ICT Risk Assessment

Fabrizio Baiardi
f.baiardi@unipi.it

Syllabus

-
- Security
  - New Threat Model
  - New Attacks
  - Countermeasures

Cloud provider

Encryption

Working with encrypted data

- A client stores its data on a cloud system
- The client wants to implement some computations on the data without leaking any information about
  - the data
  - the data and which data is used by the computation
- Examples
  - Store your personal information on the cloud and compute your tax declaration
  - Store some information on the cloud and search this information
- Requires some proper encryption scheme because only a few schemes satisfies the constrains

Homorphic encryption = Holy gray of encryption

Let

- *R* and *S* be sets
- E an encryption function *R* →*S*

E is

- **Additively homomorphic if** E(a+b)=PLUS(E(a), E(b))
- **Multiplicatively homomorphic if** E(a×b)=MULT(E(a), E(b))
- **Mixed-multiplicatively homomorphic** *E*(*xy*)=Mixed-mult(*E*(*x*),E(*y*))

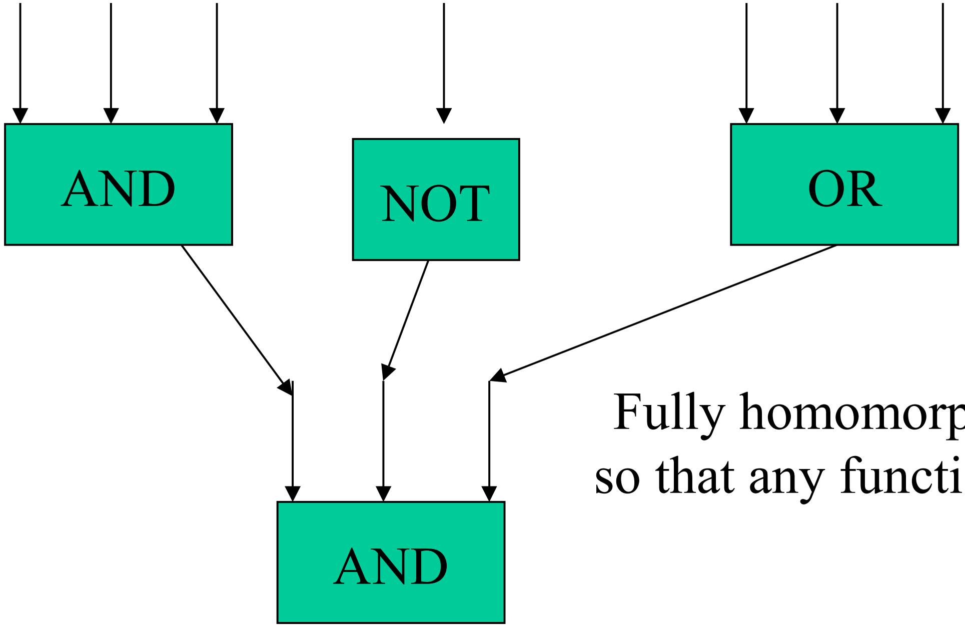
E is fully homomorphic if there are no limitations on the manipulations can be performed.

Homomorphic encryption

- Data is stored at the provider
- Computation is implemented at the provider
- Inputs are encrypted by the client
- The output is transmitted to the client that decrypts it
- No trivial solution is accepted = almost all the computation has to be executed by the provider to prevent cases where
  - the data is transmitted to the client,
  - the client decrypts the data
  - the client computes the results
  - the results are encrypted
  - the results are transmitted to the provider

Fully homomorphic

In the following the manipulation will be represented as a circuit that implements some boolean operations on the data of interest and where the operators are gates



Fully homomorphic = NAND gates so that any function can be computed

Meaning

- Any computation can be expressed as a Boolean circuit: a series of additions and multiplications
- Using such a scheme, any circuit (consisting of AND and XOR) could be homomorphically evaluated, effectively allowing the construction of programs which may be run on the encryptions of their inputs to produce an encryption of their output
- Since such a program never decrypts its inputs, it could be run by an untrusted party without revealing its inputs and internal state.

But our case introduce further constrains-1

- No optimization of the computation is possible
  - Circuit minimization may not be applied because it leaks information about data that is accessed
  - A random access machine cannot be used because it leaks the information of which data has been accessed by the computation (use an Oblivious RAM is possible)
  - This efficiency can be recovered only if information about data that has been used can leak
- The size of the output must be fixed in advance = the number of output wires in the circuit must be fixed in advance.
  - If I request all of my files that contain a combination of keywords, I should also specify how much data I want to be retrieved (e.g. 1MB).
  - From my request, the cloud will generate a circuit for a function that outputs the first megabyte of the correct files,
  - The output is truncated or padded with zeros prevent leaking something a priori about the relationship between the function (that is known) and my data.

But our case introduce further constrains-2

- ***semantic security*** against chosen-plaintext attacks (CPA) : given a ciphertext *c* that encrypts either *m*0 or *m*1, it is hard for an adversary A to decide which of the two values c encrypts, even if it is allowed to choose both *m*0 and *m*1.

  "hard" = if A runs in polynomial time and guesses correctly with probability 1/2 + *OE*, then *OE* = A's *advantage*, must be negligible. Otherwise, A *breaks* the semantic security of the encryption scheme.

- If an encryption scheme is *deterministic* (= there is only one ciphertext that encrypts a given message) then it cannot be semantically secure.

  An attacker can easily tell whether *c* encrypts *m*0 by encrypting *m*0 and by checking if the results is equal to *c*.

- A semantically secure encryption scheme must be *probabilistic*
  - several ciphertexts that encrypt a given message
  - encryption chooses one randomly according to some distribution

Encryption scheme e

- Four algorithms
  - KeyGen_e , Encrypt_e , Decrypt_e (must be efficient)
  - Evaluate_e
- *Efficient* = runs in time poly(L) where L = bit-length of the keys.
- KeyGen_e uses L to generate
  - a single key sk in a *symmetric* scheme,
  - two keys an *asymmetric* scheme, a public key pk and secret key sk.
- Evaluate_e is associated to a set F_e of *permitted functions* such that
  - f in F_e
  - if $c1, \ldots, ct$ are such that $ci = \text{Encrypt}_e(\text{pk}, mi)$ then
    - $\text{Evaluate}_e,(\text{pk}, f, c1, \ldots, ct) = c$
    - $f(m1, \ldots, mt) = \text{Decrypt}_e(\text{sk}, c)$ (sk if symmetric)

e is fully homomorphic if any function belongs to F_e

Constrains

- decrypting *c,* the output of Evaluate$_e$ takes the *same amount of computation* as decrypting *c*1, a ciphertext output by Encrypt$_e$
- *c* is the same size as *c*1 (*compact ciphertexts* requirement)

Informally,

- the size of *c* and the time needed to decrypt it do not grow with the complexity of *f*; rather, they are *completely independent* of *f*
- the complexity of Decrypt$_e$, as well as those of KeyGen$_e$ and Encrypt$_e$, must remain polynomial in L

A first approximation - 1

Assume L= N, P = L^2 and Q = L^5 *A (symmetric) Encryption Scheme*

KeyGen$_e$(L): The key is a random P-bit odd integer p.

Encrypt$_e$(p, m): To encrypt a bit m in {0, 1},

1) choose a random N-bit number m' such that m' = m mod 2.
2) output c= m' + pq, where q is a random Q-bit number.

Decrypt$_e$(p, c): Output (c mod p) mod 2 where

1) (c mod p) = c' in ($-p/2$,$p/2$) ← m' and m have the same parity
2) p divides $c - c'$

we recover q by finding the multiple of p closest to c and the noise parity is the encrypted bit

(c mod p) = *noise* associated to the ciphertext c
= distance to the nearest multiple of p

Decryption works because the noise m' has the same parity as the message m.
A ciphertext output by Encrypt is a *fresh* ciphertext, since it has small (N-bit) noise.

A first approximation – 1 bis

A Somewhat Homomorphic Scheme

- KEYGEN_ϵ: Output a random odd integer p
- For bit $m \in \{0, 1\}$, let a random $m' = m \bmod 2$ (ie. m' is even if $m = 0$, odd if $m = 1$). Pick a random q. Then $\text{ENCRYPT}_\epsilon(m, p) = c = m' + pq$. m' is the noise associated with the plaintext.
- Let $c' = c \bmod p$ where $c' \in (-p/2, p/2)$. Then $\text{DECRYPT}_\epsilon(p, c) = c' \bmod 2$. c' is considered to be the noise associated with the ciphertext (ie. the shortest distance to a multiple of p)

The Homomorphism: (Multiplication) Let $m_1, m_2 \in \{0, 1\}$. Then

$$e(m_1, p)e(m_2, p) = (m_1' + pq_1)(m_2' + pq_2)$$

$$\implies d(c) = (m_1' + pq_1)(m_2' + pq_2) \bmod p \bmod 2 = m_1' \cdot m_2' \bmod 2 = m_1 \cdot m_2$$

A first approximation - 2

$\text{Add}_e(c1, c2) = c1 + c2$

$\text{Sub}_e(c1, c2) = c1 - c2$

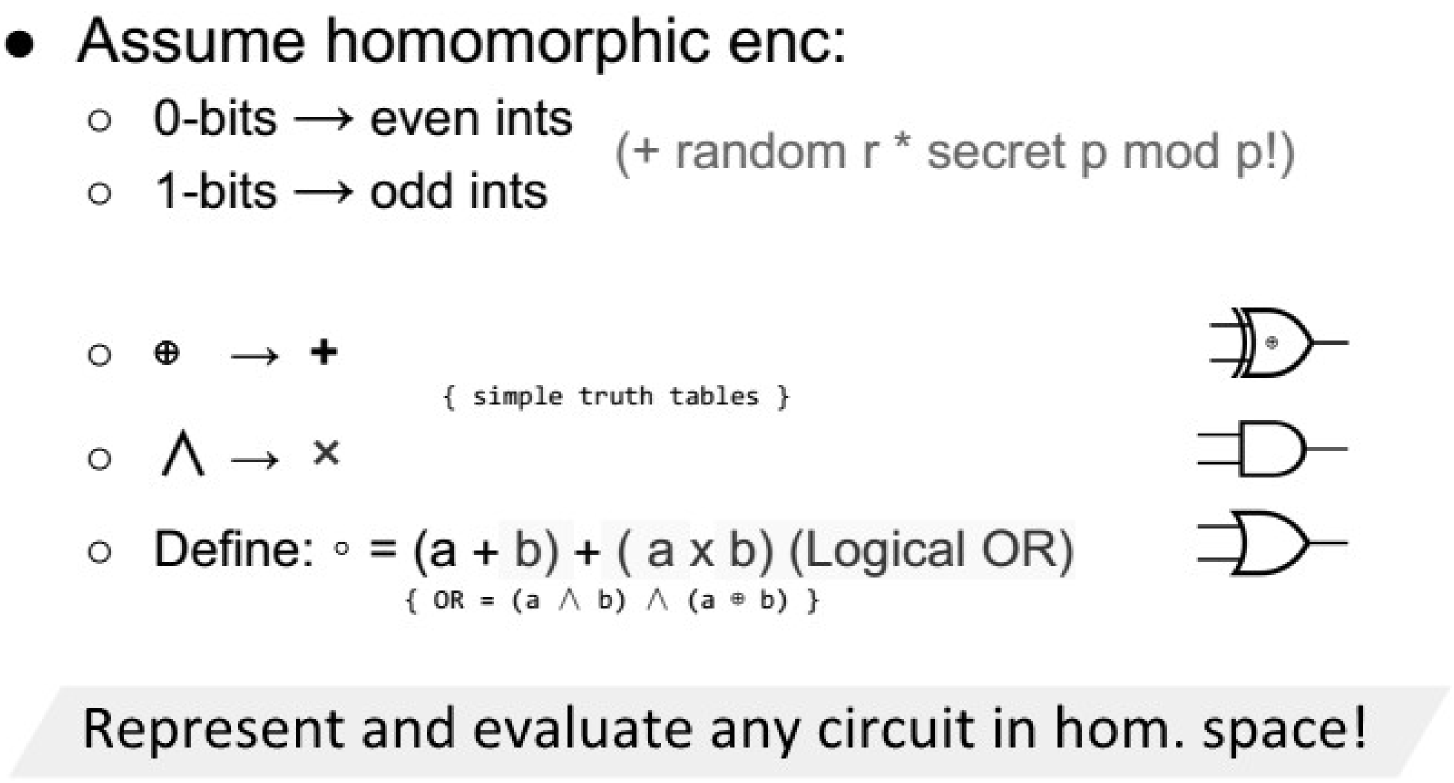
$\text{Mult}_e(c1, c2) = c1 \bullet c2.$

$\text{Evaluate}_e(f, c1, \ldots, ct) =$

1) Express *f* as a circuit *C* with XOR and AND gates
2) Let *C*' be the same circuit as *C*, but with XOR and AND gates replaced by addition and multiplication gates over the integers.
3) Output $c = f'(c1, \ldots, ct)$ where *f*' is the multivariate polynomial that corresponds to *C*'.

If this work, we can deduce a pubblic encryption scheme

Example



- Assume homomorphic enc:
  - 0-bits ⟶ even ints
  - 1-bits ⟶ odd ints

(+ random r * secret p mod p!)

  - ⊕ → +

{ simple truth tables }

  - ∧ → ×
  - Define: ∘ = (a + b) + (a x b) (Logical OR)

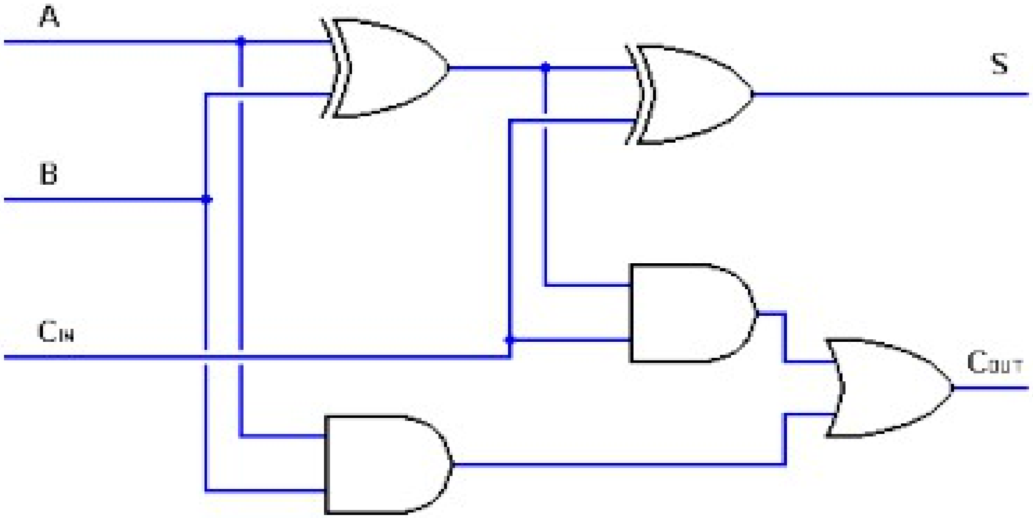
{ OR = (a ∧ b) ∧ (a ⊕ b) }

Represent and evaluate any circuit in hom. space!

Example

Single Bit Adder

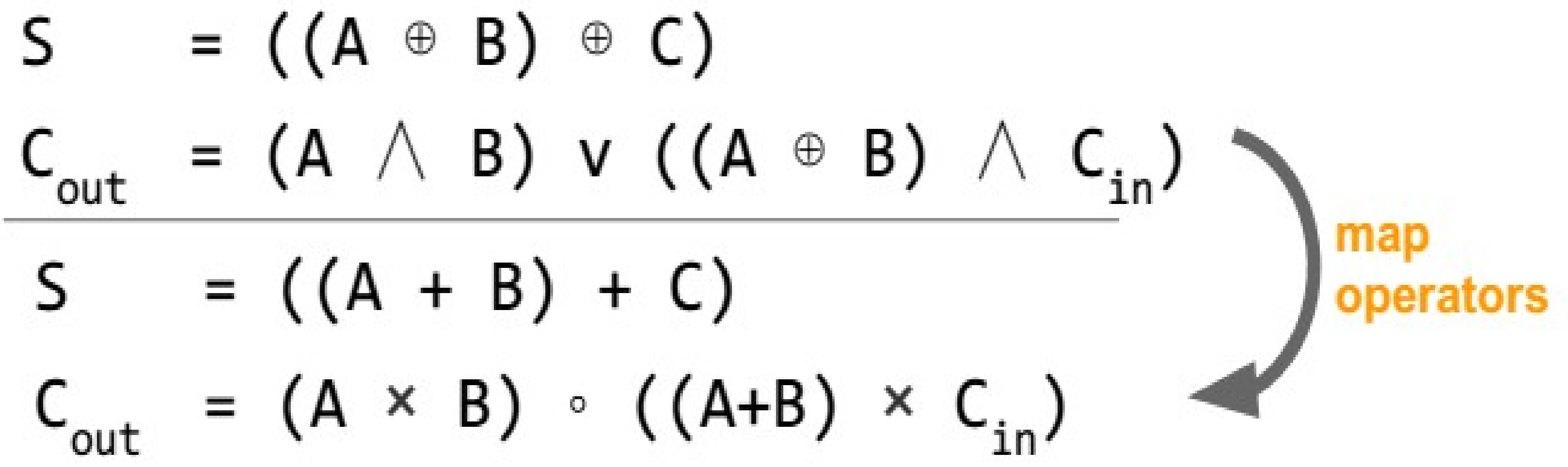
- A,B: inputs, C_{in}: carry-in, S: sum, C_{out}: carry-out



$$S = ((A \oplus B) \oplus C)$$

$$C_{out} = (A \wedge B) \vee ((A \oplus B) \wedge C_{in})$$

Example



$$S = ((A \oplus B) \oplus C)$$

$$C_{out} = (A \wedge B) \vee ((A \oplus B) \wedge C_{in})$$

$$S = ((A + B) + C)$$

$$C_{out} = (A \times B) \circ ((A+B) \times C_{in})$$

Example

$$S = ((A + B) + C)$$

apply

	A	B	C_{in}	S	C_{out}
	1	0	1	0	1
encrypted	3	4	7	?	?

$$S = ((3 + 4) + 7) = ?$$

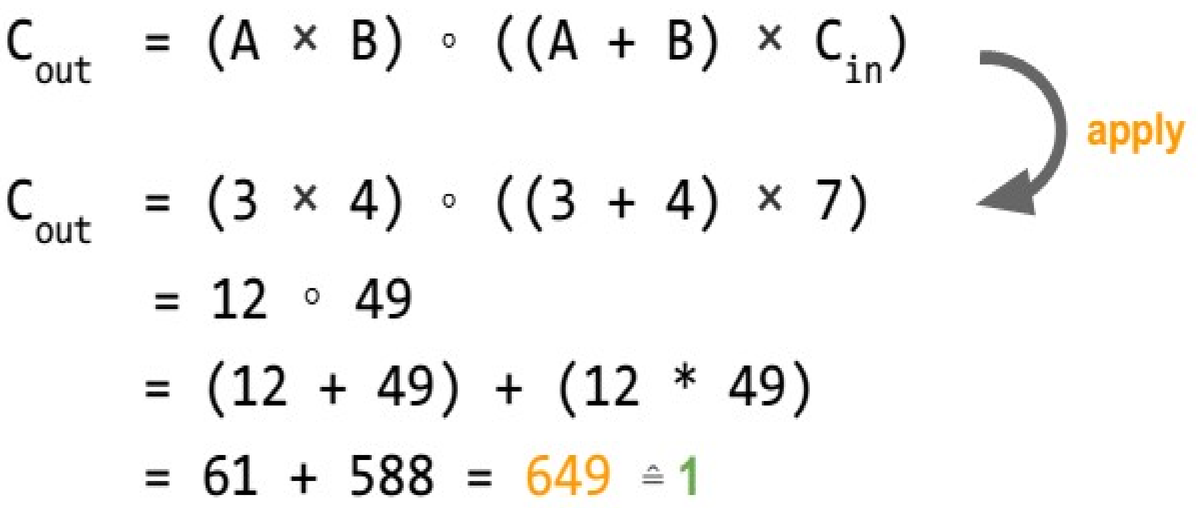
Example

$$S = ((A + B) + C)$$

	A	B	C_{in}	S	C_{out}
	1	0	1	0	1
encrypted	3	4	7	14	?

$$S = ((3 + 4) + 7) = 14 \triangleq 0$$ ✓

Example



$$C_{out} = (A \times B) \circ ((A + B) \times C_{in})$$

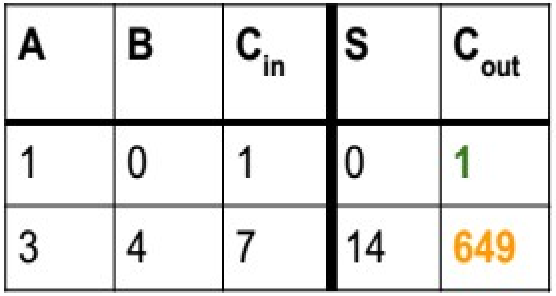
apply

$$C_{out} = (3 \times 4) \circ ((3 + 4) \times 7)$$

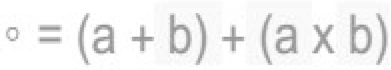
$$= 12 \circ 49$$

$$= (12 + 49) + (12 * 49)$$

$$= 61 + 588 = 649 \mathrel{\hat{=}} 1$$



A	B	C_{in}	S	C_{out}
1	0	1	0	1
3	4	7	14	649



$\circ = (a + b) + (a \times b)$

Does it works ? - 1

Let us consider $c = c1 \cdot c2$,

We have that $c = (m'1 + pq1)\ (m'2 + pq2) =$
$= m'1 \cdot m'2 + pq'$ for some integer q'.

Assume that noises $m'1$ and $m'2$ are small enough, so that
$| m'1 \cdot m'2 | < p/2$, This implies that
$(c \bmod p) = m'1 \cdot m'2$

and since c_i's noise is m'_i, which has the same parity as mi.

$(c \bmod p) \bmod 2 = m1 \cdot m2$

If $c1$ and $c2$ have $k1$- and $k2$-bit noises, the ciphertext of $c1 \cdot c2$ has (roughly) $(k1 + k2)$-bit noise.

Does it works ? - 2

What happens when we perform *many* Add_e, Sub_e, and Mult_e as prescribed by the circuit representing a function f ?

1) We have that $f’’(c1, ..., ct) = f‘(m’1, ..., m’\mathrm{t}) + pq’$ for some integer $q’$, where $m’t$ is the noise associated to ci.
2) If $| f‘(m’1, ..., m’t)| < p/2$, then ($f‘(c1, …, ct) \bmod p) = f‘(m’1, ..., m’t)$.
3) By reducing modulo 2, we obtain the correct result: $f(m1, …, mt)$.

This show that e can handle those functions for which $| f‘ (a1, …, at)|$ is *always* less than $p/2$ if all of the ai are at most N bits

Bootstrappable Encryption

e is *bootstrappable* if it can handle its own decryption function augmented by a single gate= the functions that e can handles includes its own decryption function composed with some useful work, one gate

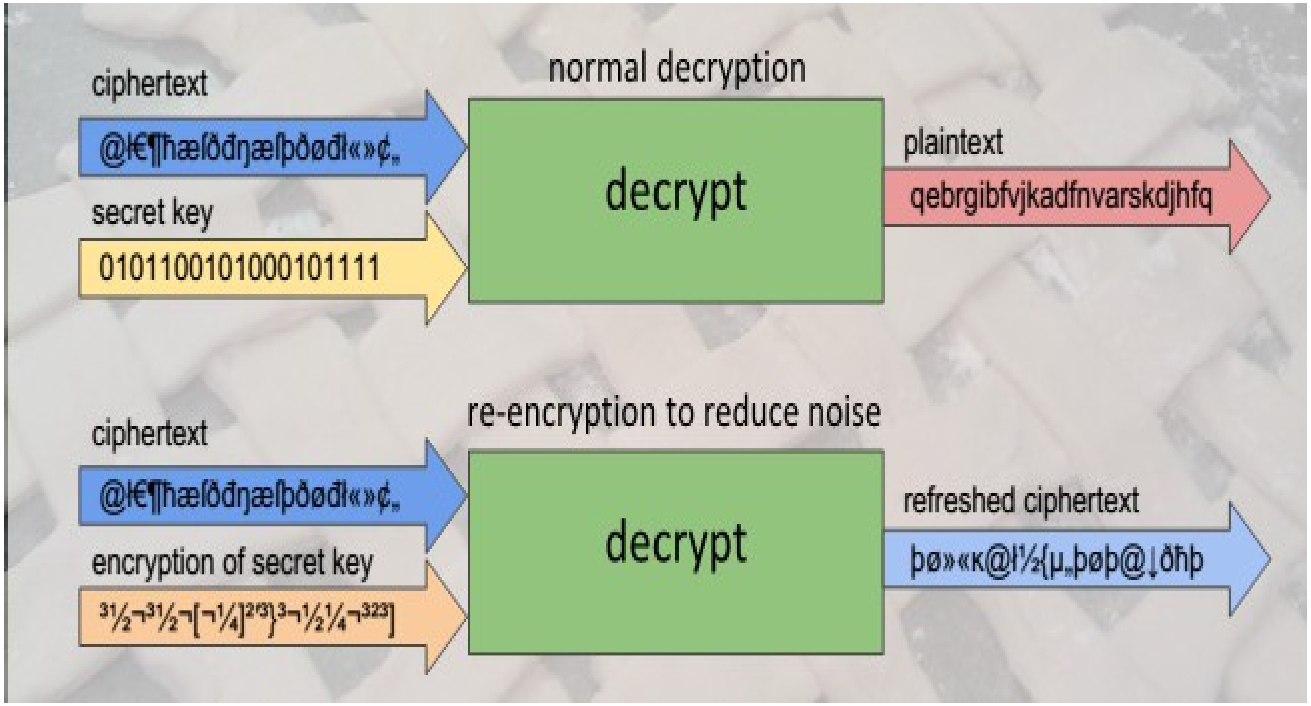
Suppose that *e* can handle

1. the decryption function, expressed as a circuit *De* of size polynomial in L
2. *De* augmented by an Add, Sub, or Mult gate modulo 2.

   De augmented by Add = two copies of *De* connected by an Add gate.

   This is a *complete* set of circuits, in the sense that if these four circuits are in F*e*, then one can construct from *e* a scheme *e*' that is fully homomorphic.

Bootstrappable Encryption



Bootstrappable Encryption

1. Assume that
   1 c1 encrypts m under key pk1
   2 sk1 is an encrypted secret key
   3 skv1 is a vector of ciphertexts that encrypt the bits of sk1 under pk2 via Encrypt$_e$(pk2, sk1j).

Recrypt$_e$(pk2, D_e, skv1, c1).

1. Generate c ‘1 via Encrypt$_e$ (pk2, c1j) over the bits of c1
2. Output c = Evaluate$_e$ (pk2, D_e, sk1, c ‘1)

- The decryption circuit D_e has input wires for
  1. the bits of a secret key
  2. the bits of a ciphertext.
- Evaluate$_e$ takes in the bits of sk1 and c1, each encrypted under pk2.

As long as e can handle D_e :

a) e is used to evaluate the decryption circuit homomorphically.
b) the output c is an encryption under pk2 of Decrypt$_e$(sk1, c1) = m.
c) Recrypt$_e$ outputs a new encryption of m, but under pk2.

Bootstrappable Encryption

To understand Recrypt*e* consider that *m* is doubly encrypted at one point,

a) first under pk1

b) next under pk2.

Ordinarily, the only thing one can do with a doubly encrypted message is to peel off the outer encryption first, and then decrypt the inner layer.

Instead, Recrypt*e*

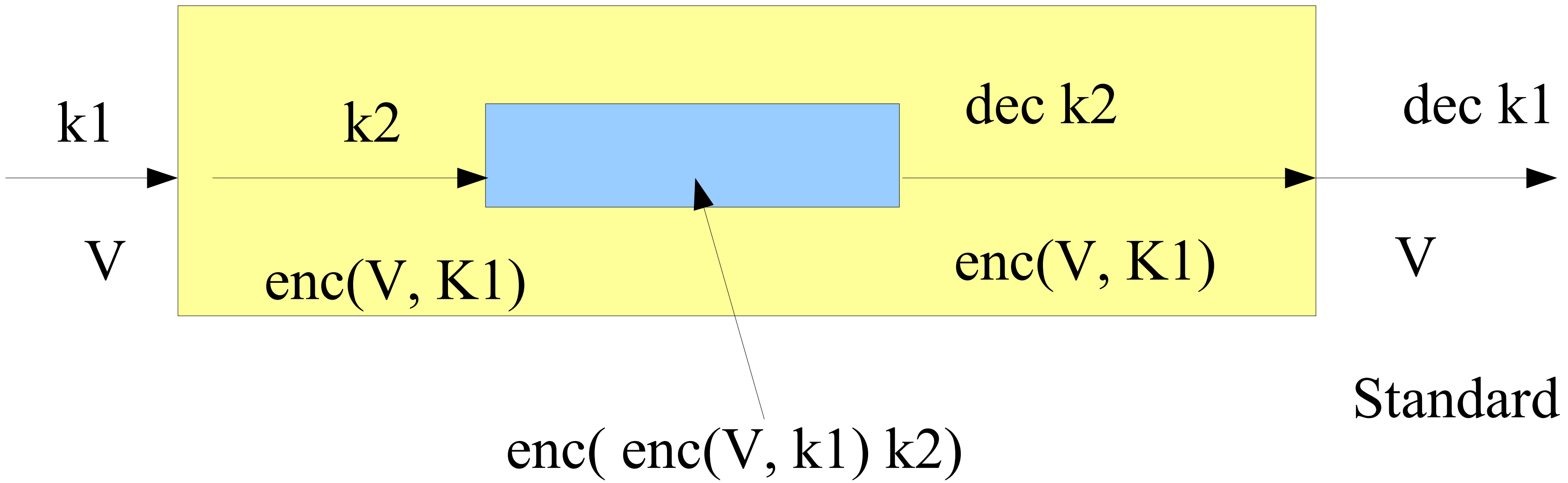
a) uses Evaluate$_e$ algorithm to remove the *inner* encryption,

b) by evaluating *De* removes the noise associated to the first ciphertext under pk1 (decryption removes noise),

c) simultaneously introduces new noise by evaluating through Evaluate*e* the ciphertexts under pk2.

As long as the noise added is less than the removed one, we have made “progress.”
Obviously we have to add some functions (Add, Sub, Mul)

Bootstrappable Encryption



k1

V

k2

enc(V, K1)

dec k2

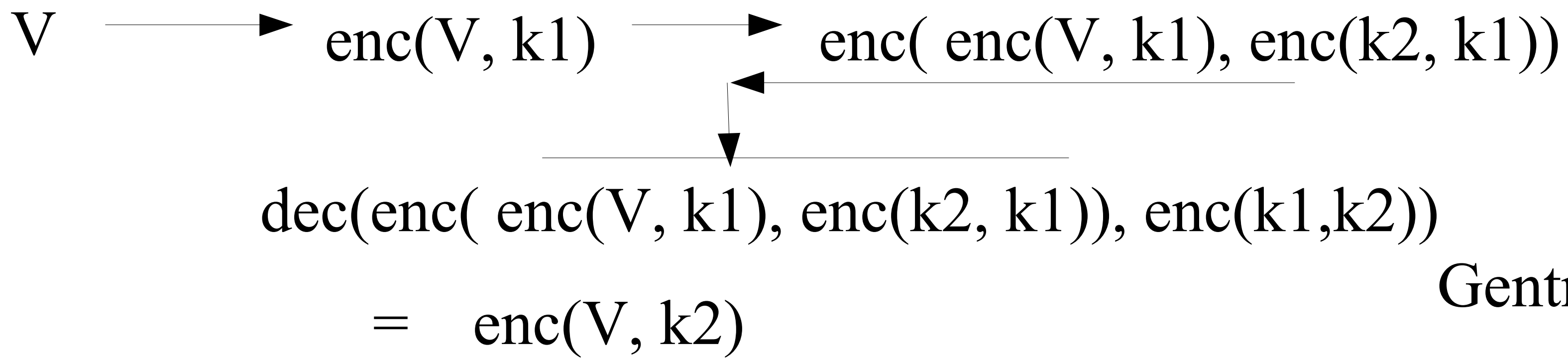
enc(V, K1)

dec k1

V

enc(enc(V, k1) k2)

Standard



V

enc(V, k1)

enc(enc(V, k1), enc(k2, k1))

dec(enc(enc(V, k1), enc(k2, k1)), enc(k1,k2))

= enc(V, k2)

Gentry

A full homomorphic scheme

Suppose that

a) *e* can handle *De* augmented by some gate, e.g., Add; call this augmented circuit *D*Add.

b) *c*1 and *c*2 encrypt *m*1 and *m*2 respectively, under pk1,

if *c’* 1 and *c‘* 2 encrypt the bits of the ciphertexts under pk2 then

c =Evaluate$_e$(pk2, *D*Add, sk1, *c* ‘1 , *c* ‘2)

encrypts *m*1 ⊕ *m*2 under pk2 .

We get a fully homomorphic encryption scheme *e*’ by recursing this process where the key in *e*’ is

a) a sequence of public keys (pk1, …, pka+1)

b) a chain of encrypted secret keys sk1, ..., ska, where sk*i* is encrypted under pk*i*+1.

A full homomorphic scheme

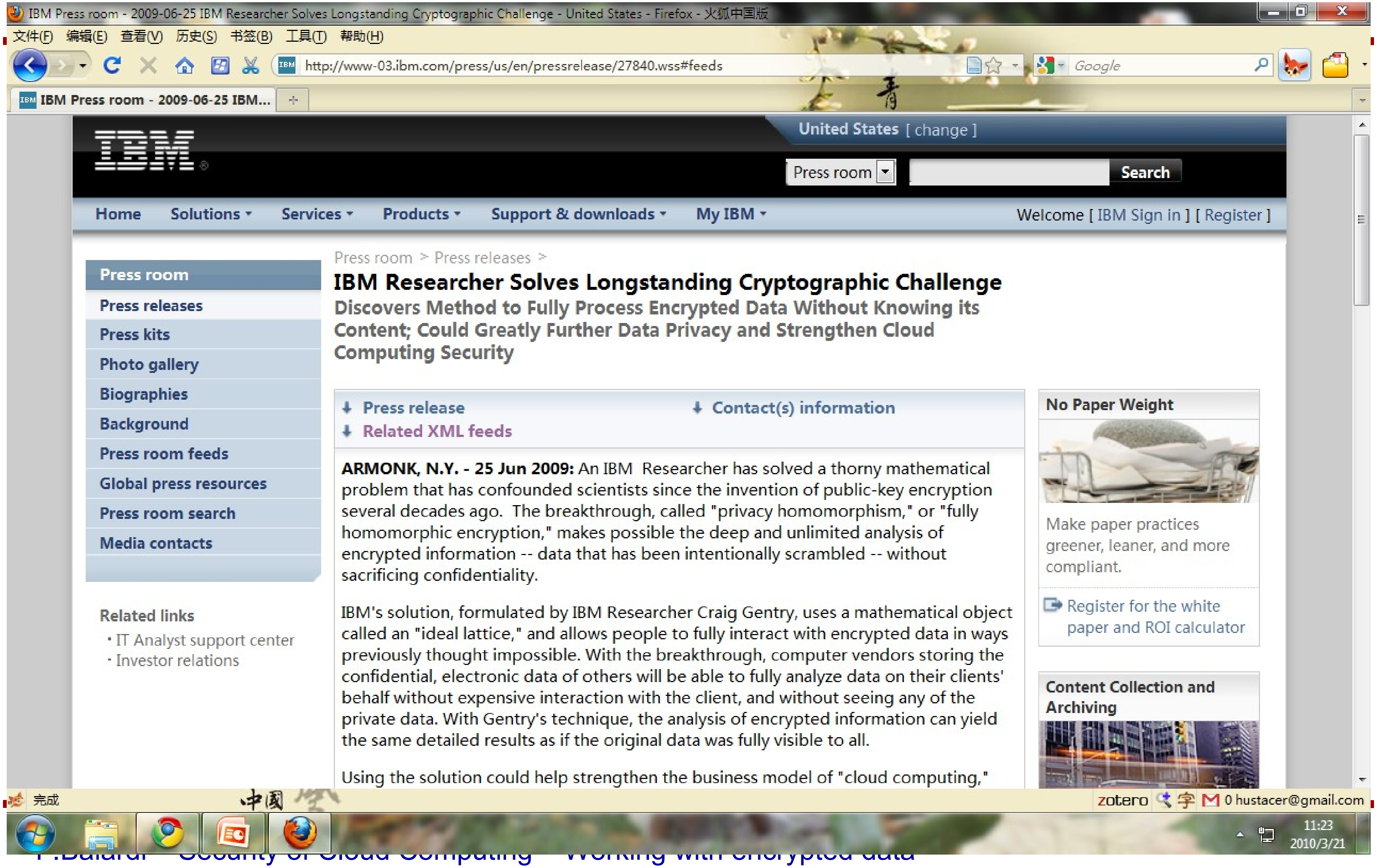
To evaluate a function *f* in *e*',

1. we express *f* as a circuit, topologically arrange its gates into levels,
2. scan sequentially the levels and for a gate at level *i* + 1(e.g., an Add gate)
    1. take as input the encrypted secret key sk*i* and a couple of ciphertexts associated to output wires at level *i* that are under pk*i*,
    2. homomorphically evaluate *D*Add to get a ciphertext under pk*i*+1 associated to a wire at level *i* + 1.
3. output the ciphertext associated to the output wire of *f*.

Putting the encrypted secret key bits sk1, ..., ska in the public key of *e*' is not a problem for security because these bits are indistinguishable from encryptions of 0 as long as *e* is semantically secure

Last step: reduce the complexity of the key, instead of several pubblic keys we have the same key for all the level (no information is leaked by revealing the encyption of a secret key under a pubblic key, circular security)

Breakthrough(2009)


IBM Press room - 2009-06-25 IBM Researcher Solves Longstanding Cryptographic Challenge - United States - Firefox - 火狐中国版
文件(F) 编辑(E) 查看(V) 历史(S) 书签(B) 工具(T) 帮助(H)
http://www-03.ibm.com/press/us/en/pressrelease/27840.wss#feeds
Google
IBM Press room - 2009-06-25 IBM...
IBM
United States [change]
Press room
Search
Home
Solutions
Services
Products
Support & downloads
My IBM
Welcome [IBM Sign in] [Register]
Press room
Press releases
Press kits
Photo gallery
Biographies
Background
Press room feeds
Global press resources
Press room search
Media contacts
Related links
IT Analyst support center
Investor relations
Press room > Press releases >
Press release
Contact(s) information
Related XML feeds
No Paper Weight
Make paper practices greener, leaner, and more compliant.
Register for the white paper and ROI calculator
Content Collection and Archiving
完成
zotero
0 hustacer@gmail.com
11:23
2010/3/21

IBM Researcher Solves Longstanding Cryptographic Challenge

Discovers Method to Fully Process Encrypted Data Without Knowing its Content; Could Greatly Further Data Privacy and Strengthen Cloud Computing Security

ARMONK, N.Y. - 25 Jun 2009: An IBM Researcher has solved a thorny mathematical problem that has confounded scientists since the invention of public-key encryption several decades ago. The breakthrough, called "privacy homomorphism," or "fully homomorphic encryption," makes possible the deep and unlimited analysis of encrypted information -- data that has been intentionally scrambled -- without sacrificing confidentiality.

IBM's solution, formulated by IBM Researcher Craig Gentry, uses a mathematical object called an "ideal lattice," and allows people to fully interact with encrypted data in ways previously thought impossible. With the breakthrough, computer vendors storing the confidential, electronic data of others will be able to fully analyze data on their clients' behalf without expensive interaction with the client, and without seeing any of the private data. With Gentry's technique, the analysis of encrypted information can yield the same detailed results as if the original data was fully visible to all.

Using the solution could help strengthen the business model of "cloud computing,"

Practical … ☺ ?

According to an article on Forbes.com, Gentry's solution has a catch: It requires immense computational effort. In the case of a Google search, for instance, performing the process with encrypted keywords would multiply the necessary computing time by around 1 trillion, Gentry estimates.

1 trilion = 10 ^{12}

If we exploit Moore’s law , after 40 years an homomorphic search would be as efficient as a search today

Practical … ☺ ? Bruce Scheiner

Unfortunately -- you knew that was coming, right? -- Gentry’s scheme is completely impractical. It uses something called an ideal lattice as the basis for the encryption scheme, and both the size of the ciphertext and the complexity of the encryption and decryption operations grow enormously with the number of operations you need to perform on the ciphertext -- and that number needs to be fixed in advance. And converting a computer program, even a simple one, into a Boolean circuit requires an enormous number of operations. These aren't impracticalities that can be solved with some clever optimization techniques and a few turns of Moore's Law;

Despite this, IBM’s PR machine has been in overdrive about the discovery. Its press release makes it sound like this new homomorphic scheme is going to rewrite the business of computing: not just cloud computing, but "enabling filters to identify spam, even in encrypted email, or protection information contained in electronic medical records." Maybe someday, but not in my lifetime.

This is not to take anything away anything from Gentry or his discovery. Visions of a fully homomorphic cryptosystem have been dancing in cryptographers' heads for thirty years. I never expected to see one. It will be years before a sufficient number of cryptographers examine the algorithm that we can have any confidence that the scheme is secure, but -- practicality be damned -- this is an amazing piece of work.

A simpler problem

- Let us consider now the case where the client wants to search a file stored by the cloud
- We assume that the file will never be updated (or that is updated outside the cloud)
- Obviously also in this case no information should be leaked so that the previous conditions about non deterministic encryption always holds
- We consider symmetric encryption since there is no need to transmit the result of the query

Tokenisation

For a set of documents, the client execute the following for each document before uploading it to a remote, untrusted server.

the input document is tokenised into a set of words, *W*. This tokenisation needs to still contain *all* of the input symbols, whilst separating words from punctuation.

For example, a sentence such as “Something, something2!” would need to be transformed into

{‘Something’ , ‘,<space>’ , ‘something2’ , ‘!’}.

The bundling of the first comma and space together into the same ‘word’ is probably acceptable, as it is unlikely that a user would want to search for “, ”.

A simpler problem

- 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
  - □ $G: K_G$ _ X^l is a pseudorandom generator for some l and $X = \{0, 1\}^{n-m}$
  - □ $F: K_F \times X$ _ Y is a pseudorandom function, $X = \{0, 1\}^{n-m}$ and $Y = \{0, 1\}^m$
  - □ $E: K_E \times Z$ _ Z is a pseudorandom permutation, $Z = X \times Y = \{0, 1\}^n$

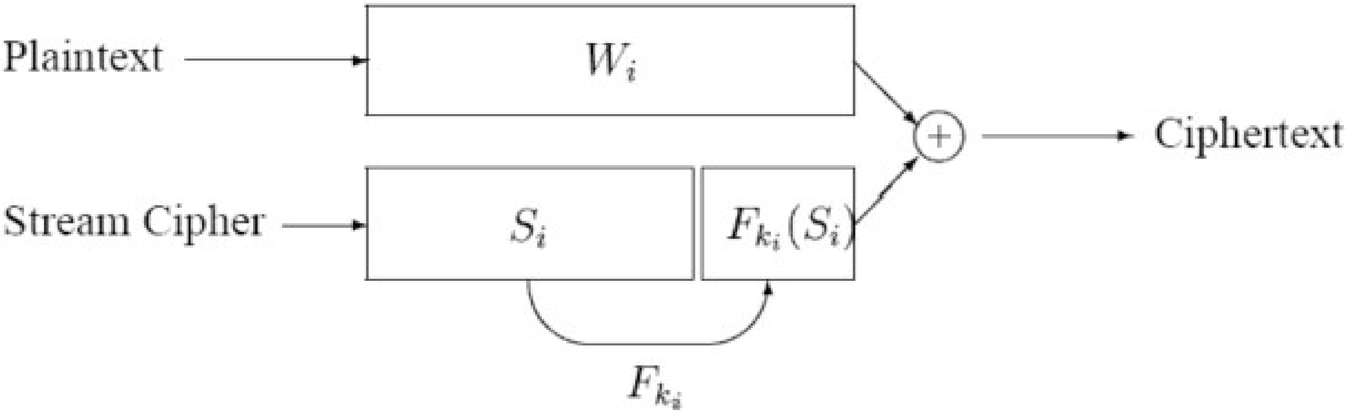
Constrains

- Leak no information about the structure of the file
- Leak no information about the query
- Avoid crypting and decrypting the file
- Avoid trusted third party

First solution

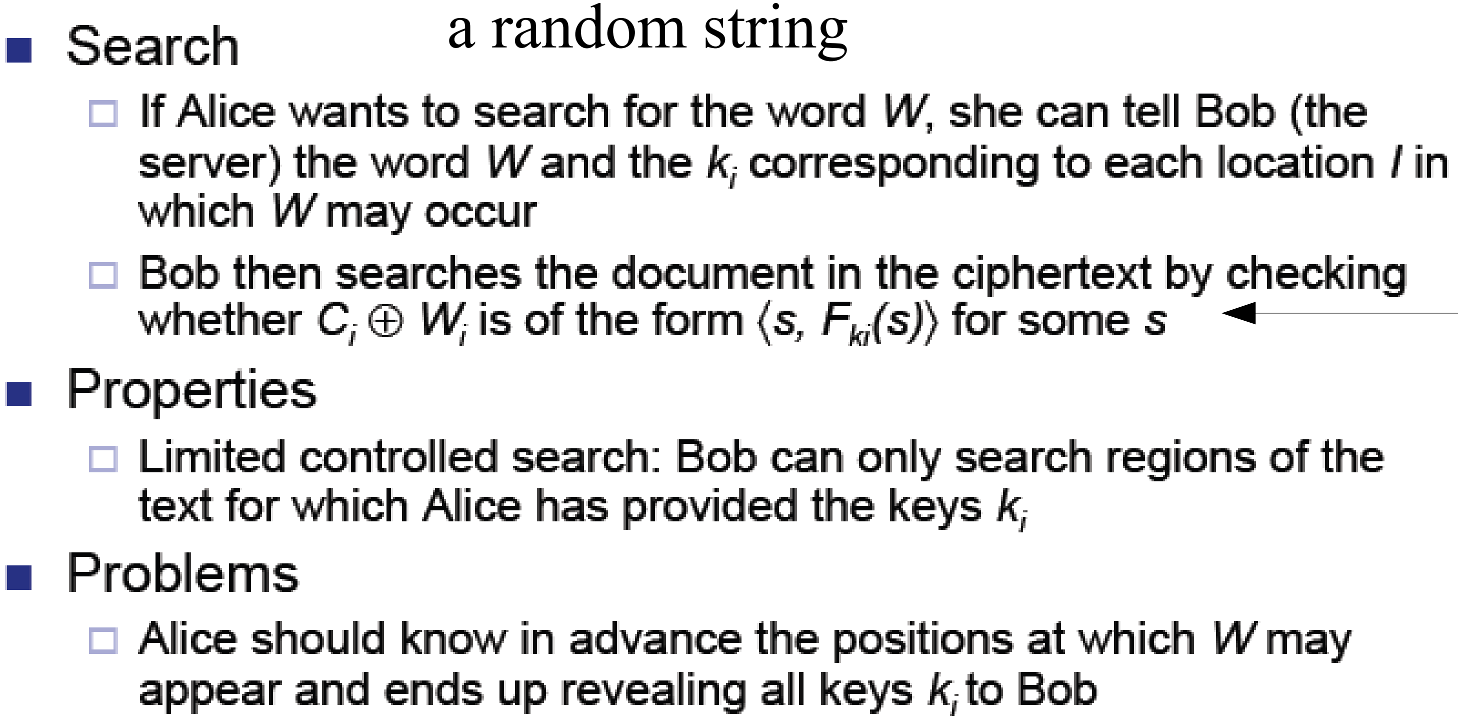
- Step by step
  - □ Alice generates a sequence of pseudorandom values $S_1, \ldots, S_l$ using G where each S_i is n-m bits long
  - □ She computes $T_i = \langle S_i, F_{ki}(S_i)\rangle$
  - ♦ She outputs the ciphertext $C_i = W_i \oplus T_i$
- Properties
  - □ The key k_i can be the same or chosen independently for every word
  - □ If F and G are secure, then the sequence of T_i is a secure pseudorandom generator

First solution



First solution - Problems

This is the reason why we cannot exor with a random string

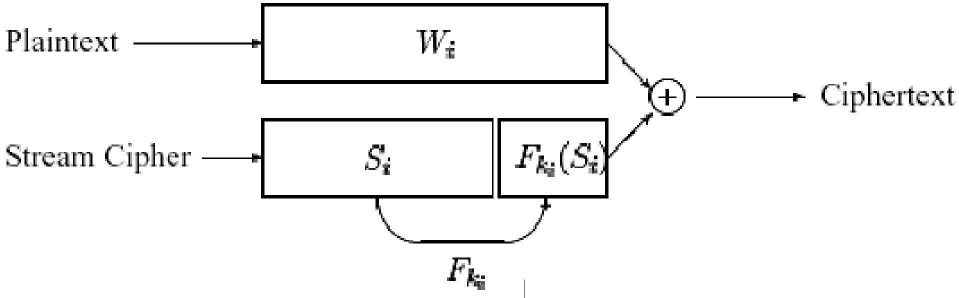


- Search
  - □ If Alice wants to search for the word *W*, she can tell Bob (the server) the word *W* and the k_i corresponding to each location *l* in which *W* may occur
  - □ Bob then searches the document in the ciphertext by checking whether $C_i \oplus W_i$ is of the form $\langle s, F_{ki}(s)\rangle$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

Improvement -1/Controlled Searching

- Alice uses an additional pseudorandom function f: K_F x $\{0, 1\}^*$ _ 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

Controlled Searching



To perform controlled searching, we tie the key k_i to the word W_i.

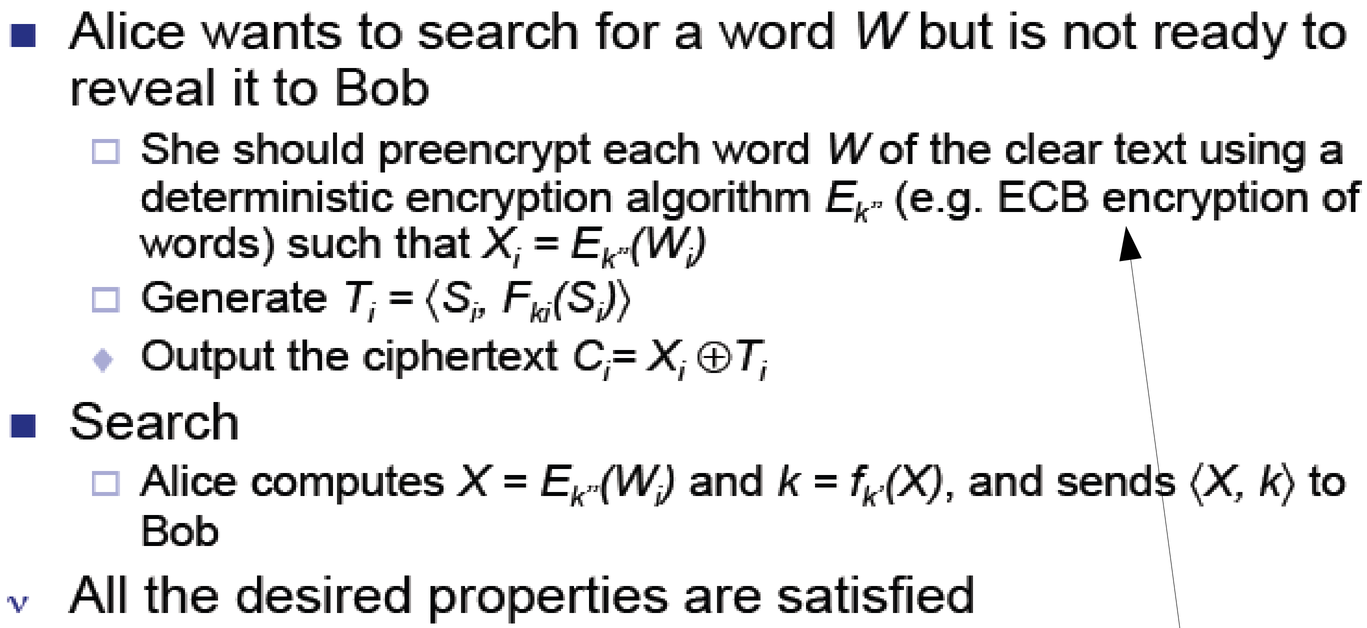
To do that, we introduce a new pseudorandom function $\mathrm{f} : \mathrm{K}_\mathrm{F} \times \{0,1\}^* \rightarrow K_F$

keyed with a secret key chosen uniformly at random. Now, $k_i = f_{k'}(W_i)$.

To search : the untrusted server is given W and $\mathrm{f}_{\mathrm{k_i}}(\mathrm{W})$.

How do we decrypt now?
The issue of hiding search queries is still unresolved.

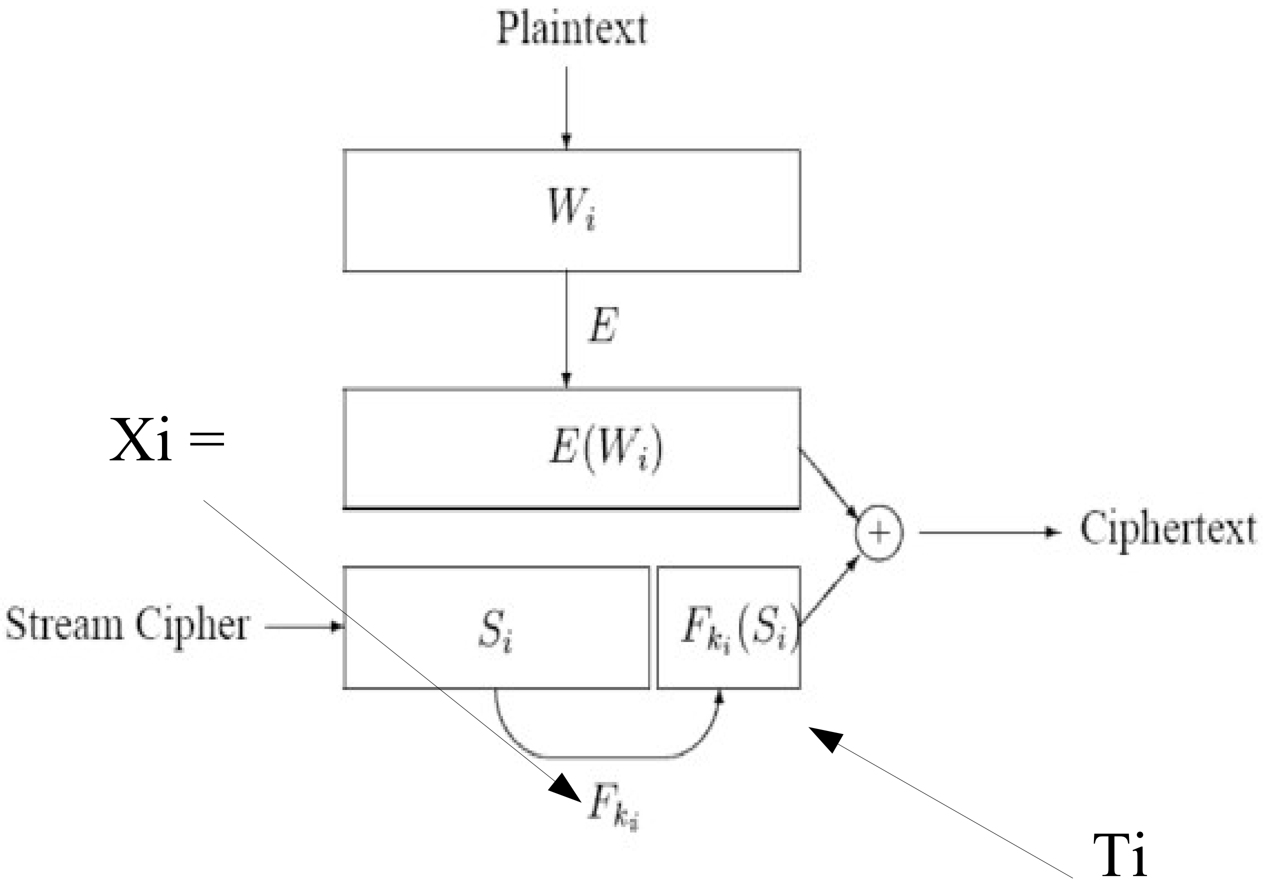
Improvement - 2



- 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 $T_i = \langle S_i, F_{ki}(S_i)\rangle$
  - ♦ 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 $\langle X, k\rangle$ to Bob
- All the desired properties are satisfied

AES mode

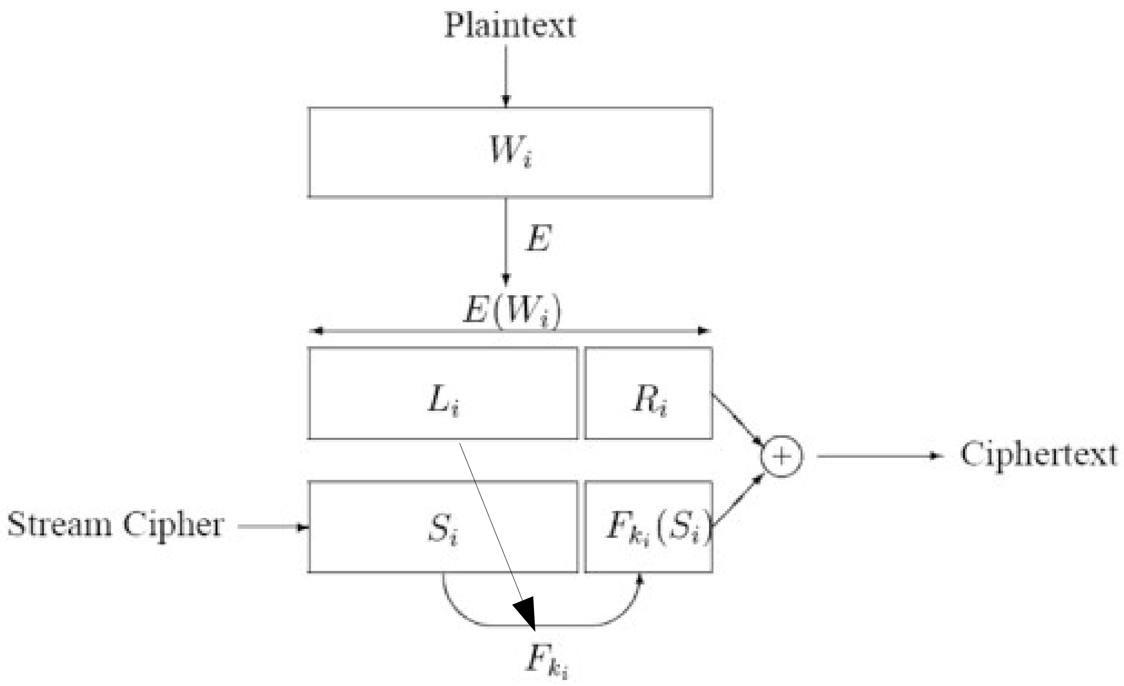
Improvement - 2



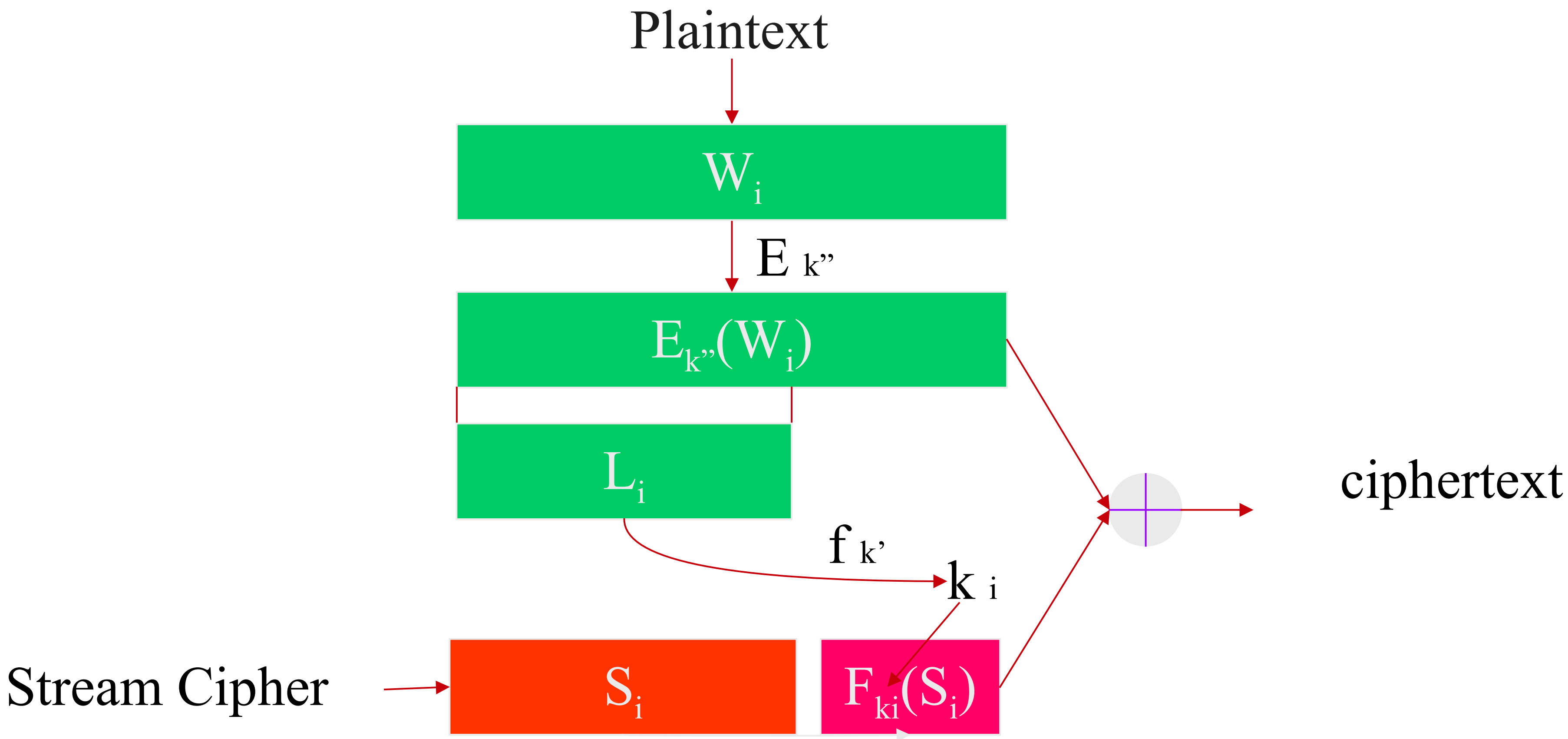
Problem

- Problem with previous solution
  - 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 $\langle L_i, R_i \rangle$, 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

Complete solution



The Final Picture



A Complete Solution - 1

1. Generate *k*1, *k*2 and *k*3 from the master key for encryption (e.g. password). These keys
   1. must be different and should be derived from the master private key in such a way that knowing *k*1 doesn't reveal either *k*2 or *k*3 (as well as all other combinations).
   2. allow us to reveal at most two keys to the server, giving it enough information to perform a search, but not to decrypt the document or understand what we're searching for.

Tokenise the file and repeat the following for each word Wi

2. Wi is encrypted with a standard block cipher. This uses *k*2 as generated in the previous step and can be performed using either ECB mode, or CBC mode with a fixed IV.

$$Xi = Ek2\ (Wi)$$

3. The next step takes *x* bits from the stream cipher *G* seeded on the key *k*3. These bits are denoted as *Si and their length is lower* than that of the encrypted word, *Xi*. The choice of *x* should be pre-determined and be consistent throughout the system.

A Complete Solution - 2

4. The encrypted word, Xi , is then split into left and right halves (Li and Ri) where
    1. the length of Li is x
    2. the length of Ri is $length(Xi) - x$.

$$Xi = \langle Li\ , Ri \rangle$$

5. A word specific key, ki , is then created by combining the left half, Li , with the key $k1$ before hashing it.

$$ki = f_{k1}(Li)$$

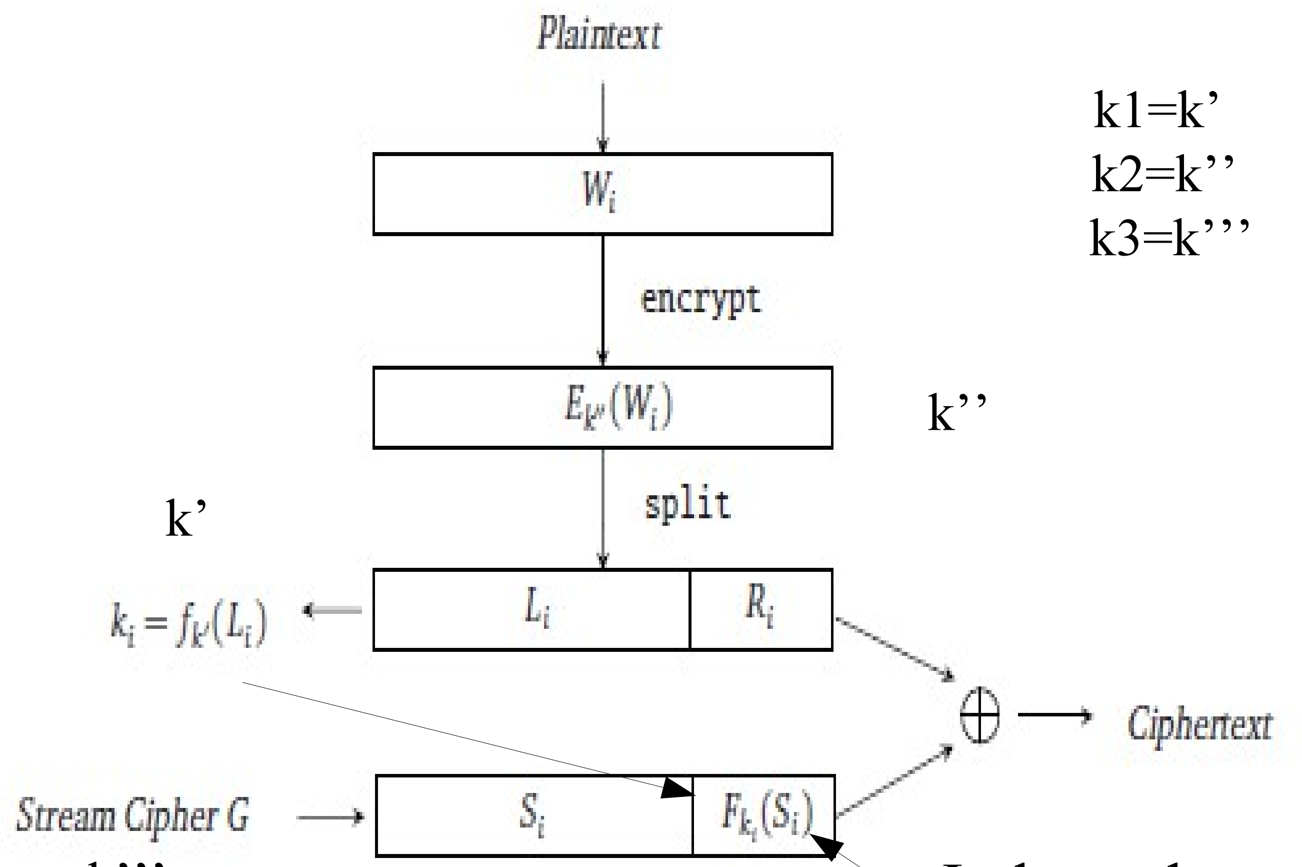
6. Then Si *(from step 3)* is combined with ki either through a process such as XOR or concatenation before being hashed to produce a number of bits, equal in length to that of Ri .

$$F_{ki}(Si)$$

7. The final step performs a XOR between $\langle Li\ , Ri \rangle$ and $\langle Si\ , F_{ki}(Si) \rangle$.

$$Cipher\ tex\ t = \langle Li\ , Ri \rangle \oplus \langle Si\ , F_{ki}(Si) \rangle.$$

A Complete Solution - 3



k1=k'
k2=k''
k3=k'''

k''

k'

k'''

It depends upon L_i and S_i

Search client

Given a word *W*, the **client** :

1. Generates *k*1, *k*2 and *k*3
2. Encrypts word *W* using the same block cipher and key (*k*2) as the encryption process to produce the encrypted word, *X*

$$X = Ek2\ (W)$$

3. Extracts the left part (*L*), consisting of the same *x* bits as used in the encryption process and uses it to generate the word-specific key, *k,* as in step 5 of the encryption process by combining they key *k*1 with the left part of *X* before hashing them.

$$k = f_{k1}(L)$$

4. Sends $\langle X, k \rangle$ to the **untrusted server**

Search server

1. For each encrypted word block *C* in the document, XOR it with the encrypted word *X* to produce the pair $\langle Si , F_k(Si)\rangle$
2. Since the length of *x* in step 3 of encryption is known, the server takes this number of bits from the front of $\langle Si , F_k(Si)\rangle$ to retrieve *Si* , the bits the encryption takes from the stream cipher
3. Since the server knows both *Si* and *k*, received by the client, it can combine *Si* with *k* and hashed using the same process as in step 6 of encryption and compared to the right part of the pair = $F_k(Si)$
4. If this matches, then the word is found and the current document can be added to a list of documents, to be returned to the client after the entire document set is inspected

This process leaks *no* information about what word is being searched for to the server, whilst still allowing it to determine matching documents

Decryption

It is similar to searching.

After downloading a document, the client generates the three keys k', k'' and k then it iterates over each encrypted block, C, as follows:

- Take x bits from the stream cipher, seeded on key k''', to create Si before XORing them with the first x bits of the ciphertext block, C to reveal the left part of the ciphertext word, L.
- To determine the right part, R, the client know Si and L and can generate the word specific key k as in the encryption process.

$$k = f_{k'}(L)$$

3. Using k, the client generates $F_k(Si)$. which can be used to fully restore X the encrypted word.
4. Since the client knows the pair $\langle Si\ , F_k(Si)\rangle$ and the ciphertext, it can XOR them to retrieve X.
5. The client knows the key k'' used to encrypt X so it can trivially retrieve the plaintext word.

Order preserving encryption

Assume we want to encrypt a list of values in a way that preserves some ordering among the elements

We want to encrypt v_1, …, v_n into ev_1, …, ev_n so that

$v_i <_o v_j$ implies $ev_i < ev_j$

- Choose a random generator RG
- Choose a key K to use as the seed of the random generator
- Let ord(v_1, …, v_n) be the ordered permutation of v_1, …, v_n according to $<_o$
- We encrypt the i-the element of ord(v_1, …, v_n) as

RG(K)+RG(RG(K))+ … Rg^i(K)

We store an encrypted database on the cloud and can answer questions such as “send me all the records of people earning more than x” provided that

- there is one people earning x or
- we encrypted all the possible earnings

Some information leaks …

Alternative solution: Bloom filter

- We build an index of a file to simplify the search
- Efficient method to encode set membership
- The set: n elements (n is large)
- The Bloom filter: array of m bits (m is small)
- q independent hash functions:

  $h_1:\{0,1\}^* \rightarrow [1,m]$, …., $h_i:\{0,1\}^* \rightarrow [1,m]$, …, $h_q:\{0,1\}^* \rightarrow [1,m]$;

Properties

– History independent

– Once added, elements can’t be removed

Bloom Filter

- For each element $s \in S$, the array bits at positions $h_1(s), \ldots,$ $h_q(s)$ are set to 1.

- A location can be set to 1 multiple times, but only the first is noted.
- To determine if an element a belongs to the set S, we check the bits at positions $h_1(a), \ldots, h_q(a)$
  - If all the checked bits are 1's, then we assume a is a member of the set.
  - There is, however, some probability of a false positive, in which a appears to be in S but actually is not.
  - False positives occur because each location may have also been set by some element other than a.
  - On the other hand, if any checked bits are 0, then a is definitely not a member of S = no false negatives.

Bloom Filter

h1(‘water’)=2 h1(‘sky’)=1

h2(‘water’)=5 h2(‘sky’)=5 q=3

h3(‘water’)=9 h3(‘sky’)=7

1	1			1		1		1	
1	2	3	4	5	6	7	8	9	10

h1(‘air’)=2, h2(‘air’)=5, h3(‘air’)=7 simultaneously = false positive!

To minimize false positive rate FP, need to choose

$$r = \ln 2 * \frac{m}{n} \qquad FP = \left(\frac{1}{2}\right)^{r}$$

Bloom Filter: multiple occurrence

To record multiple occurrence, when useful

h1('water1')

h2('water1')

1	1			1		1		1	
1	2	3	4	5	6	7	8	9	10

h1('water2')

h2('water2')

h3('water2')

Encrypted Bloom Filter

Restrict ability to compute the hash functions by introducing a secret

h1(w,k1) = f(w,k1)

h2(w,k2) = f(w,k2)

… …

hq(w,kq) = f(w,kq)

Private and Secure indexes

- Index is an additional structure that allows the remote server to perform searches efficiently
- Computed over unencrypted documents
- Private index should preserve user’s privacy
- Indexes associated with each document
- Security model: IND-CKA semantic security against adaptive chosen keyword attack

  = a secure index does not reveal anything about the document’s content
- Security game:

  given two encrypted documents of equal size, and an index, decide which document is encoded in the index

Secure Index

- An index is a Bloom filter, with pseudorandom functions used as hash functions
- A collection of 4 algorithms:
  - Keygen(s)
  - Trapdoor(Kpriv,w)
  - BuildIndex(D,Kpriv)
  - SearchIndex(Tw,ID)
- Keygen generates:
  - pseudo-random function f
  - master key Kpriv=(k1,…,kr)

Secure Index

- □□□□□□□(s)Keygen(s): Choose a random s bit master keys Kpriv and return it.

- □□□□□□□□(w,Kpriv):
  Given n an arbitrary word ww, output its trapdoor representation
  Tw=fT(w,Kpriv)

Secure Index

- □□□□□□□□□□(D,Kpriv)
  Given a document DD with unique ID Did∈{0,1}∗ and a list of distinct words (w1,w2
  …,wt) initialize the $m=2^b$ bit Bloom filter array BB to zeros and, for each word wi:

- Compute the trapdoor token xi=fT(wi,Kpriv)
- Compute the rb bit codeword fB(Did,xi) and split it into
  r indices $(y_{i,1},y_{i,2},\ldots,y_{i,r})$ of b bits each.
- Set the bits $(y_{i,1},y_{i,2},\ldots,y_{i,r})$ of the Bloom filter array BB to 1.
- To hide the number of distinct words in the document,
  - pick (with replacement) (u−t)r additional random bits of BB, where u is some publicly known upper bound on the number of words in DD, and set them to 1.
  - Insert random word in the index
- Output (Did,B) as the index of the document DD.

Secure Index

- □□□□□□□□□□□□(Tw,Did,B)

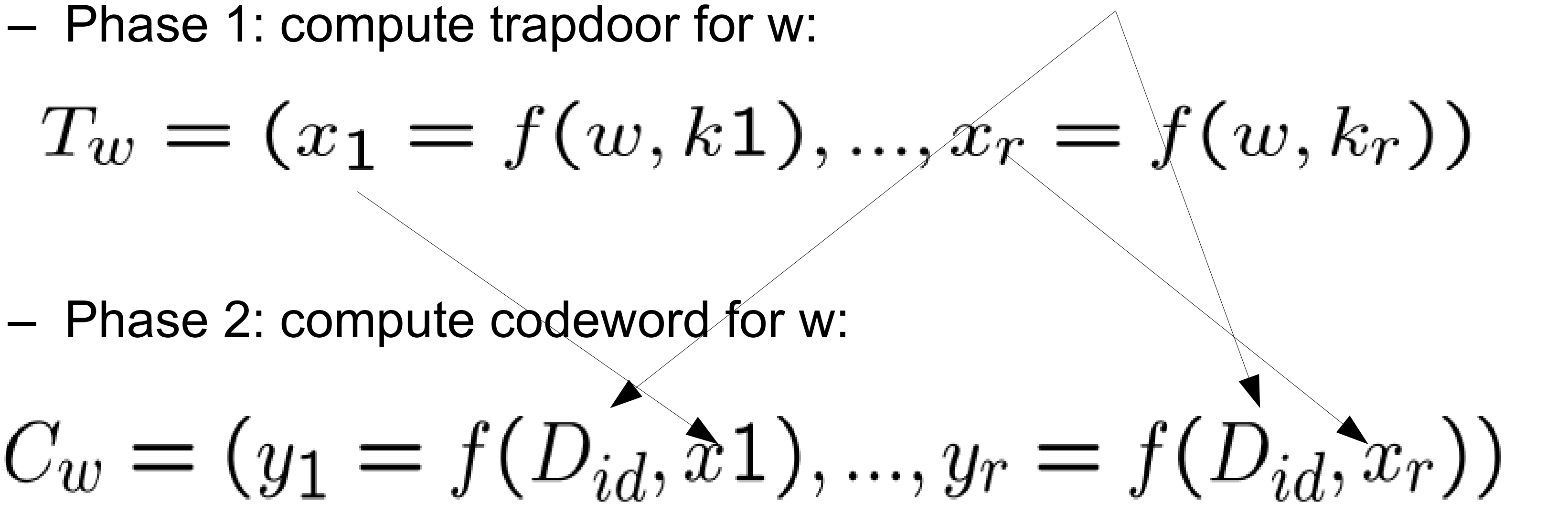
  Compute the rb bit codeword fB(Did,Tw) and split it into rr indices $(y_1,y_2,\ldots,y_r)$ of b bits each.

  Output $B_{y_1} \wedge B_{y_2} \wedge \cdots \wedge B_{y_r}$ where Bi denotes theii-th bit of the Bloom filter array B, and $\wedge$ denotes bitwise AND.

Secure Index

Build Index

For each word w in document Did:



- Phase 1: compute trapdoor for w:

$$T_w = (x_1 = f(w, k1), ..., x_r = f(w, k_r))$$

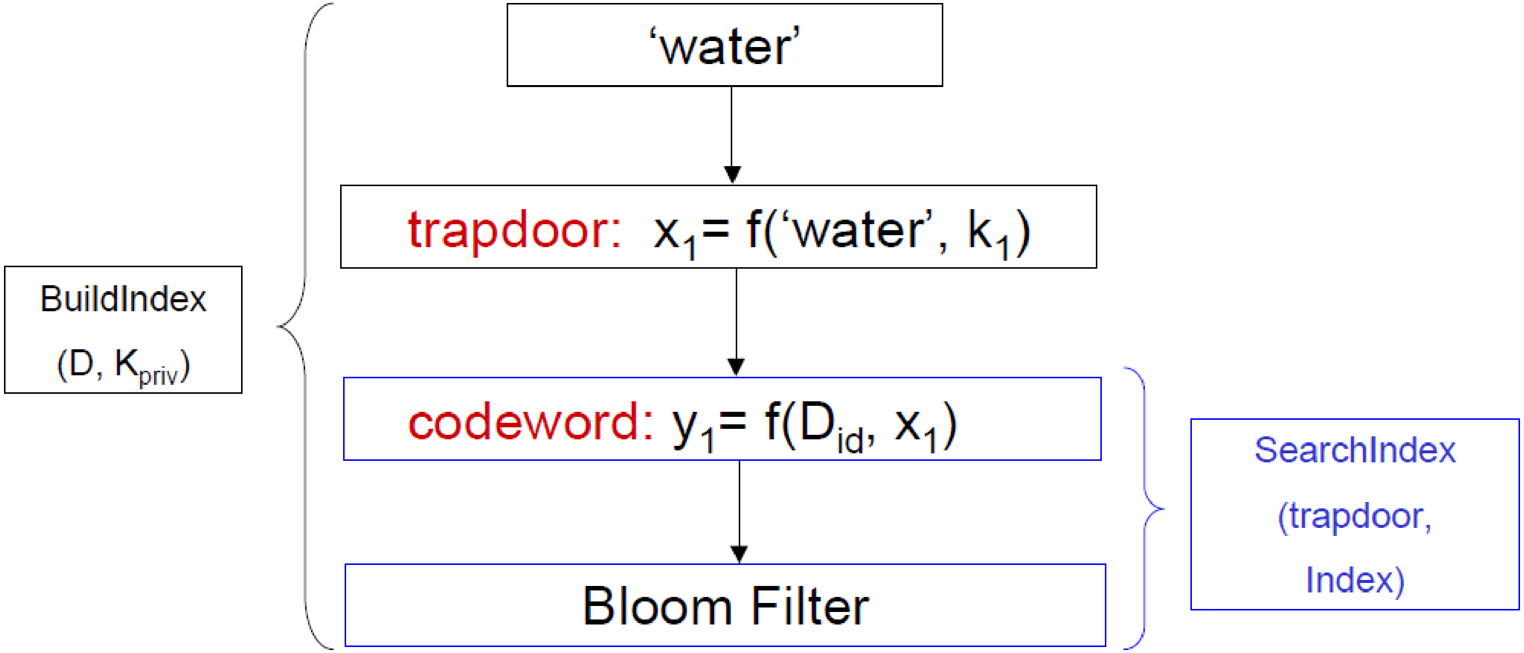
Distinct documents=
Distinct names =
Distinct values

- Phase 2: compute codeword for w:

$$C_w = (y_1 = f(D_{id}, x1), ..., y_r = f(D_{id}, x_r))$$

- Insert codeword into document’s Bloom filter

Secure Index Usage



‘water’

trapdoor: x_1= f(‘water’, k_1)

codeword: y_1= f(D_{id}, x_1)

Bloom Filter

BuildIndex

(D, K_{priv})

SearchIndex

(trapdoor,

Index)

Achieving IND-CKA

We need codewords because if the trapdoors Tx = f(x, k1), . . . , f(x, kq) are directly inserted into the Bloom filter index, the index is vulnerable to correlation attacks where the similarity of two documents is deduced by comparing Bloom filters indexes for overlaps, or lack thereof, of 1’s in the Bloom filter.

But, not enough to achieve IND-CKA because an adversary can win game easily

Solution:

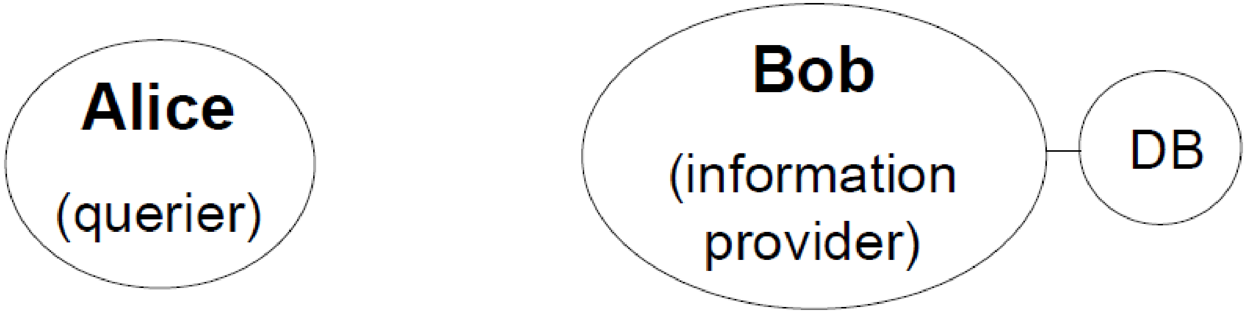
- u = upper bound on the number of words in Did
- v = number of distinct words in Did
- insert into index (u-v) random words so that two documents have the same number of tokens in their index even if they contain distinct number of tokend

But:

- u is computed relative to the encrypted document
- requires encryption of documents before building the index

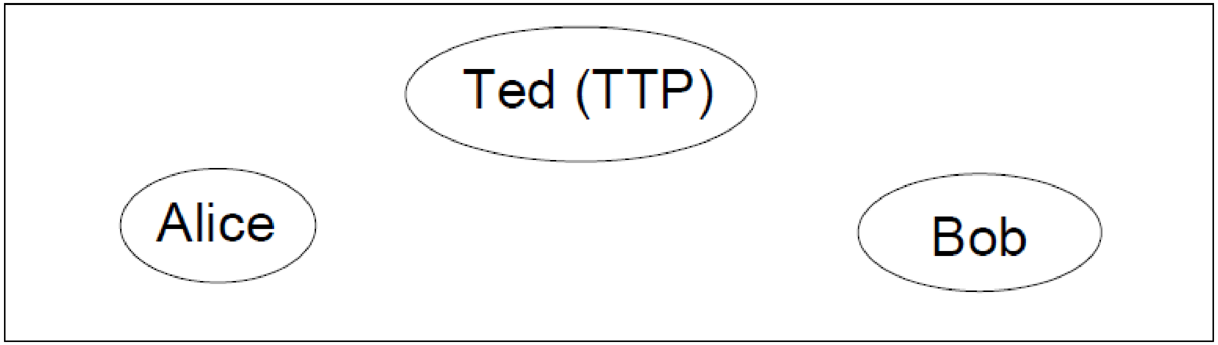
Privacy Enhanced Search

- Bellovin, Cheswick, “Privacy-enhanced Searches Using Encrypted Bloom Filters”
- Two parties want to share data selectively
- The parties don’t trust each other



Properties

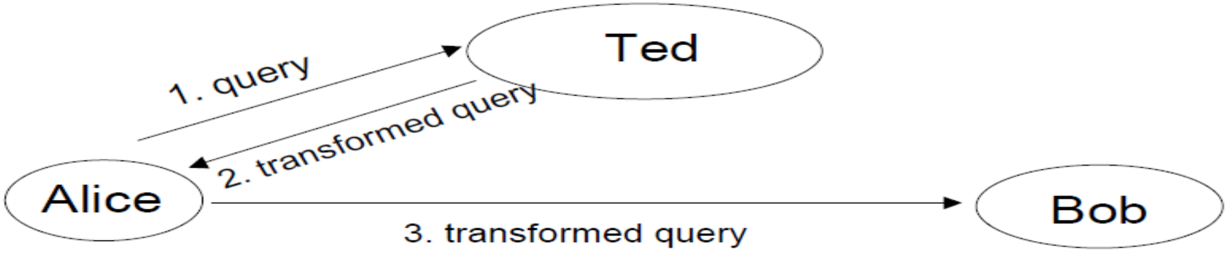
- Alice should be able to retrieve only documents matching valid queries
- Bob should not find contents of queries



- No third party should gain knowledge about queries or documents

The Basic Scheme

- Three-party negotiation between Alice, Bob and Ted to provision Ted with the transformation keys
- Bob prepares his DB as a collection of encrypted Bloom filters



Alice wants to search Bob DB without revealing any information

Group Chipers

- The set of all keys k forms an Abelian group under the operation composition of encryption

$$E_{k_1}(E_{k_2}(W)) = E_{k_1 \circ k_2}(W)$$

- Ted knows $r_{A,B} = k_B \circ k_A^{-1}$

- Given $E_{k_A}(W)$, Ted can compute

$$E_{r_{A,B}}(E_{k_A}(W)) = E_{r_{A,B} \circ k_A}(W) = E_{k_B}(W)$$

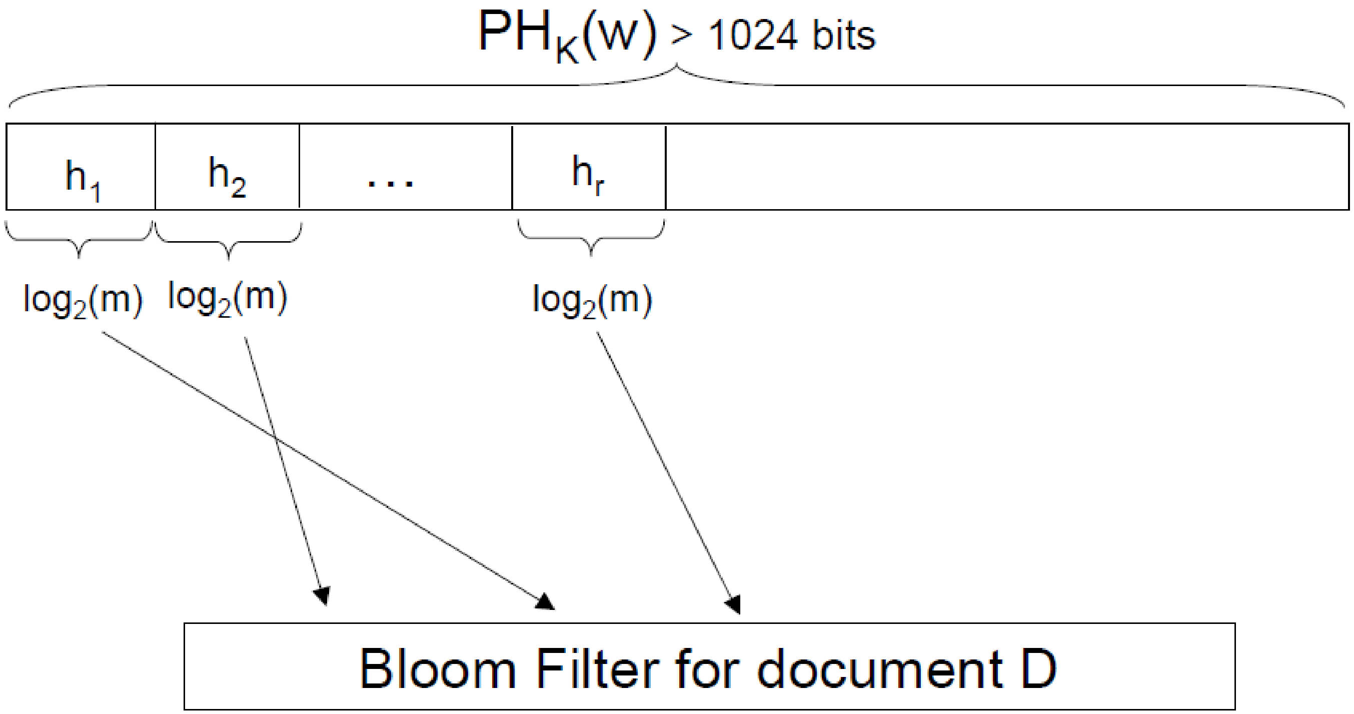
Group Chipers as Hash Functions

- Pohlig-Hellman encryption
  $PH_k(X) = X^k \text{ mod } p$
- Decrypt using d , such that $kd \equiv 1 \text{ mod } (p-1)$
- Since p > 1024 bits, use output of encryption as hash function
- Bob computes encrypted Bloom filters:
  - For each document D
    - For each word W in D
      - Compute $PH_{k_B}(W)$ and use chunks of $\lceil \log_2 m \rceil$ of it as hash functions to insert into Bloom filter for document D

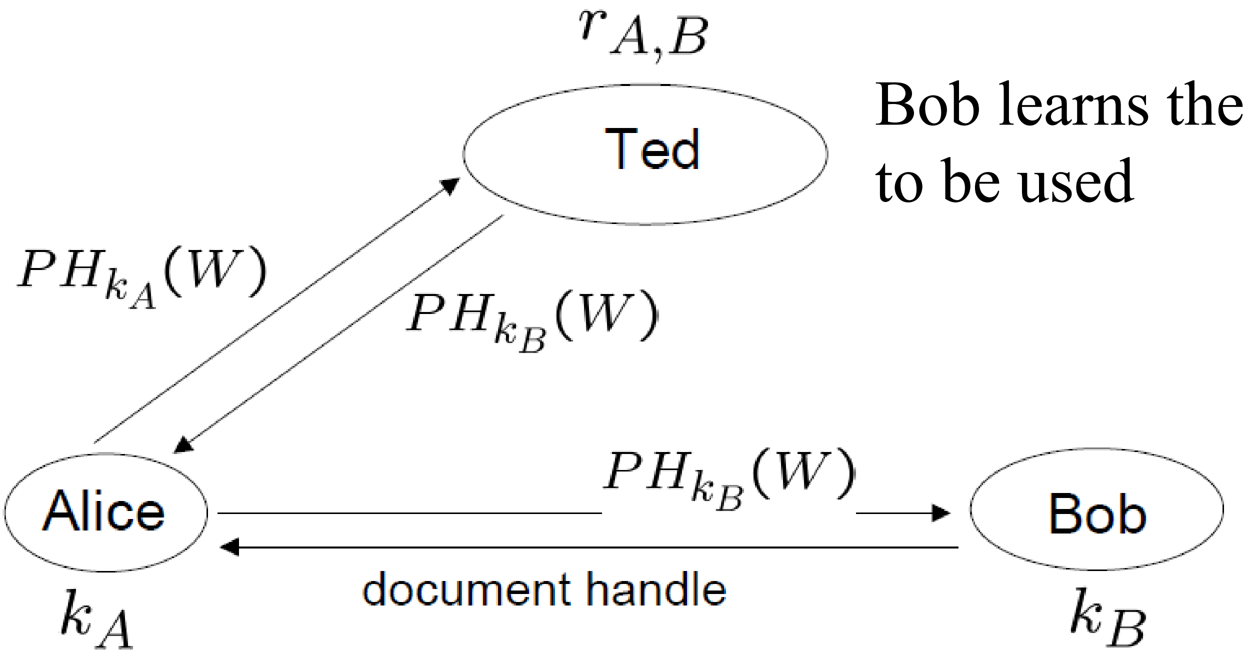
Group Chipers as Hash Functions

- Pohlig-Hellman encryption
  $PH_k(X) = X^k \bmod p$
- Decrypt using d , such that $kd \equiv 1 \bmod (p-1)$
- Since p > 1024 bits, use output of encryption as hash function
- Bob computes encrypted Bloom filters:
  - For each document D
    - For each word W in D
      - Compute $PH_{k_B}(W)$ and use chunks of $\lceil \log_2 m \rceil$ of it as hash functions to insert into Bloom filter for document D

Group Chipers as Hash Functions



The Basic Scheme Revised



Bob learns the hash values to be used

Bob uses $PH_{k_B}(W)$
to query the Bloom filter
of each document in the DB

Optimal Bloom Filter Parameters

If m is the size of the Bloom filter array and n is the number of unique words in the document then False Positive= $(1/2)^a$ where a = (ln 2)($\underline{m/n}$)

We can determine the optimal parameters as follow

1. Initially we choose FP. To this purpose considers that if FP increases, the server returns irrelevant documents that can be weeded out by mechanical scanning after being decrypted. Hence, FP is proportional to the communication overhead but it does not affect correctness
2. Compute q the number of pseudo-random function keys as −log2(FP).
3. Scan every document in the set and compute *nu* as the number of unique words. $n = \beta\, nu$ where β is a constant factor to allow for updates Only unique words in the document set are considered rather than all possible words
3. With q and n determined, the array size m is given by $m = nq/\ln 2$.

Secure Conjunctive Keyword Search Over Encrypted Data

Philippe Golle
Jessica Staddon
Palo Alto Research Center

Brent Waters
Princeton University

Motivating Scenario

Alice has a large amount of data

Which is private

Which she wants to access any time and from anywhere

Example: her emails

Alice stores her data on a remote server

Good connectivity

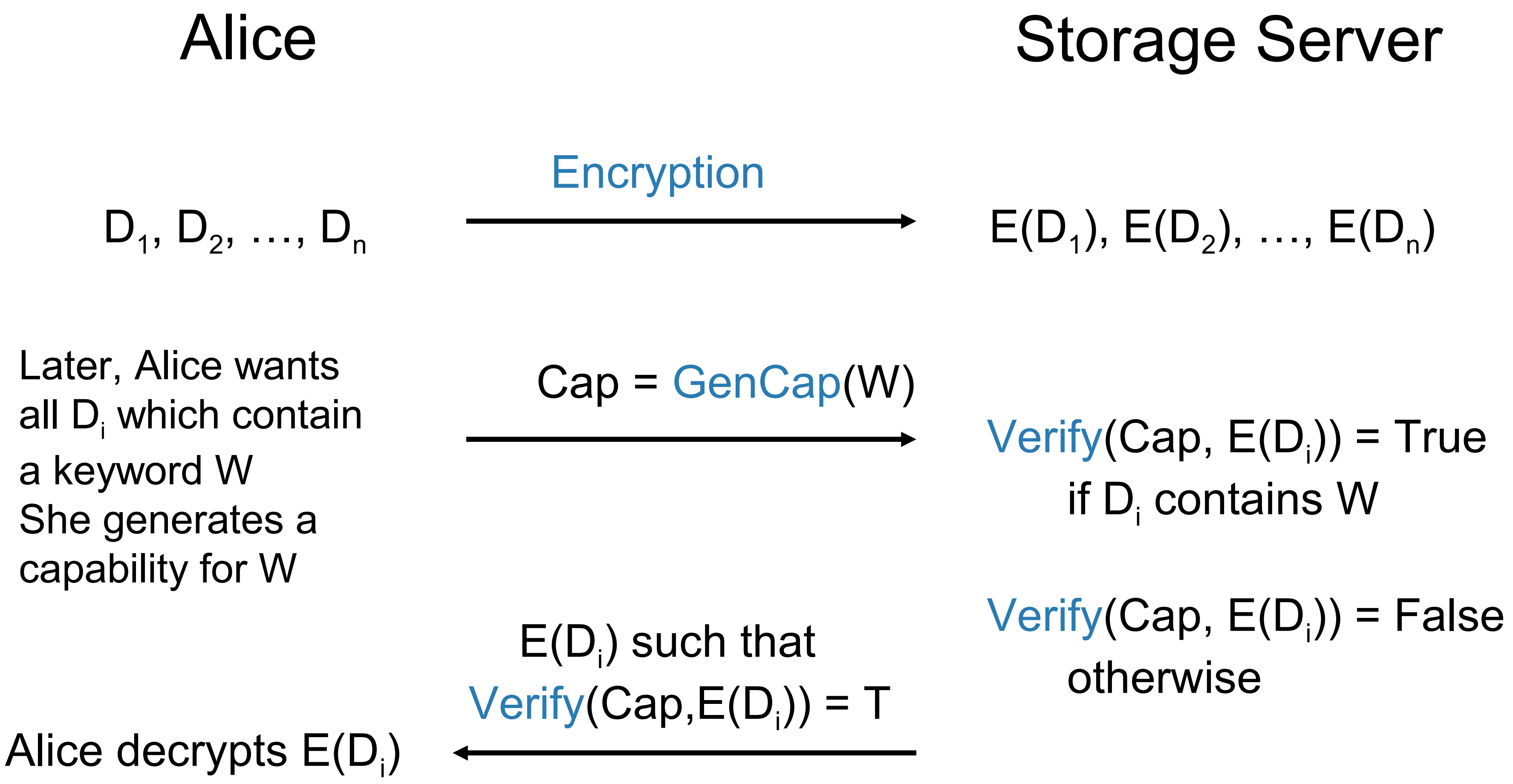
Low administration overhead

Cheaper cost of storage

But untrusted

1. Alice may not trust the server
   - Data must be stored encrypted
1. Alice wants ability to search her data
   - Keyword search: “All emails from Bob”
1. Alice wants *powerful*, *efficient* search
   - She wants to ask conjunctive queries
   - E.g. ask for “All emails from Bob AND received last Sunday”

Search on Encrypted Data



Alice

Storage Server

$D_1, D_2, \ldots, D_n$ $\xrightarrow{\text{Encryption}}$ $E(D_1), E(D_2), \ldots, E(D_n)$

Later, Alice wants all D_i which contain a keyword W
She generates a capability for W

Cap = GenCap(W)

Verify(Cap, $E(D_i)$) = True if D_i contains W

Verify(Cap, $E(D_i)$) = False otherwise

$E(D_i)$ such that Verify(Cap,$E(D_i)$) = T

Alice decrypts $E(D_i)$

Single Keyword Search

Solution of Song, Wagner & Perrig

- [2000 IEEE Security and Privacy]
- Define a security model for single keyword search
- Propose provably secure protocols

Limitations

- Limited to queries for a *single* keyword
- Can’t do boolean combinations of queries
  - Example: “emails from Bob AND (received last week OR urgent)”

We focus on conjunctive queries

- Documents D_i which contains keywords W_1 and W_2 … and W_n
- More restrictive than full boolean combinations
- But powerful enough! (see search engines)

Possible Approaches to Conjunctive Queries

Alice wants all documents with keywords W_1 and W_2 … and W_n

Computing set intersections

She generates capabilities **Cap_1** , **Cap_2** … **Cap_n** for **W_1** ,**W_2** … **W_n**

Storage server finds sets of documents **S_1** ,**S_2** … **S_n** that match the capabilities **Cap_1** , **Cap_2** … **Cap_n** and returns the intersection $\cap S_i$

Problem

Server learns a lot of extra information on top of result of conjunctive query

E.g. "Emails from Bob & Secret"
"Emails from President & Non-secret"
} "Emails from President & Secret"

•Defining Meta-Keywords

- Define a meta-keyword for every possible conjunction of keywords
- E.g. "Email from Bob & Secret" → meta-keyword "From Bob || Secret"
- Meta-keywords are associated with documents like regular keywords
- Problem: with m keywords, we must define 2^m meta-keywords to allow for all possible conjunctive queries.

Possible Approaches to Conjunctive Queries

Alice wants all documents with keywords W_1 and W_2 … and W_n

Computing set intersections

She generates capabilities $\mathbf{Cap}_1$, $\mathbf{Cap}_2$ … $\mathbf{Cap}_n$ for $\mathbf{W}_1$,$\mathbf{W}_2$ … $\mathbf{W}_n$

Storage server finds sets of documents $\mathbf{S}_1$,$\mathbf{S}_2$ … $\mathbf{S}_n$ that match the capabilities $\mathbf{Cap}_1$, $\mathbf{Cap}_2$ … $\mathbf{Cap}_n$ and returns the intersection $\cap \mathbf{S}_i$

Problem

Server learns a lot of extra information on top of result of conjunctive query

E.g. “Emails from Bob & Secret”
“Emails from President & Non-secret”

“Emails from President & Secret”

•Defining Meta-Keywords

- Define a meta-keyword for every possible conjunction of keywords
- E.g. “Email from Bob & Secret” → meta-keyword “From Bob || Secret”
- Meta-keywords are associated with documents like regular keywords
- Problem: with m keywords, we must define 2^m meta-keywords to allow for all possible conjunctive queries.

Outline

Model and definitions

- Model of documents
- Define conjunctive keyword search
- Security model for conjunctive queries

Basic protocol

- Size of capabilities is linear in the number of documents (**n**)

Amortized Protocol

- Size of capabilities is linear in **n** but linear cost is incurred *offline* before the query is asked
- Standard security assumptions

Constant-size Protocol

- Size of capabilities is constant in **n**
- But relies on new hardness assumption

Model of Documents

- We assume structured documents where keywords are organized by fields

m fields

n docs

	From	To	Date	Status
D_1	Alice	Bob	06/01/2004	Urgent
D_2	Alice	Charlie	05/28/2004	Secret
	…	…	…	…
D_n	Dave	Alice	06/04/2004	Non-urgent

The documents are associated with the rows $D_i = (W_{i,\,1}, \ldots, W_{i,\,m})$

Conjunctive Search on Encrypted Data

Encryption: same as before

Generating a Capability

Before: Cap = GenCap(W)

Now: Cap = Gencap(j_1, …,j_t, W_{j1}, …, W_{jt}) where

j_1, …,j_t are t field indices

W_{j1}, …, W_{jt} are t keywords

Example: GenCap(“From, Date”, “Bob, 06/04/2004”)

Verifying a capability

Let Cap = Gencap(j_1, …,j_t, W_{j1}, …, W_{jt})

Verify (Cap, D) returns True if

D has keyword W_{j1} in field j_1

…

D has keyword W_{jt} in field j_t

Security Model

Informally "capabilities reveal no more information than they should"

In particular, capabilities can't be combined to create new ones

$$\text{GenCap}(j_1, j_2, W_1, W_2)\ \&\ \text{GenCap}(j_1, W_1) \rightarrow \text{GenCap}(j_2, W_2)$$

Except for "trivial" set-theoretic combinations

$$\text{GenCap}(j_1, j_2, W_1, W_2)\ \&\ \text{GenCap}(j_1, W_1) \rightarrow \text{GenCap}(j_1, j_2, W_1, \neg W_2)$$

Formally: we define the following game with an adversary A

A calls Encrypt and GenCap

A chooses two documents D_0 and D_1 and receives $E(D_b)$

A again calls Encrypt and GenCap

A guesses the bit b

A wins if

A guesses b correctly = guesses whether has received D_0 or D_1

None of the capabilities given in Steps 1 and 3 distinguish D_0 from D_1

A protocol is secure if A wins with prob non-negligibly > 1/2

Outline

Model and definitions

Model of documents

Define conjunctive keyword search

Security model for conjunctive queries

Basic protocol

Size of capabilities is linear in the number of documents (**n**)

Amortized Protocol

Size of capabilities is linear in **n** but linear cost is incurred *offline* before the query is asked

Standard security assumptions

Constant-size Protocol

Size of capabilities is constant in **n**

But relies on new hardness assumption

Basic Protocol

- Parameters
  - A group G of order q in which Decision Diffie Hellman is hard and a generator g of G
  - A keyed hash function f_k (Alice has the secret key k)
  - A hash function h

- Encrypting $D_i = (W_{i,1}, \ldots, W_{i,m})$
  - Let $V_{i,j} = f_k(W_{i,j})$
  - Let a_i be a random value

$$E(D_i) = \left(g^{a_i V_{i,1}}, g^{a_i V_{i,2}}, \ldots, g^{a_i V_{i,m}}\right)$$

- Intuition
  - Alice commits to the encrypted keywords
  - The a_i's ensure that commitments are different for each document
    - Same keyword looks different in different documents
  - The commitments are malleable *within* the same document
    - Product of commitments = commitment to sum
    - Commitments are NOT malleable *across* different documents

Malleable

- An encryption algorithm is malleable if an adversary can transform a ciphertext into another ciphertext which decrypts to a related plaintext.
- Given an encryption of a plaintext *m*, it is possible to generate another ciphertext which decrypts to *f*(*m*), for a known function *f*, without necessarily knowing or learning *m*.
- Malleability is often an undesirable property in a general-purpose cryptosystem, since it allows an attacker to modify the contents of a message.
- A bank uses a stream cipher to hide its financial information, and a user sends an encrypted message containing,

  “TRANSFER $0000100.00 TO ACCOUNT #199."

  An attacker that

  - can modify the message on the wire,
  - guess the format of the unencrypted message,
  - could to change the amount of the transaction, or the recipient

Basic Protocol (Continued)

$$E(D_i) = \left(g^{a_i V_{i,1}}, g^{a_i V_{i,2}}, \ldots, g^{a_i V_{i,m}}\right)$$

- Generating a capability Gencap(j_1, …,j_t, W_{j1}, …, W_{jt})

$$s = \sum_{w=1}^{t} f_k(W_{j_w}) \qquad Cap = \left(h(g^{a_1 s}), \ldots, h(g^{a_n s})\right)$$

- Verifying a capability

$$h\left(\prod_{w=1}^{t} g^{a_i V_{i,j_w}}\right) = h\left(g^{a_i s}\right)$$

- Intuition
  - The commitments are malleable
  - The capability that allows the *verification* of commitments is not malleable

Basic Protocol: Example

	From	To	Status
√	$g^{a_1 f_k(Alice)}$	$g^{a_1 f_k(Bob)}$	$g^{a_1 f_k(Urgent)}$
X	$g^{a_2 f_k(Alice)}$	$g^{a_2 f_k(Dave)}$	$g^{a_2 f_k(Secret)}$

Capability for emails from Alice to Bob is

- Let s = f_k (alice) + f_k (Bob)
- $Cap = \left(h(g^{a_1 s}), h(g^{a_2 s})\right)$

$$g^{a_1(f_k(Alice)+f_k(Bob))}$$

$$g^{a_2(f_k(Alice)+f_k(Dave))}$$

$$g^{a_1 f_k(Alice)} \; g^{a_1 f_k(Bob)}$$

$$g^{a_2 f_k(Alice)} \; g^{a_2 f_k(Dave)}$$

$$h\left(g^{a_1(f_k(Alice)+f_k(Bob))}\right) = h(g^{a_1 s})$$

$$h\left(g^{a_2(f_k(Alice)+f_k(Dave))}\right) \neq h(g^{a_2 s})$$

Problem:
the size of capabilities is linear in n

Amortized Protocol

- Parameters: unchanged

- Encrypting a document $D_i = (W_{i,1}, \ldots, W_{i,m})$
  - Let $V_{i,j} = f_k(W_{i,j})$
  - Let a_i be a random value

$$E(D_i) = (\boxed{g^{a_i}}, g^{a_i V_{i,1}}, g^{a_i V_{i,2}}, \ldots, g^{a_i V_{i,m}})$$

Further value

Amortized Protocol (Continued)

- Generating a capability Gencap($j_1, \ldots, j_t, W_{j1}, \ldots, W_{jt}$)
  - Pick a random value r
  - A proto-capability → $Q = \left(h(g^{a_1 r}), h(g^{a_2 r}), \ldots, h(g^{a_n r})\right)$
  - The query part → $C = r + \sum_{w=1}^{t} f_k(W_{j_w})$
- Intuition
  - In the basic protocol, we had $s = \sum_{w=1}^{t} f_k(W_{j_w})$
  - Now, the proto-capability is independent of the query
    - It can be transmitted "offline" before the query
  - The random value r ties the proto-capability to the query

- Verification: compute $R_i = g^{a_i C} g^{-a_i \sum_{w=1}^{t} V_{i,j_w}}$

return True if $h(R_i) = h(g^{a_i r})$ and False otherwise

Constant Protocol

- Parameters
  - Two group G_1 and G_2 of order q
  - An admissible bilinear map e : G_1 X G_1 → G_2
  - A generator g of G_1
  - A keyed hash function f_k

- Encrypting a document D = (W_1, …, W_m)
  - Let $V_i = f_k(W_i)$
  - Let $R_{i,j}$ be values chosen uniformly independently at random
  - Let

$$E(D) = g^{a_i}, \left(g^{a_i(V_{i,1}+R_{i,1})}, \ldots, g^{a_i(V_{i,m}+R_{i,m})}\right), \left(g^{a_i\alpha R_{i,1}}, \ldots, g^{a_i\alpha R_{i,m}}\right)$$

Constant Protocol (Continued)

- Generating a capability Gencap(j_1, …,j_t, W_{j1}, …, W_{jt})

$$Cap = \left(g^{\alpha r}, g^{\alpha r\left(\sum_{w=1}^{t} f_k(W_{j_w})\right)}, g^{r}, j_1,\ldots, j_t \right)$$

- Verification

$$e\left(g^{\alpha r\left(\sum_{w=1}^{t} f_k(W_{j_w})\right)}, g^{a_i} \right) = \prod_{k=1}^{t} \left(\frac{e\left(g^{\alpha r}, g^{a_i(V_{i,j_k}+R_{i,j_k})}\right)}{e\left(g^{r}, g^{a_i \alpha R_{i,j_k}}\right)} \right)$$

Conclusion

Our contributions:

Define security model for conjunctive keyword search on encrypted data and propose 3 protocols

1. Linear communication cost
2. Amortized linear communication cost

    Standard hardness assumption

1. Constant cost

    Uses new hardness assumption

Future work

Extend to full boolean queries

The OR operator appears tricky…

Indistinguishability of capabilities

Hide the fields that are being searched on

Encrypting a function

In some cases there is a natural encryption of the function

As an example, in the case of the solution of a LP problem we can

a) add some random value to the matrices that codifies the problem

b) pass the problem to a SaaS service that solves it

c) remove the random values from the solution

This exploits the linearity of the considered problem

Not general