Searchable Encryption: Part 2

CS 5158/6058 Data Security and Privacy

Spring 2018

Instructor: Boyang Wang

Inverted Index

- Inverted Index (built from keyword-file matrix)
  - Only keeps associated file identifiers (pointers or memory addresses) for each keyword

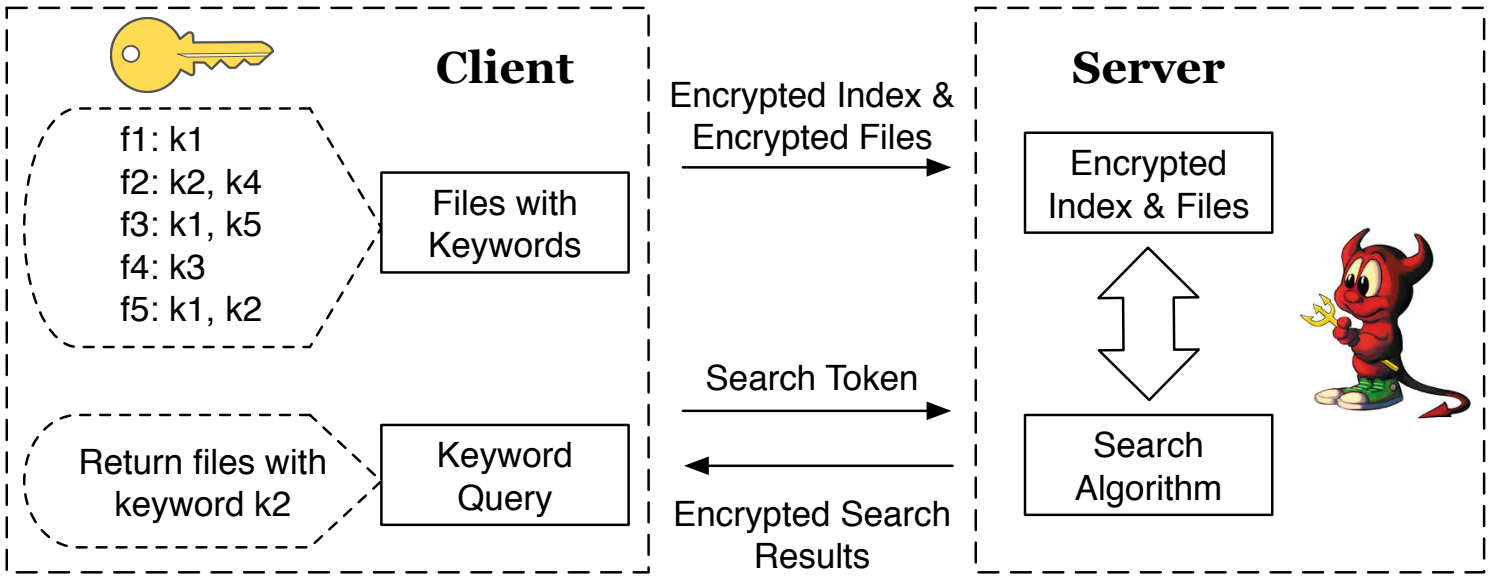
	f1	f2	f3	f4	f5
k1	1	0	0	0	1
k2	0	1	0	1	1
k3	1	0	0	0	0
k4	0	0	1	0	0

Inverted Index

k1: id1, id5
k2: id2, id4, id5
k3: id1
k4: id3

Searchable Encryption

- Client builds an encrypted index, and encrypts files with AES
- Server can still search, but does not learn keyword query or returned files



- KeyGen: given a security parameter λ, client outputs a secret key sk.
- Enc: given a set of files $f_1, ..., f_n$ and a secret key sk, client outputs an encrypted index Γ and a set of encrypted files $c_1, ..., c_n$.
- Token: given a keyword k and a secret key sk, client outputs a token tk.
- Query: given an encrypted index Γ, a set of encrypted files $c_1, ..., c_n$ and a token tk, server returns all the encrypted files that associated with tk.

Building Blocks

- Pseudo Random Function (PRF)
  - Deterministic encryption
    - E.g., $PRF_k(m_1) = w_1$, $PRF_k(m_2) = w_2$
    - If $m_1 = m_2$, then $w_1 = w_2$
- Advanced Encryption Standard (AES)
  - AES-CBC: probabilistic encryption
- Inverted Index with non-crypto hash functions

Searchable Encryption

- Client generates two keys based on a security parameter
  - One key for PRF to encrypt index
  - One key for AES to encrypt files
    - Can also generate multiple AES keys, s.t. each file is encrypted with a different AES key

Searchable Encryption

- Client builds an encrypted index
  1. Pre-process files, extract keywords
  2. Encrypts keywords (with PRF)
  3. Builds an encrypted index
  4. Encrypts files (with AES)
  5. Uploads encrypted index and encrypted files

- Client generates a search token
  - Encrypt a keyword query (with PRF)

Searchable Encryption

- Server searches with this token on encrypted index
  - Still same search algo in matrix/inverted index
  - Return matched file identifiers/addresses
  - Return corresponding encrypted files

- Why search over encrypted index is possible?
  - Equality checking —> either Yes or No
    - Check a query is equal to an item using PRF
    - Does not need to know what it is.

Example of Enc

- Example: there are 3 files
  - `f1={uc is in ohio.}`
  - `f2={ohio has cs.}`
  - `f3={uc is uc.}`

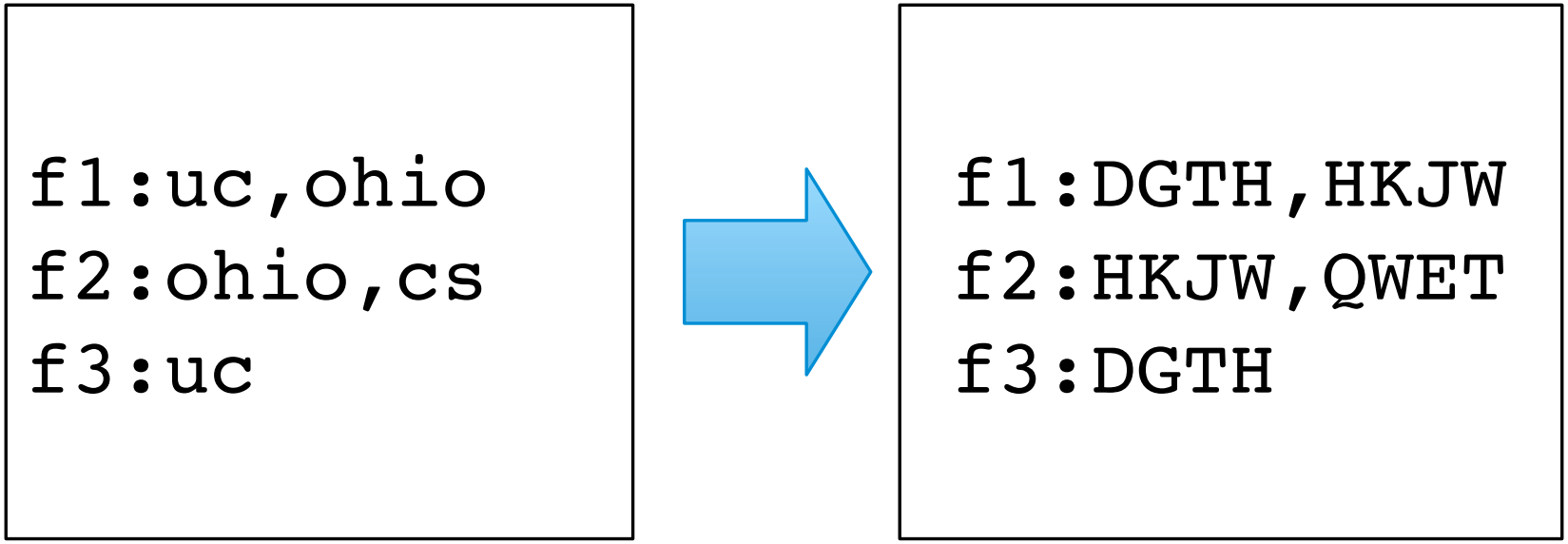
- Extract keywords from 3 files
  - `f1: uc, ohio;`
  - `f2: ohio, cs;`
  - `f3: uc;`

Example of Enc

- Encrypt each keyword with Pseudo Random Function
  - PRF is deterministic, can be used in search

$PRF_k(\texttt{uc}) = \texttt{DGTH}$; $PRF_k(\texttt{ohio}) = \texttt{HKJW}$;

$PRF_k(\texttt{cs}) = \texttt{QWET}$



Example of Enc

$PRF_k(\texttt{uc}) = \texttt{DGTH}$; $PRF_k(\texttt{ohio}) = \texttt{HKJW}$;

$PRF_k(\texttt{cs}) = \texttt{QWET}$

- Build an encrypted matrix

```
f1:DGTH,HKJW
f2:HKJW,QWET
f3:DGTH
```

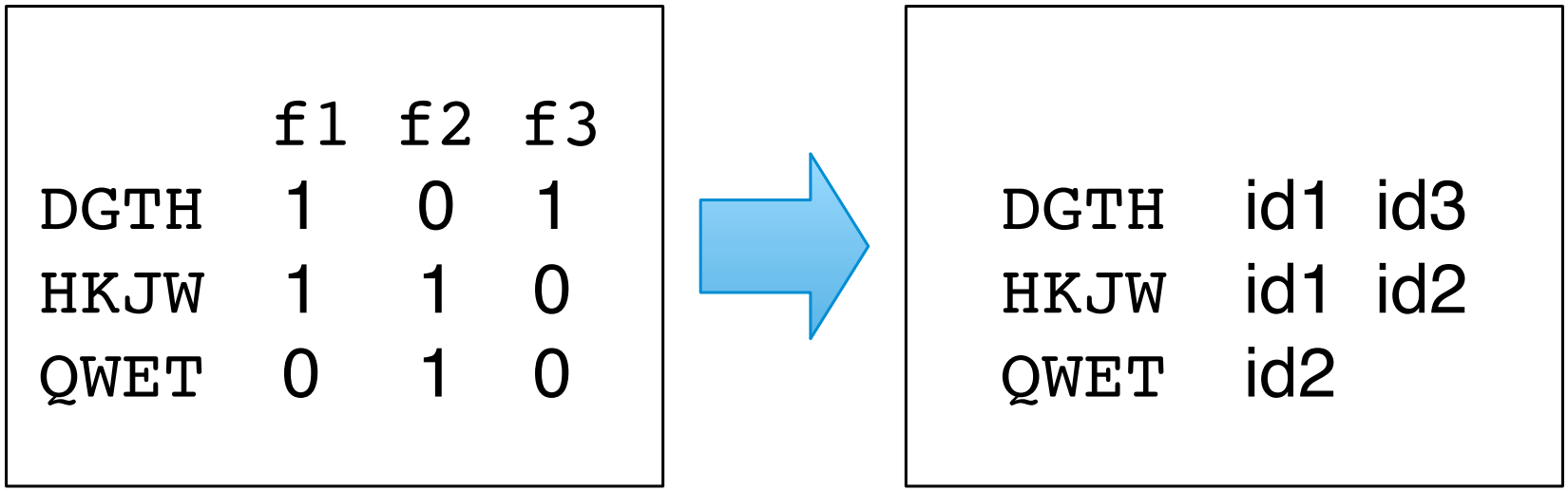
	f1	f2	f3
DGTH	1	0	1
HKJW	1	1	0
QWET	0	1	0

Example of Enc

$PRF_k(\texttt{uc}) = \texttt{DGTH}$; $PRF_k(\texttt{ohio}) = \texttt{HKJW}$;

$PRF_k(\texttt{cs}) = \texttt{QWET}$

- Build an encrypted inverted index

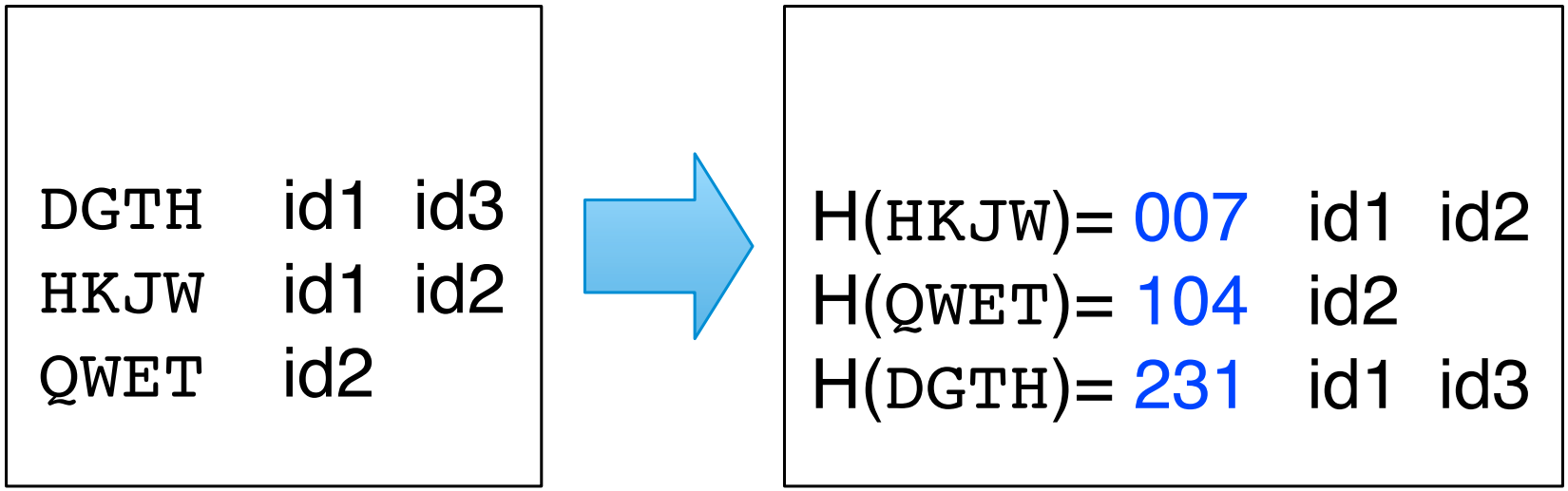


Example of Enc

$PRF_k(\texttt{uc}) = \texttt{DGTH}$; $PRF_k(\texttt{ohio}) = \texttt{HKJW}$;

$PRF_k(\texttt{cs}) = \texttt{QWET}$

- Build an encrypted inverted index with hash table

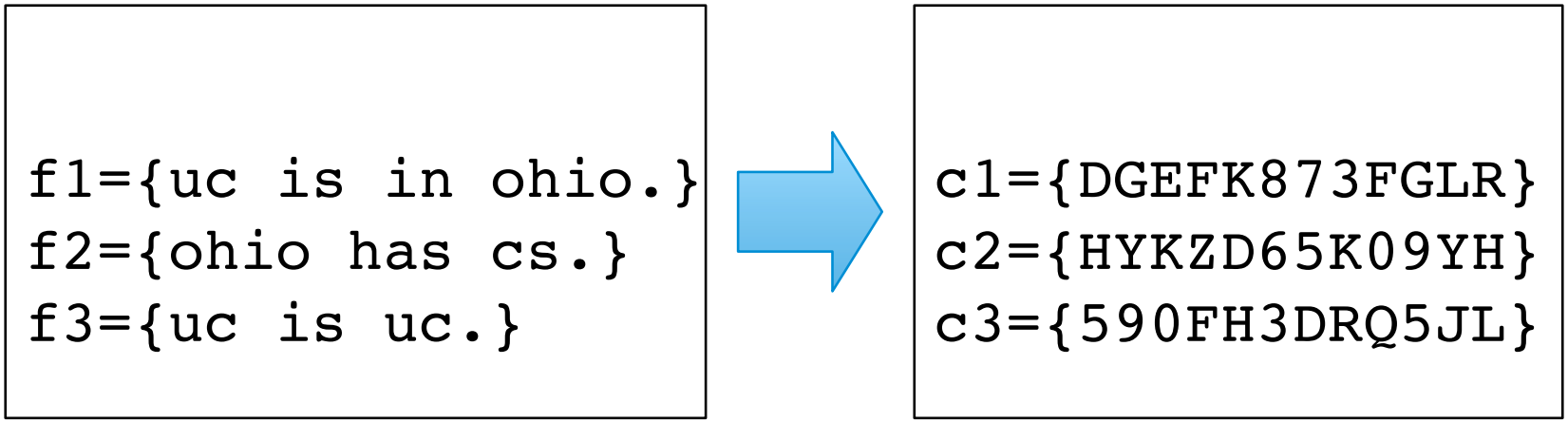


Example of Enc

$AES.Enc_{sk}(\mathtt{f1}) = \mathtt{c1}$; $AES.Enc_{sk}(\mathtt{f2}) = \mathtt{c2}$;

$AES.Enc_{sk}(\mathtt{f3}) = \mathtt{c3}$

- Encrypt files f1, f2, f3 with AES



Example of Enc

- Client uploads encrypted index and encrypted files

Encrypted Index			
H(HKJW)=	007	id1	id2
H(QWET)=	104	id2	
H(DGTH)=	231	id1	id3

Encrypted Files
c1={DGEFK873FGLR}
c2={HYKZD65K09YH}
c3={590FH3DRQ5JL}

- Server does not know keywords or files

Comparison

- Server's review before & after using SE

Server's view (in plaintext)

H(cs)= 027	id2	f1={uc is in ohio.}
H(ohio)= 041	id1 id2	f2={ohio has cs.}
H(uc)= 098	id1 id3	f3={uc is uc.}

Server's view (With SE)

H(HKJW)= 007	id1 id2	c1={DGEFK873FGLR}
H(QWET)= 104	id2	c2={HYKZD65K09YH}
H(DGTH)= 231	id1 id3	c3={590FH3DRQ5JL}

Example of Token & Query

- Client wants to search keyword **ohio**
  - Encrypts **ohio** with PRF, PRF_k(**ohio**) = **HKJW**
  - Submits **HKJW** to server

- Server computes H(**HKJW**)=007, finds id1 & id2, and returns **c1** & **c2**

H(HKJW)=	007	id1	id2
H(QWET)=	104	id2	
H(DGTH)=	231	id1	id3

c1={DGEFK873FGLR}
c2={HYKZD65K09YH}
c3={590FH3DRQ5JL}

Example of Token & Query

- Client decrypts **`c1`** and **`c2`**, and obtains **`f1`** and **`f2`**

  $AES.Dec_{sk}$(**`c1`**) = **`f1`**; $AES.Dec_{sk}$(**`c2`**) = **`f2`**;

  `f1={uc is in ohio.}`

  `f2={ohio has cs.}`

- Server only knows client searched **`HKJW`**

- Client receives the same files as it searches in plaintext: submits **`ohio`**, receives **`f1`** and **`f2`**

Build It Yourself

- Practice: 3 files
  - `f1={dayton is in usa.}`
  - `f2={usa has data.}`
  - `f3={dayton is dayton.}`

- Extract keywords from 3 files
  - `f1: ???, ???;`
  - `f2: ???, ???;`
  - `f3: ???;`

Build It Yourself

- Practice: 3 files
  - `f1={dayton is in usa.}`
  - `f2={usa has data.}`
  - `f3={dayton is dayton.}`

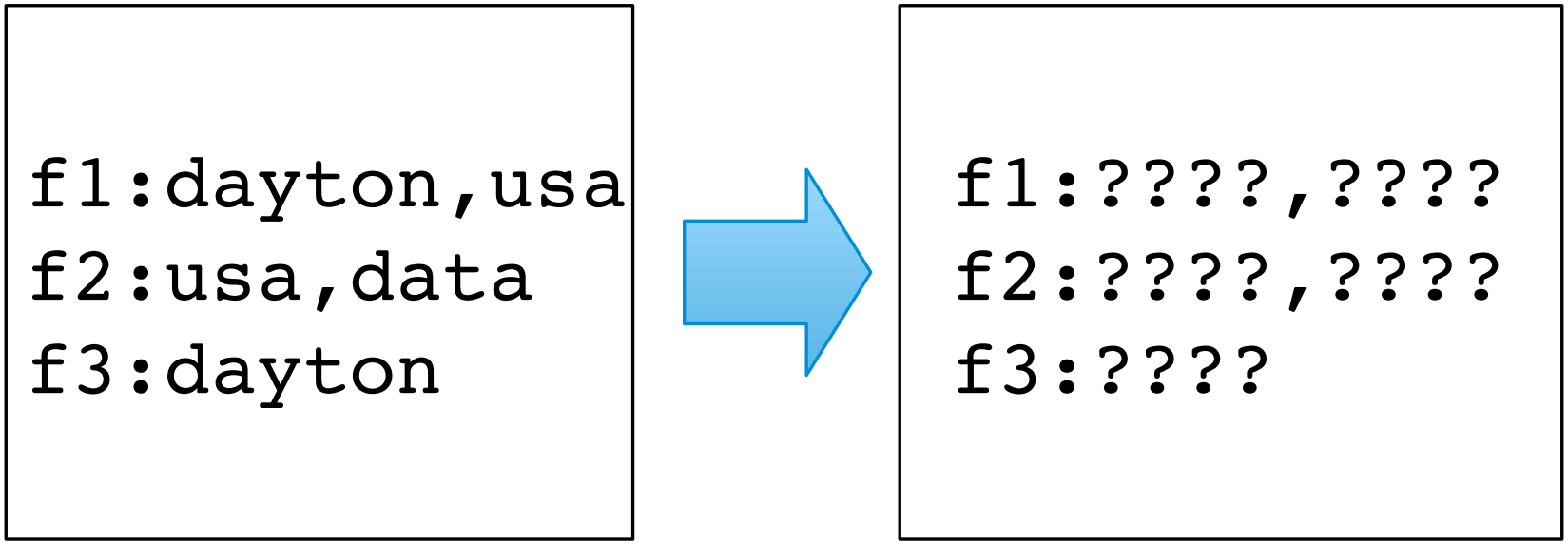
- Extract 3 keywords from 3 files
  - `f1: dayton, usa;`
  - `f2: usa, data;`
  - `f3: dayton;`

Build It Yourself

- Encrypt each keyword with Pseudo Random Function

$PRF_k(\texttt{dayton}) = \texttt{TJKH}$; $PRF_k(\texttt{usa}) = \texttt{HAQP}$;

$PRF_k(\texttt{data}) = \texttt{QLOT}$

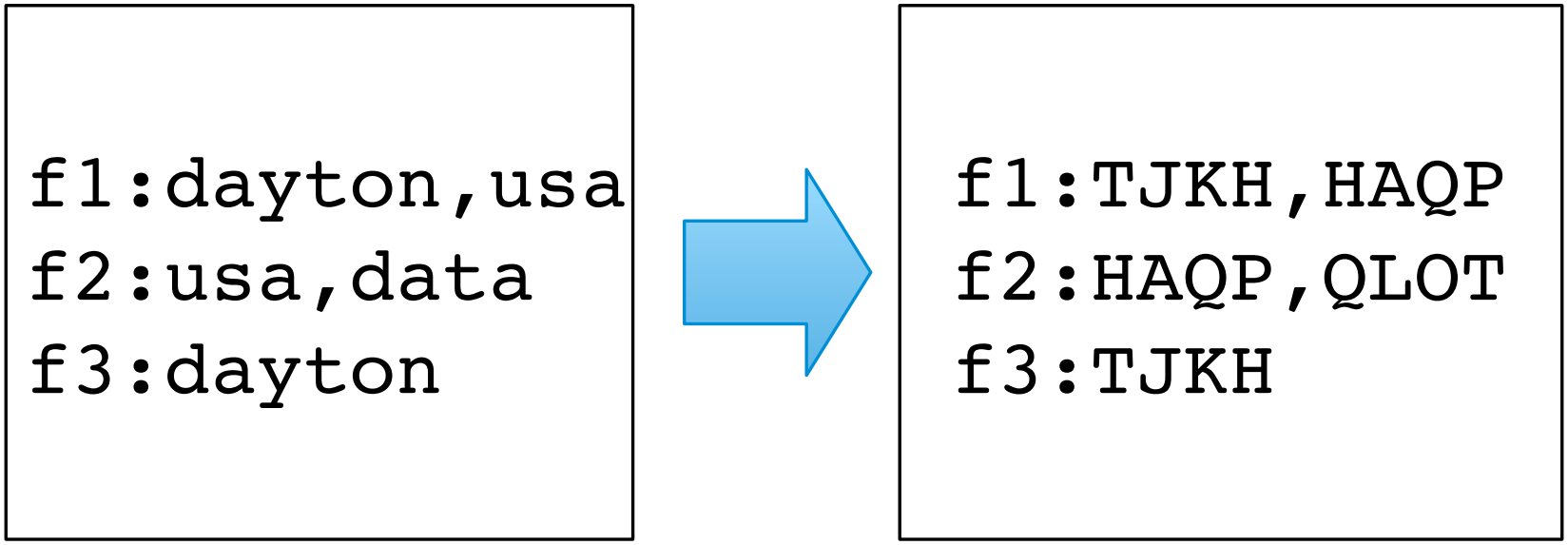


Build It Yourself

- Encrypt each keyword with Pseudo Random Function

$PRF_k(\texttt{dayton}) = \texttt{TJKH}$; $PRF_k(\texttt{usa}) = \texttt{HAQP}$;

$PRF_k(\texttt{data}) = \texttt{QLOT}$



Build It Yourself

$PRF_k(\texttt{dayton}) = \texttt{TJKH}$; $PRF_k(\texttt{usa}) = \texttt{HAQP}$;

$PRF_k(\texttt{data}) = \texttt{QLOT}$

- Build an encrypted matrix

```
f1:TJKH,HAOP
f2:HAOP,QLOT
f3:TJKH
```

	f1	f2	f3
TJKH	?	?	?
HAOP	?	?	?
QLOT	?	?	?

Build It Yourself

$PRF_k(\texttt{dayton}) = \texttt{TJKH}$; $PRF_k(\texttt{usa}) = \texttt{HAQP}$;

$PRF_k(\texttt{data}) = \texttt{QLOT}$

- Build an encrypted matrix

```
f1:TJKH,HAOP
f2:HAOP,QLOT
f3:TJKH
```

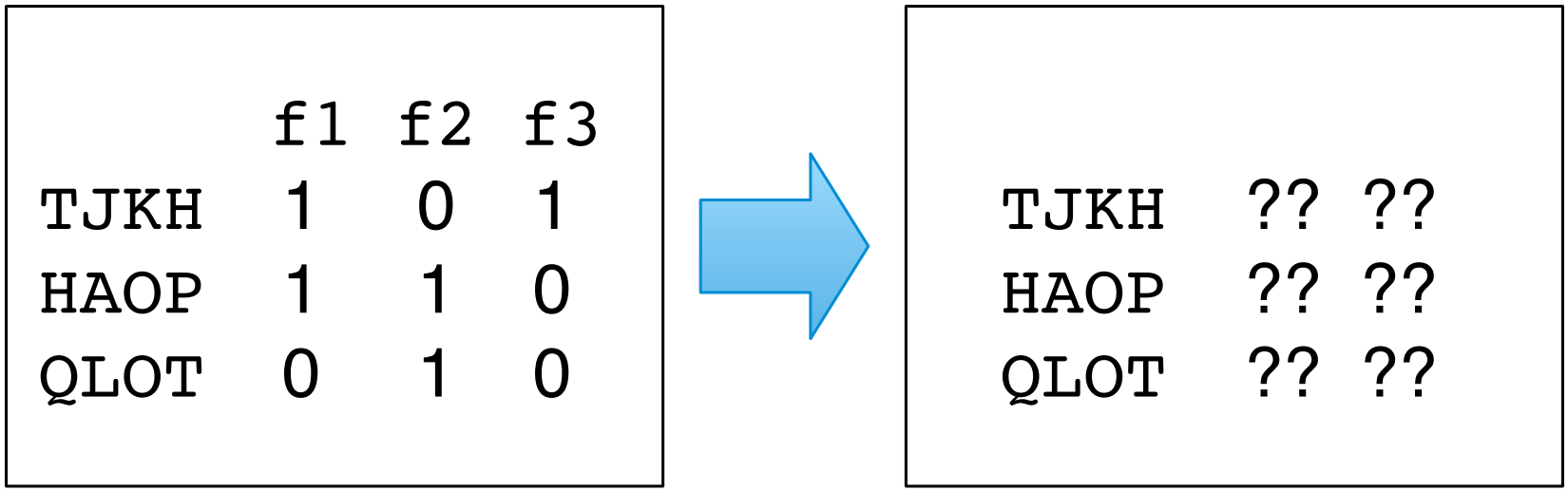
	f1	f2	f3
TJKH	1	0	1
HAOP	1	1	0
QLOT	0	1	0

Build It Yourself

$PRF_k(\texttt{dayton}) = \texttt{TJKH}$; $PRF_k(\texttt{usa}) = \texttt{HAQP}$;

$PRF_k(\texttt{data}) = \texttt{QLOT}$

- Build an encrypted inverted index

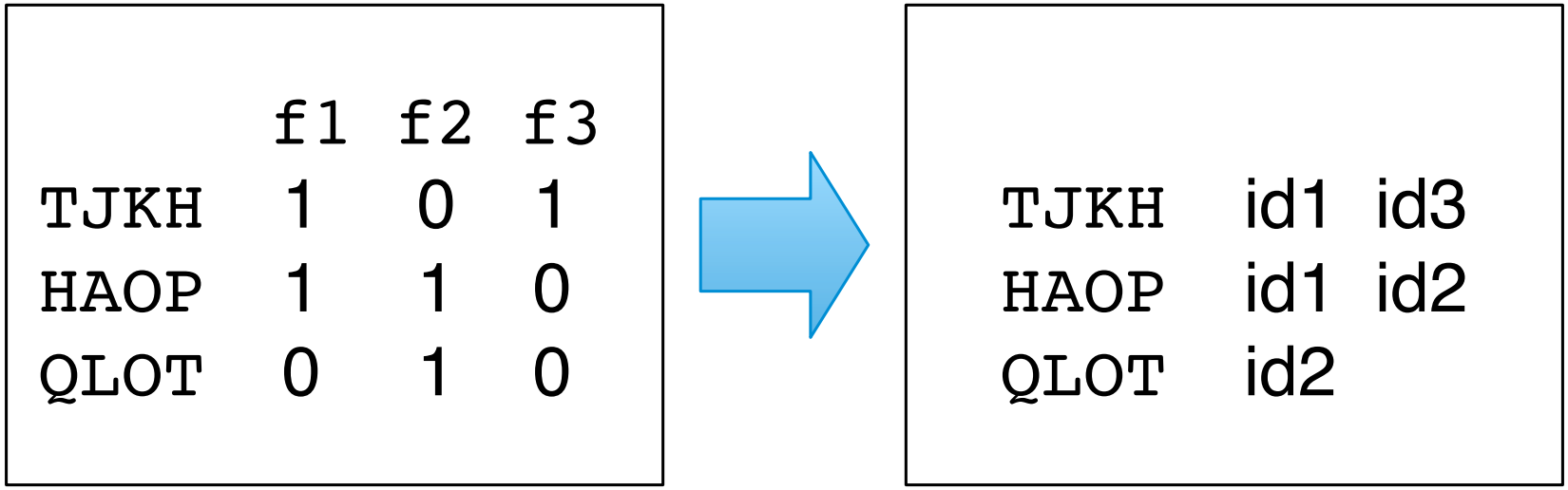


Build It Yourself

$PRF_k(\texttt{dayton}) = \texttt{TJKH}$; $PRF_k(\texttt{usa}) = \texttt{HAQP}$;

$PRF_k(\texttt{data}) = \texttt{QLOT}$

- Build an encrypted inverted index

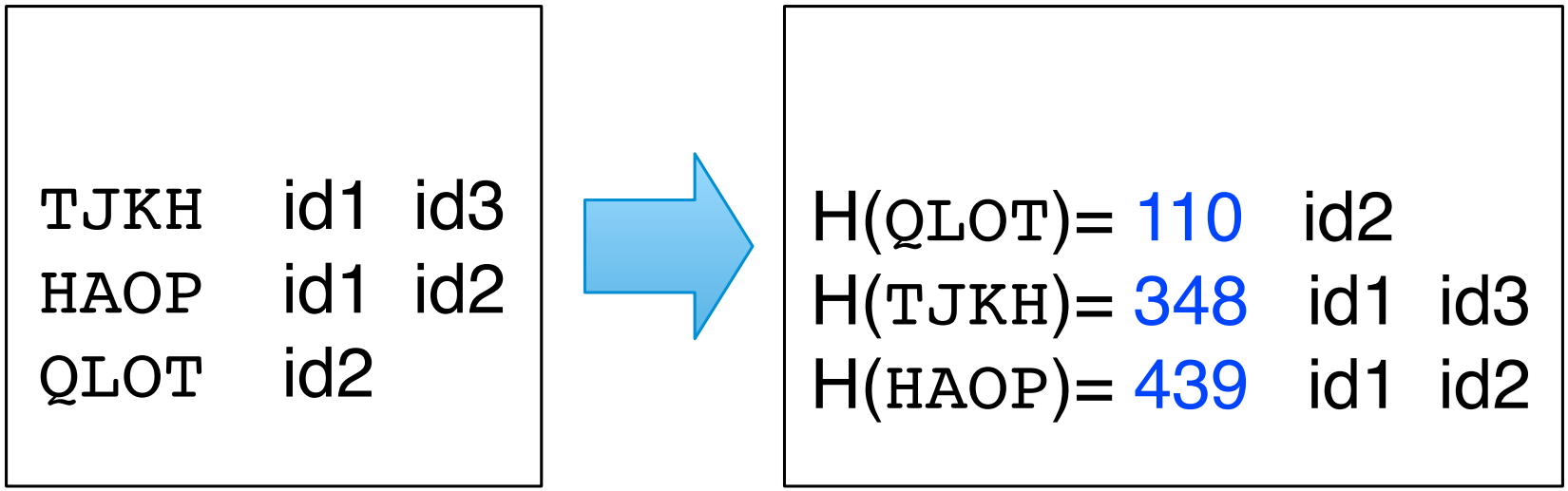


Build It Yourself

Hash(TJKH) = 348; Hash(HAQP) = 439;

Hash(QLOT) = 110;

- Build an encrypted inverted index with hash table

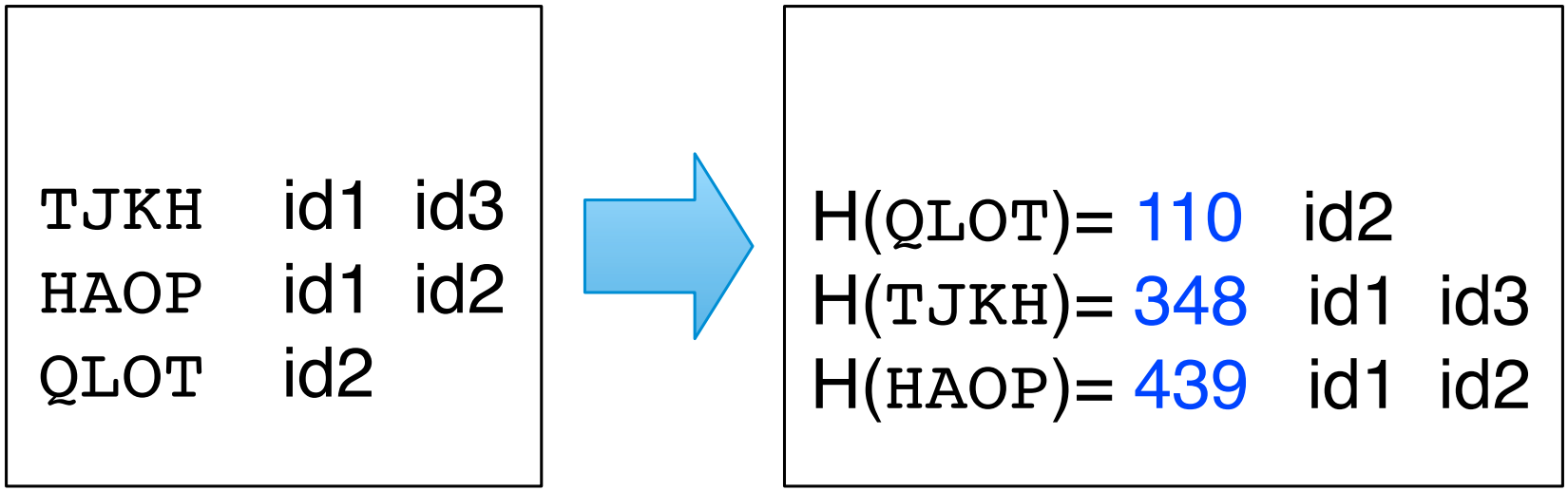


Build It Yourself

$PRF_k(\texttt{dayton}) = \texttt{TJKH}$; $PRF_k(\texttt{usa}) = \texttt{HAQP}$;

$PRF_k(\texttt{data}) = \texttt{QLOT}$

- If the client wants to search "**data**", what is token?
- Which files will be retrieved?

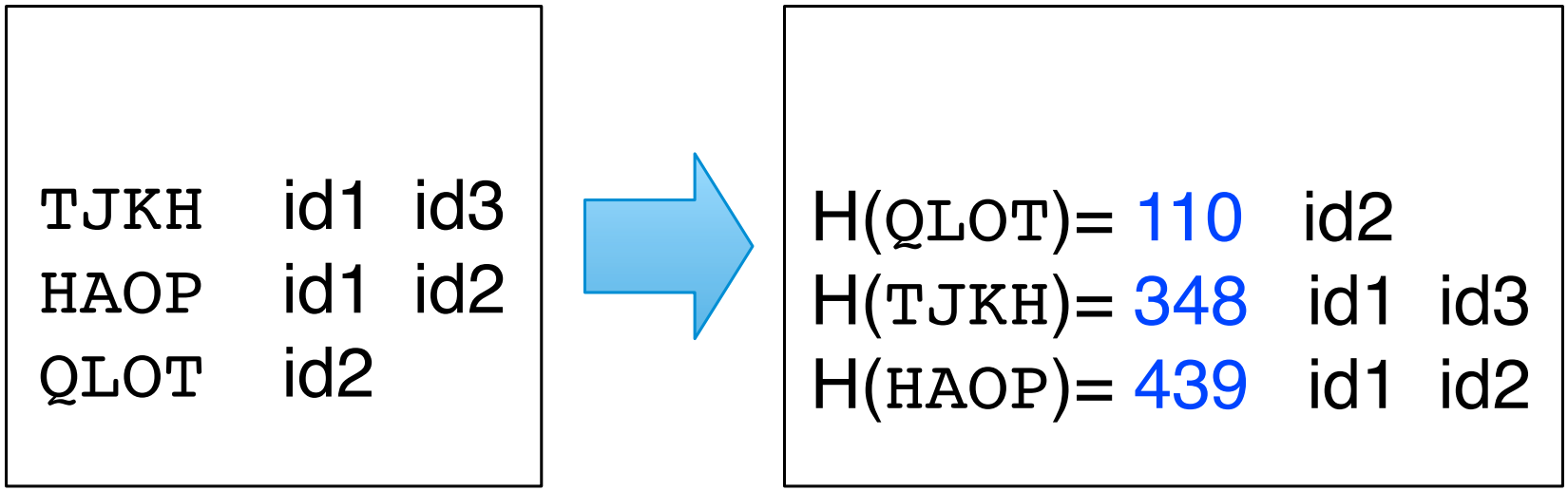


Build It Yourself

$PRF_k(\texttt{dayton}) = \texttt{TJKH}$; $PRF_k(\texttt{usa}) = \texttt{HAQP}$;

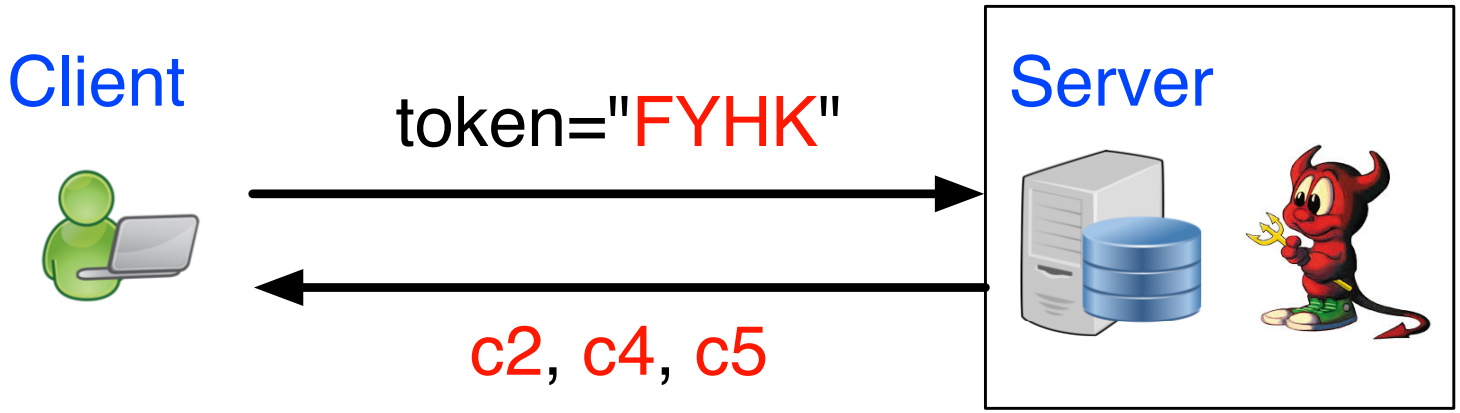
$PRF_k(\texttt{data}) = \texttt{QLOT}$

- If the client wants to search "**data**", token = QLOT
- encrypted file 2 will be retrieved.



Leakage in SE

- SE does not hide all the information.
  - Server knows the no. of keywords searched
    - One token is one keyword
  - Server knows how many files were retrieved
    - Files are encrypted, Number does not change



Leakage after Search

- SE does not hide all the information.
  - Server knows two keyword queries are identical
    - Token: $PRF_k(m1) = w1$, $PRF_k(m2) = w2$
    - If w1 = w2, then m1 = m2
  - If Alice always searches “UC”, then server knows she repeats same queries
  - Server knows “popular” encrypted files
    - Knows no. of access for each id/address
    - id2 retrieved 100 times, id1 retrieved 3 times

Leakage after Search

- Leakage Function
  - Access pattern (for each token): a binary vector with n bits, n is the total number of files, 0 indicates not return, 1 indicate return
  - E.g., n=5, {f1, f2, f3, f4, f5}
  - Given a search token tk, its access pattern is b = (0, 0, 1, 0, 1), return file f3 and f5
- Two tokens are indistinguishable if their access pattern have same number of 1s.

Leakage after Search

- Access pattern have same number of 1s.
  - Given tk, access pattern b = (0, 1, 1, 0, 0)
  - Given tk', access pattern b' = (1, 1, 0, 0, 0)
  - tk and tk' are indistinguishable
    - The order of files could change (permutation)
    - Same number of returned files
- If (advanced) adversary knows "dayton" and "usa" both return 2 files, if token TJKH returns 2 files, this adversary cannot tell which keyword

Leakage after Search

- Access pattern have different number of 1s.
  - Given tk, access pattern b = (0, 0, 1, 0, 0)
  - Given tk', access pattern b' = (1, 1, 0, 0, 0)
  - tk and tk' are distinguishable
    - Different number of returned files

- If (advanced) adversary knows only "data" returns 1 file, then a token QLOT returns 1 file, then QLOT must be "data" based on access pattern.

Leakage before Search

- Leakage before any search
  - Attacker knows the structure of encrypted index
  - QLOT's linked list only has 1 node (i.e., 1 id)
  - If (advanced) adversary knows only "data" has 1 associated id/file, then data <—> QLOT

	f1	f2	f3
TJKH	1	0	1
HAOP	1	1	0
QLOT	0	1	0

TJKH	id1	id3
HAOP	id1	id2
QLOT	id2	

Adding Dummy Nodes

- Add dummy nodes in linked lists
  - All the linked lists have the same size
  - Tradeoff: Extra search time to scan dummy nodes

	f1	f2	f3
DGTH	1	0	1
HKJW	1	1	0
QWET	0	1	0

DGTH	id1	id3
HKJW	id1	id2
QWET	id2	idx

Adding Dummy Nodes

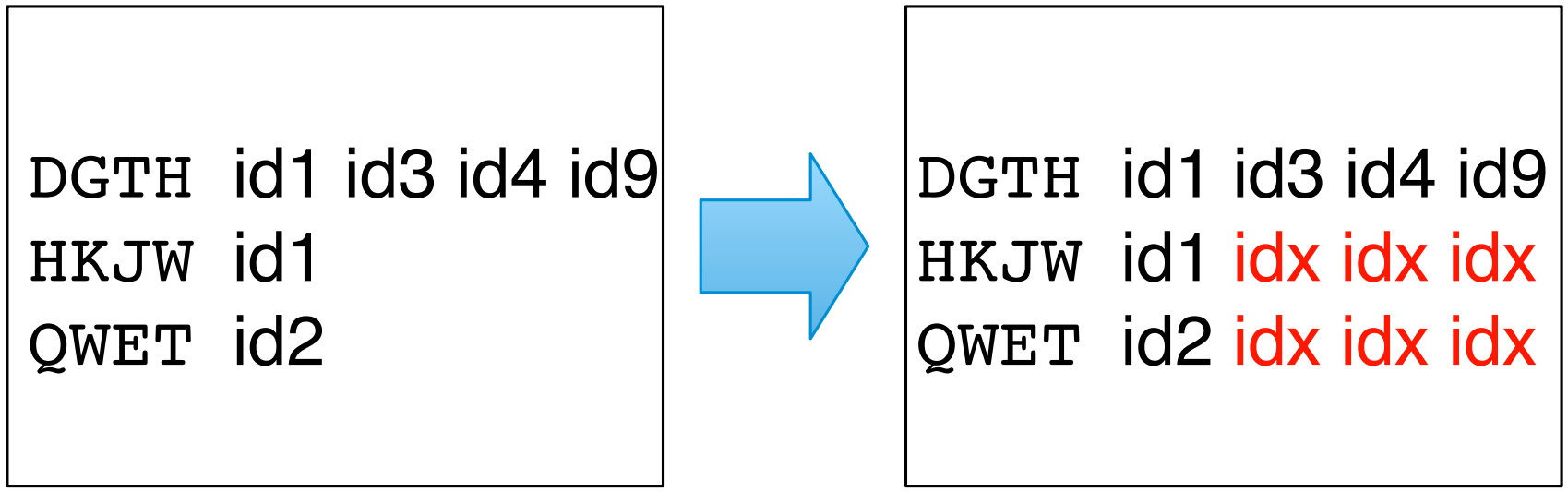
- Add dummy nodes in linked lists
  - If idx is not a real id, easy to discover
  - if idx is a real id (id1 or id3), extra unrelated files in search results

	f1	f2	f3
DGTH	1	0	1
HKJW	1	1	0
QWET	0	1	0

DGTH	id1	id3
HKJW	id1	id2
QWET	id2	idx

Adding Dummy Nodes

- Add dummy nodes in linked lists
  - Less efficient if one has a long linked list
  - For **HKJW**, instead of 1 node, 4 nodes now need to be scanned, but 3 of them are dummies



Research in SE

- Different types of queries
- New attacks based on access pattern
- Hiding access pattern

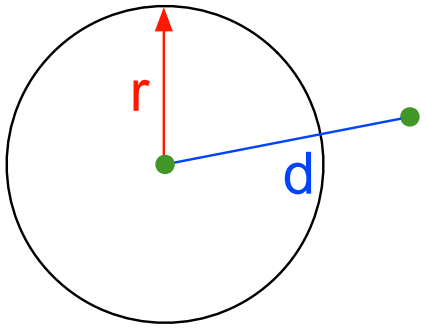
Different Queries

- Keyword search
  - Find files including keyword “UC”
  - Equality (easier to do on encrypted data)

- Range queries
  - Find age between [10, 20]
  - Comparisons (not easy, but possible)
  - Has to leak more information

Different Queries

- Nearest Neighbor queries
  - Find friends close to me
  - Compute a distance then compare which one is the smallest (not easy, but possible)



- Queries in Location-Based Services
  - Find a point inside a circle
  - Compute then compare (not easy, but possible)

New Attacks

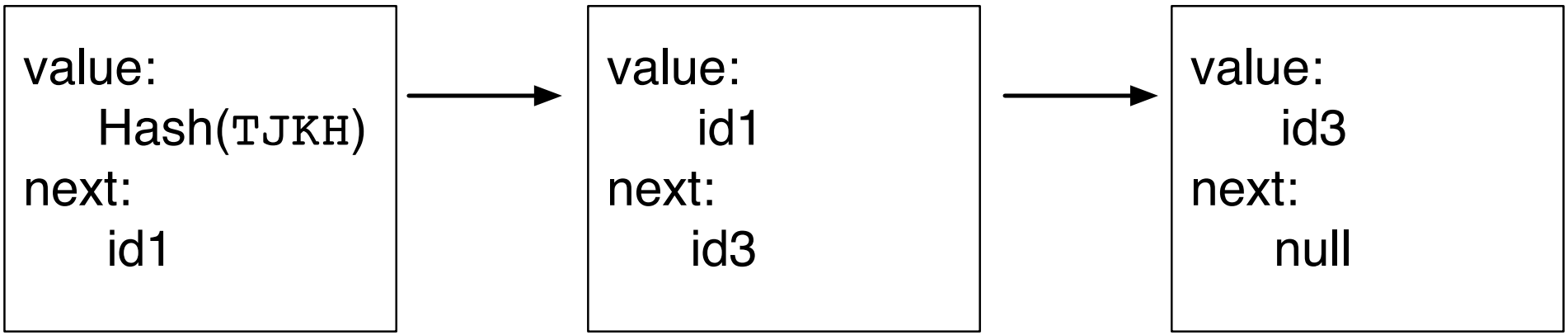
- Based on the leakage on access pattern
  - Passive attacks: an attacker has another similar dataset in plaintext + its access pattern
    - Use matching problems to attack
  - Active attacks: select and inject files smartly, s.t. each possible token will have unique access pattern (i.e., a unique binary vector)

Hiding Access Pattern

- Oblivious RAM
  - Server doest now know which encrypted files client retrieved
  - Client still gets associated files for a token
- Main idea:
    - Retrieve a small group of encrypted files
    - Re-encrypt, shuffle, and re-upload
- Theoretically possible, huge costs in practice
    - More cost than downloading the entire data

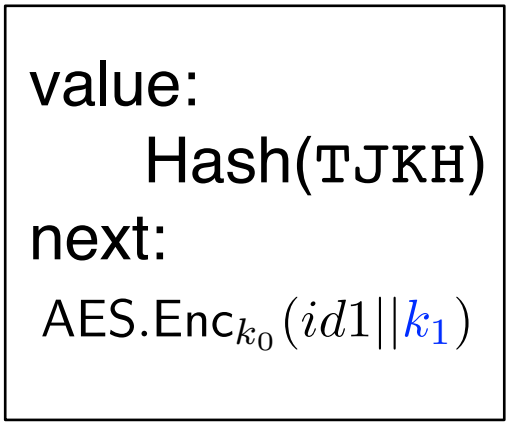
Leakage before Search

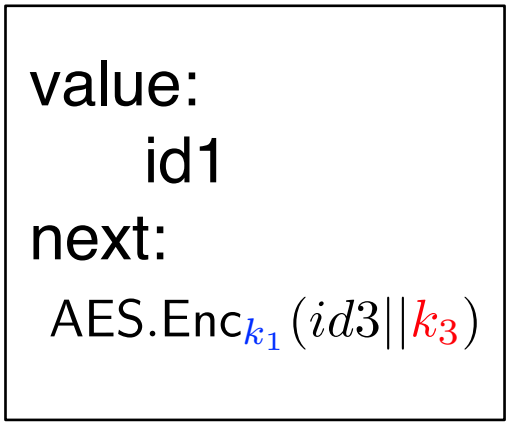
- Leakage before any search
  - Attacker knows the next node of each node based on all the pointers
  - Hash(`TJKH`)'s next is id1, id1's next is id3, id3's next is null (end of linked list)

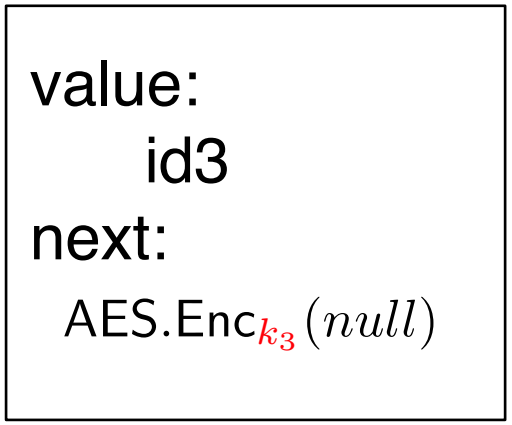


Hiding Pointers

- Only search a parent node will leak its child node
  - Given **TJHK** and k_0, search Hash(**TJKH**), decrypt & reveal next node is id1 and obtain k_1
  - Given id1 and k1, search node id1, decrypt & reveal next node is id3 and obtain k3







Additional Reading

R. Curmola, J. Garay, S. Kamara, R. Ostrovsky, "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions," in the Proceedings of the 13th ACM Conference on Computer and Communication Security (CCS) 2006