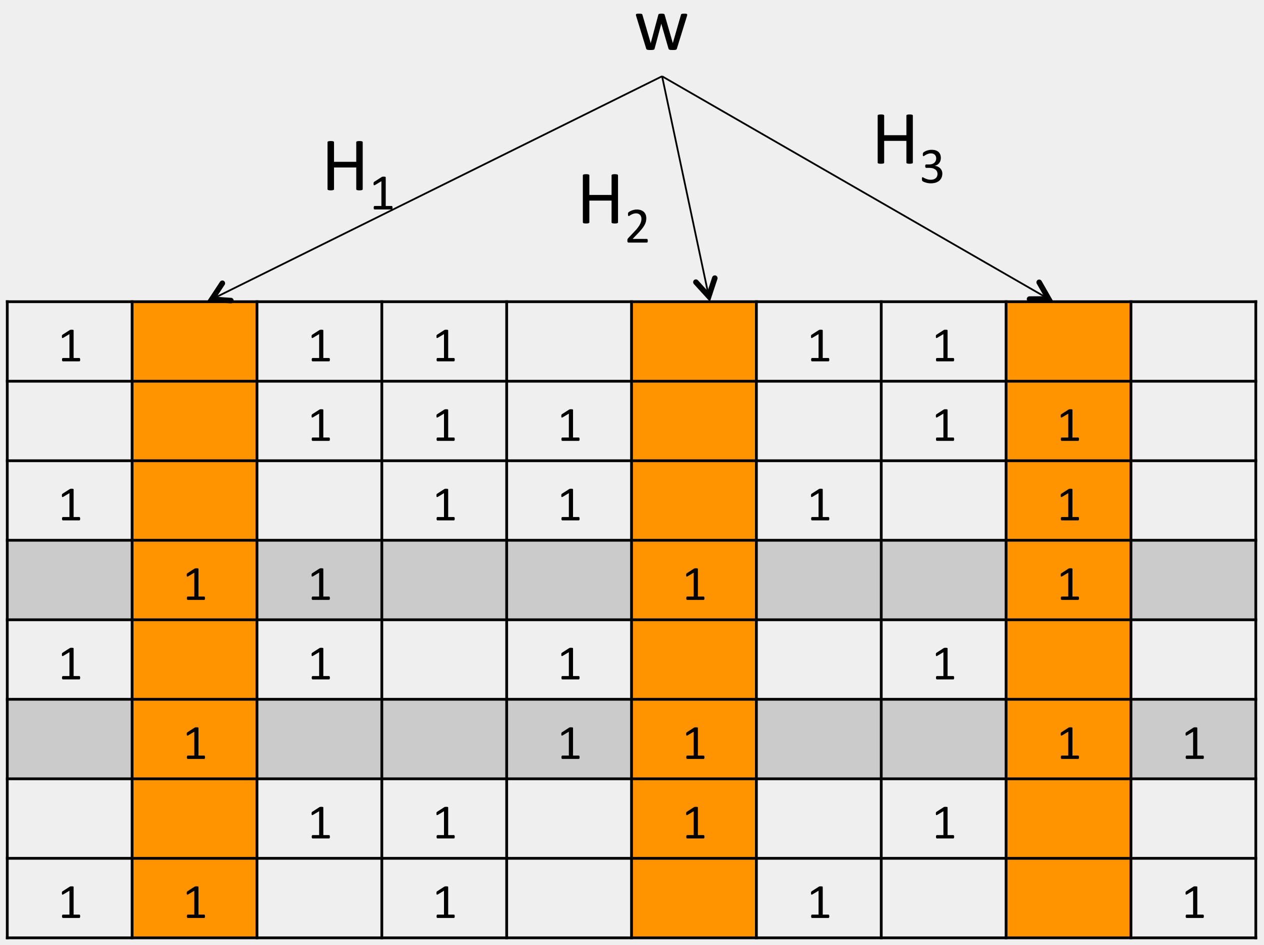
BF for secure index



	🟧 1				🟧	1	1	🟧	
1		🟧 1	1	🟧 1		🟧		1	
🟧	🟧 1		1	1		1		🟧 1	
	1	🟧 1	🟧			1		1	1
	🟧	1		🟧 1			🟧 1		
	🟧 1		1	🟧 1	1			1	🟧 1
		🟧	🟧 1		1	🟧	1		
🟧 1	1		1	🟧 1		1		🟧	

each document uses a different set of hash functions!

Original BF



1		1	1			1	1		
		1	1	1			1	1	
1			1	1		1		1	
	1	1			1			1	
1		1		1			1		
	1			1	1			1	1
		1	1		1		1		
1	1		1			1			1

Performance

- False accept rate is the same as the original BF.
- For each document, Bob has to compute the hash *r* times. Hence, computation load is higher compare to the original BF.

Security: Storage Privacy

- A security model is given in [4].

  "*Semantic Security Against Adaptive Chosen Keyword Attack*"

  It is formulated as a game between an attacker and the system. The scheme can be proven secured against such attacker, assuming that the pseudo-random function *f* ( ) is secure.

  As a consequence, suppose an attacker knows that the dataset is equally likely to be either a particular D_1 or D_2 . Even if the attacker has seen the BF, he is unable to correctly guess the database with probability more than ½ + a small noticeable probability.

Security: Query privacy

- Very weak protection. After Bob has seen a query *w* (i.e. the trapdoor keys of ***w***), he knows which documents contain ***w***.

- Note that from the trapdoor keys, it is still computational hard to find ***w***. However, if Bob knows the relationship of the documents and the queries, he may able to infer some useful information of ***w***, for e.g. whether it is a common word.

Remark

- In the technical report[4], page 7. An additional step-3 is included. This step adds random words to the table.

  What is the consequence if this step is omitted?

  Why step-3 is not required in our construction? (we already assume that each document has exactly n unique words)

Remark

- If the query asks for documents that contain both w_1 and w_2, this can be carried out using one search instead of 2.

- However, if the query asks for documents that contain either w_1 or w_2, this still has to be carried out in two searches.

Other methods and Variants.

Other methods: Inverted linked-lists

- Curtomal et al [1] proposed a method that hides the inverted linked-lists.

$$\begin{tikzcd}
\boxed{w_1} \arrow[r] & \boxed{d_1} \arrow[r] & \boxed{d_{15}} \arrow[r] & \boxed{d_7} & \\
\boxed{w_2} \arrow[r] & \boxed{d_3} \arrow[r] & \boxed{d_2} \arrow[r] & \boxed{d_{10}} \arrow[r] & \boxed{d_9} \\
 & \ldots & & & \\
\boxed{w_m} \arrow[r] & \boxed{d_{23}} \arrow[r] & \boxed{d_2} & &
\end{tikzcd}$$

The linked-lists are "encrypted" and "stored" in a certain way s.t without the "trapdoor key" to the header, the server is unable to determine the structure.

- Similar to Goh’s method, it can provide storage privacy but not query privacy.

Other methods: PIR

- Private Information Retrieval (PIR) are protocols designed to achieve query privacy. In such schemes, storage privacy is not taken into consideration and many known schemes actually store the dataset in clear.

- Known PIR schemes use some form of homomorphic properties in the encrypted domain, and thus are computational expensive.

- There are PIR schemes that achieve both query and storage privacy, for e.g. [8].

Other methods: Oblivious RAM

- Oblivious RAM [6] are schemes that allow writing and reading items to/from the storage server without revealing which items being read/written. Known schemes are "asymptotically" efficient but incur large hidden constant factors. (i.e. theoretically efficient but practically infeasible).

  (The requirement is similar to PIR. However, unlike PIR, in oblivious RAM, the storage can be modified after each access.)

  By combining oblivious RAM with either Goh's or Curtmola et al.'s method query privacy can be achieved.

Other methods: Information theoretic

Another approach uses “bucketization” to achieve privacy, for e.g [5]. During the process, some information of the dataset is discarded. These schemes are information theoretic secure, i.e. even if the adversaries have sufficiently long computing time and large memory, they are unable to infer useful information (note that information is discarded ☺) However, it is difficult to provide strong security guarantee (i.e. the security models are weak by considering a weak adversary).

5. Conclusion

No secure and practical solution yet.

References

[1] K.M. Chung, Y. Kalai, and S.P. Vadhan. *Improved Delegation of Computation Using Fully Homomorphic Encryption.* CRYPTO 2010.

[2] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. *Searchable symmetric encryption: improved definitions and efficient constructions.* Conference on Computer and Communications Security, 2006.

[3] R. Gennaro, C. Gentry, and B. Parno. *Non-interactive Veriable Computing: Outsourcing Computation to Untrusted Workers*. CRYPTO 2010.

[4] E.-J. Goh. *Secure indexes*. Cryptology ePrint Archieve Report 2003/216, 2003.

[5] B. Hore, E.-C. Chang, M. Diallo and S. Mehrotra. *Indexing Encrypted Documents for Supporting Efficient Keyword Search,* Secure Data Management 2012.

[6] B. Pinkas and T. Reinman, *Oblivious ram revisited*. CRYPTO 2010.

[7] D. Song, D. Wagner, and A. Perrig. *Practical techniques for searches on encrypted data*. IEEE Symposium on Security and Privacy, 2000.

[8] J. Trostle, A. Parrish. *Efficient Computationally Private Information Retrieval from Anonymity or Trapdoor Groups*. ISC 2010.