Searchable Encryption

Xaitheng Yang

University of Minnesota, Morris

Nov 2018

Introduction

- More cloud services $\rightarrow$ storage of data on third party servers.

Introduction

- More cloud services $\rightarrow$ storage of data on third party servers.
- Storage on third party servers $\rightarrow$ potential for exposing this information to others.

Introduction

- More cloud services $\rightarrow$ storage of data on third party servers.
- Storage on third party servers $\rightarrow$ potential for exposing this information to others.
- To avoid this, data is often encrypted.

Introduction

- More cloud services $\rightarrow$ storage of data on third party servers.
- Storage on third party servers $\rightarrow$ potential for exposing this information to others.
- To avoid this, data is often encrypted.
- Needing to decrypt makes things difficult

Introduction

- More cloud services $\rightarrow$ storage of data on third party servers.
- Storage on third party servers $\rightarrow$ potential for exposing this information to others.
- To avoid this, data is often encrypted.
- Needing to decrypt makes things difficult
- The specific security and performance demands of these modern day situations has created a necessity for searchable encryption.

Outline

Databases

Background: Databases

A database can be seen as a structured set of data in the form of a table, with each row containing an entry.

Index	Document
1	Pepperoni Pizza
2	Pepperoni Pineapple Pizza
3	Sausage Pizza
4	Ham Pineapple Pizza

Background: Databases

- Queries are instructions that can be sent to a server containing a database.

Background: Databases

- Queries are instructions that can be sent to a server containing a database.
- They can be used to add, retrieve (search), update, or delete data within the database, depending on the database's functionality.

Background: Databases

- Queries are instructions that can be sent to a server containing a database.
- They can be used to add, retrieve (search), update, or delete data within the database, depending on the database's functionality.
- The source of the queries is called the *client*.

Background: Databases

- Queries are instructions that can be sent to a server containing a database.
- They can be used to add, retrieve (search), update, or delete data within the database, depending on the database's functionality.
- The source of the queries is called the *client*.
- The receiver of the queries is called the *server*.

Background: Encryption

Encryption

Background: Encryption

Encryption is the process of encoding information so that only authorized users can access it.

Background: Encryption

Encryption is the process of encoding information so that only authorized users can access it.

- Encryption key - a string of bits created for encoding and/or decoding information
- Plaintext - non-encrypted data
- Ciphertext - encrypted data

Background: Encryption

Encryption is the process of encoding information so that only authorized users can access it.

- Encryption key - a string of bits created for encoding and/or decoding information
- Plaintext - non-encrypted data
- Ciphertext - encrypted data

Index	Document
1	Pizza Box 1
2	Pizza Box 2
3	Pizza Box 3
4	Pizza Box 4

Searchable Encryption

Background: Searchable Encryption

Searchable encryption is a class of structured encryption.

Background: Searchable Encryption

Searchable encryption is a class of structured encryption.
It allows for the performance of queries on its encrypted data without having to decrypt the data.

Background: Searchable Encryption

Searchable encryption is a class of structured encryption.
It allows for the performance of queries on its encrypted data without having to decrypt the data.
Queries are run on the keywords in order to identify what data to operate on.

Background: Searchable Encryption

So, the database might look something like this:

Background: Searchable Encryption

So, the database might look something like this:

Keyword	Indexes
pepperoni	1, 2
pineapple	2, 4
sausage	3
ham	4

Index	Document
1	Pizza Box
2	Pizza Box
3	Pizza Box
4	Pizza Box

Background: Searchable Encryption

So, the database might look something like this:

Keyword	Indexes
pepperoni	1, 2
pineapple	2, 4
sausage	3
ham	4

Index	Document
1	Pizza Box
2	Pizza Box
3	Pizza Box
4	Pizza Box

or

Index	Keyword	Document
1	Pepperoni	Pizza Box
2	Pepperoni Pineapple	Pizza Box
3	Sausage	Pizza Box
4	Ham Pineapple	Pizza Box

Background: Searchable encryption

Searchable encryption necessarily leaks some amount of information.

Background: Searchable encryption

Searchable encryption necessarily leaks some amount of information.
This leakage has been shown to allow:

- leakage-abuse attacks
- full plaintext recovery of encrypted databases

Background: Forward Privacy

Forward Privacy:

Background: Forward Privacy

Forward Privacy:
A searchable encryption scheme is said to be forward private if queries to the server don't reveal which keywords are involved in the keyword/document pairs.

Background: Backward Privacy

Backward Privacy:

Background: Backward Privacy

Backward Privacy:
A searchable encryption scheme is backward private if search queries on the database don't reveal information about documents that were deleted.
This can be classified in 3 different levels

Background: Backward Privacy

An example with pizza:

Background: Backward Privacy

An example with pizza:

- (T1) add to index 1, pepperoni pineapple pizza

Background: Backward Privacy

An example with pizza:

- (T1) add to index 1, pepperoni pineapple pizza
- (T2) add to index 2, pepperoni pizza

Background: Backward Privacy

An example with pizza:

- (T1) add to index 1, pepperoni pineapple pizza
- (T2) add to index 2, pepperoni pizza
- (T3) remove from index 1, pepperoni topping

Background: Backward Privacy

An example with pizza:

- (T1) add to index 1, pepperoni pineapple pizza
- (T2) add to index 2, pepperoni pizza
- (T3) remove from index 1, pepperoni topping
- (T4) add to index 3, pineapple pizza

Background: Backward Privacy

An example with pizza:

- (T1) add to index 1, pepperoni pineapple pizza
- (T2) add to index 2, pepperoni pizza
- (T3) remove from index 1, pepperoni topping
- (T4) add to index 3, pineapple pizza

If there is a search for pepperoni:

Background: Backward Privacy

An example with pizza:

- (T1) add to index 1, pepperoni pineapple pizza
- (T2) add to index 2, pepperoni pizza
- (T3) remove from index 1, pepperoni topping
- (T4) add to index 3, pineapple pizza

If there is a search for pepperoni:

I. Backward privacy with insertion pattern:
leaks the documents currently matching a keyword, when they were inserted, and the total number of updates on the keyword.

Background: Backward Privacy

An example with pizza:

- (T1) add to index 1, pepperoni pineapple pizza
- (T2) add to index 2, pepperoni pizza
- (T3) remove from index 1, pepperoni topping
- (T4) add to index 3, pineapple pizza

If there is a search for pepperoni:

I. Backward privacy with insertion pattern:

leaks the documents currently matching a keyword, when they were inserted, and the total number of updates on the keyword.

- index 1 matches the keyword pepperoni
- the time at which this entry was added
- three updates occurred for pepperoni

Background: Backward Privacy

An example with pizza:

- (T1) add to index 1, pepperoni pineapple pizza
- (T2) add to index 2, pepperoni pizza
- (T3) remove from index 1, pepperoni topping
- (T4) add to index 3, pineapple pizza

If there is a search for pepperoni:

I. Backward privacy with insertion pattern:

leaks the documents currently matching a keyword, when they were inserted, and the total number of updates on the keyword.

- index 1 matches the keyword pepperoni
- the time at which this entry was added
- three updates occurred for pepperoni

Background: Backward Privacy

An example with pizza:

- (T1) add to index 1, pepperoni pineapple pizza
- (T2) add to index 2, pepperoni pizza
- (T3) remove from index 1, pepperoni topping
- (T4) add to index 3, pineapple pizza

If there is a search for pepperoni:
II. Backward privacy with update pattern:
when all the updates on the keyword happened

- index 1 matches the keyword pepperoni
- the time at which this entry was added
- three updates occurred for pepperoni and the time
- the time the three updates for pepperoni occurred

Background: Backward Privacy

An example with pizza:

- (T1) add to index 1, pepperoni pineapple pizza
- (T2) add to index 2, pepperoni pizza
- (T3) remove from index 1, pepperoni topping
- (T4) add to index 3, pineapple pizza

If there is a search for pepperoni:
III. Weak backward privacy:
which deletion update canceled which insertion update.

- index 1 matches the keyword pepperoni
- the time at which this entry was added
- three updates occurred for pepperoni and the time
- the time the three updates for pepperoni occurred
- the time index 1 had pepperoni removed from it

Dual Dictionary

Dual Dictionary

The dual dictionary scheme, proposes a new data structure to handle indexes, called dual dictionary.

Dual Dictionary

The dual dictionary scheme, proposes a new data structure to handle indexes, called dual dictionary.
The dual dictionary data structure consists of linked dictionaries for inverted and forward indexes

Dual Dictionary

The dual dictionary scheme, proposes a new data structure to handle indexes, called dual dictionary.
The dual dictionary data structure consists of linked dictionaries for inverted and forward indexes

- Inverted index: maintains lists of documents per keyword
- Forward index: maintains lists of keywords per document

Dual Dictionary

How it works:

Dual Dictionary

How it works:
The client uses encryption keys to create labels for the data stored in the database.

Dual Dictionary

How it works:
The client uses encryption keys to create labels for the data stored in the database.

- Delete Label - *DL*
- Search Label - *SL*

Dual Dictionary

How it works:
The client uses encryption keys to create labels for the data stored in the database.

- Delete Label - *DL*
- Search Label - *SL*

Dual Dictionary

Delete Label - *DL*

Dual Dictionary

Delete Label - *DL*

- created for every keyword matching a document
- generated using a key corresponding to the document
- client keeps a count of how many there are per document

Dual Dictionary

Delete Label - *DL*

- created for every keyword matching a document
- generated using a key corresponding to the document
- client keeps a count of how many there are per document

For a pepperoni pineapple pizza being added to an empty database:

Dual Dictionary

Delete Label - *DL*

- created for every keyword matching a document
- generated using a key corresponding to the document
- client keeps a count of how many there are per document

For a pepperoni pineapple pizza being added to an empty database:
We would generate 2 *DL*'s

Dual Dictionary

Delete Label - *DL*

- created for every keyword matching a document
- generated using a key corresponding to the document
- client keeps a count of how many there are per document

For a pepperoni pineapple pizza being added to an empty database:
We would generate 2 *DL*'s
$pizza - 1 - DL_1$

- generated using the key for **index 1** and the number **1**, because it is the **first** keyword of the **first** pizza

Dual Dictionary

Delete Label - *DL*

- created for every keyword matching a document
- generated using a key corresponding to the document
- client keeps a count of how many there are per document

For a pepperoni pineapple pizza being added to an empty database:
We would generate 2 *DL*'s

$pizza - 1 - DL_1$

- generated using the key for **index 1** and the number **1**, because it is the **first** keyword of the **first** pizza

$pizza - 1 - DL_2$

- generated using the key for **index 1** and the number **2**, because it is the **second** keyword of the **first** pizza

Dual Dictionary

Search Label - *SL*

Dual Dictionary

Search Label - *SL*

- created for every document matching a given keyword
- generated using a key corresponding to the keyword
- client keeps a count of how many there are per keyword

Dual Dictionary

Search Label - *SL*

- created for every document matching a given keyword
- generated using a key corresponding to the keyword
- client keeps a count of how many there are per keyword

For the pepperoni pineapple pizza:

Dual Dictionary

Search Label - *SL*

- created for every document matching a given keyword
- generated using a key corresponding to the keyword
- client keeps a count of how many there are per keyword

For the pepperoni pineapple pizza:
We would then generate 2 *SL*'s

Dual Dictionary

Search Label - SL

- created for every document matching a given keyword
- generated using a key corresponding to the keyword
- client keeps a count of how many there are per keyword

For the pepperoni pineapple pizza:
We would then generate 2 SL's
pepperoni $- SL_1$

- generated using the key for pepperoni and the number **1**

Dual Dictionary

Search Label - SL

- created for every document matching a given keyword
- generated using a key corresponding to the keyword
- client keeps a count of how many there are per keyword

For the pepperoni pineapple pizza:
We would then generate 2 SL's
$pepperoni - SL_1$

- generated using the key for pepperoni and the number **1**

$pineapple - SL_1$

- generated using the key for pineapple and the number **1**

Dual Dictionary

These labels are stored in two dictionaries. As a pair, (DL_i,SL_i), in Dic_1

As a triplet with a document index, $(SL_i,(DL_i, \text{index}))$, in Dic_2

Dual Dictionary

These labels are stored in two dictionaries. As a pair, (DL_i, SL_i), in Dic_1
As a triplet with a document index, $(SL_i, (DL_i, \text{index}))$, in Dic_2

	DL	SL
Dic_1	$pizza - 1 - DL_1$	$pepperoni - SL_1$
	$pizza - 1 - DL_2$	$pineapple - SL_1$

	SL	(DL, Index)
Dic_2	$pepperoni - SL_1$	$(pizza - 1 - DL_1, 1)$
	$pineapple - SL_1$	$(pizza - 1 - DL_2, 1)$

Dual Dictionary

DL	SL
$pizza - 1 - DL_1$	$pepperoni - SL_1$
$pizza - 2 - DL_1$	$pepperoni - SL_2$
$pizza - 2 - DL_2$	$pineapple - SL_1$
$pizza - 3 - DL_1$	$sausage - SL_1$
$pizza - 4 - DL_1$	$ham - SL_1$
$pizza - 4 - DL_2$	$pineapple - SL_2$

SL	(DL, Index)
$pepperoni - SL_1$	$(pizza - 1 - DL_1, 1)$
$pepperoni - SL_2$	$(pizza - 2 - DL_1, 2)$
$pineapple - SL_1$	$(pizza - 2 - DL_2, 2)$
$pineapple - SL_2$	$(pizza - 4 - DL_2, 4)$
$sausage - SL_1$	$(pizza - 3 - DL_1, 3)$
$ham - SL_1$	$(pizza - 4 - DL_1, 4)$

Dual Dictionary

Deleting a document (pizza) with index 2:

Dual Dictionary

Deleting a document (pizza) with index 2:

- calculate *DL*'s for however many keywords the document has (in this case: 2)

Dual Dictionary

Deleting a document (pizza) with index 2:

- calculate DL's for however many keywords the document has (in this case: 2)
- search Dic_1 for the DL's (in this case: $pizza - 2 - DL_1$, $pizza - 2 - DL_2$)

Dual Dictionary

Deleting a document (pizza) with index 2:

- calculate DL's for however many keywords the document has (in this case: 2)
- search Dic_1 for the DL's (in this case: $pizza - 2 - DL_1$, $pizza - 2 - DL_2$)

DL	SL
$pizza - 1 - DL_1$	$pepperoni - SL_1$
$pizza - 2 - DL_1$	$pepperoni - SL_2$
$pizza - 2 - DL_2$	$pineapple - SL_1$
$pizza - 3 - DL_1$	$sausage - SL_1$
$pizza - 4 - DL_1$	$ham - SL_1$
$pizza - 4 - DL_2$	$pineapple - SL_2$

Dual Dictionary

Deleting a document (pizza) with index 2:

- calculate DL's for however many keywords the document has (in this case: 2)
- search Dic_1 for the DL's (in this case: $pizza - 2 - DL_1$, $pizza - 2 - DL_2$)

DL	SL
$pizza - 1 - DL_1$	$pepperoni - SL_1$
$pizza - 2 - DL_1$	$pepperoni - SL_2$
$pizza - 2 - DL_2$	$pineapple - SL_1$
$pizza - 3 - DL_1$	$sausage - SL_1$
$pizza - 4 - DL_1$	$ham - SL_1$
$pizza - 4 - DL_2$	$pineapple - SL_2$

Dual Dictionary

Deleting a document (pizza) with index 2:
We would then use the result to identify which SL's to look for in Dic_2.

Dual Dictionary

Deleting a document (pizza) with index 2:
We would then use the result to identify which SL's to look for in Dic_2.

SL	(DL, Index)
$pepperoni - SL_1$	$(pizza - 1 - DL_1, 1)$
$pepperoni - SL_2$	$(pizza - 2 - DL_1, \mathbf{2})$
$pineapple - SL_1$	$(pizza - 2 - DL_2, \mathbf{2})$
$pineapple - SL_2$	$(pizza - 4 - DL_2, 4)$
$sausage - SL_1$	$(pizza - 3 - DL_1, 3)$
$ham - SL_1$	$(pizza - 4 - DL_1, 4)$

Dual Dictionary

Deleting a document (pizza) with index 2:
We would then use the result to identify which SL's to look for in Dic_2.

SL	(DL, Index)
$pepperoni - SL_1$	$(pizza - 1 - DL_1, 1)$
$pepperoni - SL_2$	$(pizza - 2 - DL_1, \mathbf{2})$
$pineapple - SL_1$	$(pizza - 2 - DL_2, \mathbf{2})$
$pineapple - SL_2$	$(pizza - 4 - DL_2, 4)$
$sausage - SL_1$	$(pizza - 3 - DL_1, 3)$
$ham - SL_1$	$(pizza - 4 - DL_1, 4)$

Then all the results from Dic_1, Dic_2, and the document would be deleted.

Dual Dictionary

Retrieving documents (pizzas) with keyword (topping) *pineapple*:

Dual Dictionary

Retrieving documents (pizzas) with keyword (topping) *pineapple*:

- the client calculates SL_i for however many documents match pineapple (in this case: 2)

Dual Dictionary

Retrieving documents (pizzas) with keyword (topping) *pineapple*:

- the client calculates SL_i for however many documents match pineapple (in this case: 2)
- search Dic_2 for the SL's (in this case: *pineapple* – SL_1, *pineapple* – SL_2)

SL	(*DL*, Index)
pepperoni – SL_1	(*pizza* – 1 – DL_1, 1)
pepperoni – SL_2	(*pizza* – 2 – DL_1, 2)
pineapple – SL_1	(*pizza* – 2 – DL_2, 2)
pineapple – SL_2	(*pizza* – 4 – DL_2, 4)
sausage – SL_1	(*pizza* – 3 – DL_1, 3)
ham – SL_1	(*pizza* – 4 – DL_1, 4)

Dual Dictionary

Retrieving documents (pizzas) with keyword (topping) *pineapple*:

- the client calculates SL_i for however many documents match pineapple (in this case: 2)
- search Dic_2 for the SL's (in this case: *pineapple* – SL_1, *pineapple* – SL_2)

SL	(*DL*, Index)
pepperoni – SL_1	(*pizza* – 1 – DL_1, 1)
pepperoni – SL_2	(*pizza* – 2 – DL_1, 2)
pineapple – SL_1	(*pizza* – 2 – DL_2, **2**)
pineapple – SL_2	(*pizza* – 4 – DL_2, **4**)
sausage – SL_1	(*pizza* – 3 – DL_1, 3)
ham – SL_1	(*pizza* – 4 – DL_1, 4)

Dual Dictionary

Retrieving documents (pizzas) with keyword (topping) *pineapple*: We would then use the result to identify which documents (pizzas) to retrieve.

Index	Document
1	Pizza Box
2	**Pizza Box**
3	Pizza Box
4	**Pizza Box**

Dual Dictionary

Retrieving documents (pizzas) with keyword (topping) *pineapple*: We would then use the result to identify which documents (pizzas) to retrieve.

Index	Document
1	Pizza Box
2	**Pizza Box**
3	Pizza Box
4	**Pizza Box**

Index	Document
1	Pepperoni Pizza
2	Pepperoni Pineapple Pizza
3	Sausage Pizza
4	Ham Pineapple Pizza

Dual Dictionary

Privacy After Searches

Dual Dictionary

Privacy After Searches

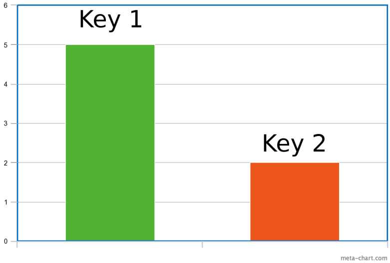
Recall: Forward private if queries to the server don't reveal which keywords are involved in the keyword/document pairs.

- Dual Dictionary switches keys after every search

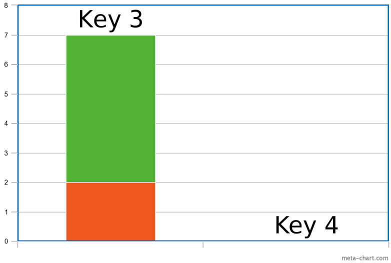
Privacy After Searches

Recall: Forward private if queries to the server don't reveal which keywords are involved in the keyword/document pairs.

- Dual Dictionary switches keys after every search



(a) Key 1 and Key 2 before search



(b) Key 3 and Key 4 after search

Dual Dictionary

Privacy

Recall: Backward private if search queries on the database don't reveal information about documents that were deleted.

Privacy

Recall: Backward private if search queries on the database don't reveal information about documents that were deleted.
The Dual Dictionary scheme isn't backward private. This is because further search queries would reveal the documents that were deleted.

Fides

Fides

Fides is a forward and backward private searchable encryption scheme.
It is a combination of:

- $\Sigma o\psi o\zeta$ - a forward private scheme
- Two-roundtrip - a technique for backward privacy

Fides

Key features of $\Sigma o\psi o\zeta$:
Forward privacy through tokens

Fides

Key features of $\Sigma o\psi o\zeta$:

Forward privacy through tokens

- Search token - generated by the number of documents matching a given keyword with a one-way trapdoor permutation

Fides

Key features of $\Sigma o\psi o\zeta$:

Forward privacy through tokens

- Search token - generated by the number of documents matching a given keyword with a one-way trapdoor permutation
  - The client keeps the most recent search token (ST_n)
  - The client can generate a new search token (ST_{n+1}) based on an old one (ST_n)
  - The server given ST_n, can derive ST_{n-1} to ST_0 but not ST_{n+1}

Fides

Key features of $\Sigma o\psi o\zeta$:

Forward privacy through tokens

- Search token - generated by the number of documents matching a given keyword with a one-way trapdoor permutation
  - The client keeps the most recent search token (ST_n)
  - The client can generate a new search token (ST_{n+1}) based on an old one (ST_n)
  - The server given ST_n, can derive ST_{n-1} to ST_0 but not ST_{n+1}
- Update token - generated to correspond to 1 search token and are paired with a document index

Fides

Key features of Two-Roundtrip:

Fides

Key features of Two-Roundtrip:

- documents aren't returned in a query
- ciphertext containing the document index and operation is returned

Fides

Key features of Two-Roundtrip:

- documents aren't returned in a query
- ciphertext containing the document index and operation is returned
  - ciphertext encrypted by the client with a key
  - key is unique for each keyword
  - operation is **addition** or **deletion**

Fides

How Fides Functions Differently from $\Sigma o \psi o \zeta$:

Fides

How Fides Functions Differently from $\Sigma o \psi o \zeta$:

$\Sigma o \psi o \zeta$ **Scheme**

Step	Client		Server
1	Search Token	$\rightarrow$	Calculate Search Tokens
2			Find Indexes with Corresponding Update Tokens
3		$\leftarrow$	Documents

Fides

How Fides Functions Differently from $\Sigma o\psi o\zeta$:

$\Sigma o\psi o\zeta$ Scheme

Step	Client		Server
1	Search Token	$\rightarrow$	Calculate Search Tokens
2			Find Indexes with Corresponding Update Tokens
3		$\leftarrow$	Documents

Fides First Trip

Step	Client		Server
1	Search Token	$\rightarrow$	Calculate Search Tokens
2			Find Ciphertext with Corresponding Update Tokens
3	Decrypt Ciphertext	$\leftarrow$	Ciphertext

Fides

How Fides works:

First Trip

Step	Client		Server
1	Search Token	$\rightarrow$	Calculate Search Tokens
2			Find Ciphertext with Corresponding Update Tokens
3	Decrypt Ciphertext	$\leftarrow$	Ciphertext

Second Trip

Step	Client		Server
4	Indexes and New Ciphertext	$\rightarrow$	Update Ciphertext
5		$\leftarrow$	Documents

Fides

How Fides works:

Update Token	Ciphertext
pepperoni 1	$e(1, ADD)$
pepperoni 2	$e(2, ADD)$
pineapple 1	$e(2, ADD)$
pineapple 2	$e(4, ADD)$
sausage 1	$e(3, ADD)$
ham 1	$e(4, ADD)$

Index	Document
1	Pizza Box
2	Pizza Box
3	Pizza Box
4	Pizza Box

Fides

How Fides works:

If we searched for pineapple

Update Token	Ciphertext
pepperoni 1	$e(1, ADD)$
pepperoni 2	$e(2, ADD)$
pineapple 1	**e(2, ADD)**
pineapple 2	**e(4, ADD)**
sausage 1	$e(3, ADD)$
ham 1	$e(4, ADD)$

Index	Document
1	Pizza Box
2	Pizza Box
3	Pizza Box
4	Pizza Box

Fides

How Fides works:

If we searched for pineapple

Update Token	Ciphertext
pepperoni 1	$e(1, ADD)$
pepperoni 2	$e(2, ADD)$
pineapple 1	**e(2, ADD)**
pineapple 2	**e(4, ADD)**
sausage 1	$e(3, ADD)$
ham 1	$e(4, ADD)$

Index	Document
1	Pizza Box
2	**Pizza Box**
3	Pizza Box
4	**Pizza Box**

Fides

How Fides works:

If we searched for pineapple

Index	Document
1	Pizza Box
2	**Pizza Box**
3	Pizza Box
4	**Pizza Box**

Index	Document
1	Pepperoni Pizza
2	**Pepperoni Pineapple Pizza**
3	Sausage Pizza
4	**Ham Pineapple Pizza**

Privacy

Privacy

Forward Private:

Fides

Privacy

Forward Private:

- the server doesn't know what keyword can be used for specific documents
- token system

Privacy

Forward Private:

- the server doesn't know what keyword can be used for specific documents
- token system

Backward Private with Update Pattern:

Privacy

Forward Private:

- the server doesn't know what keyword can be used for specific documents
- token system

Backward Private with Update Pattern:

- The server knows when updates occur, but not their content
- two-roundtrip ciphertext

Janus

Janus

Janus is a scheme that uses:

Janus

Janus is a scheme that uses:

- Any forward private scheme
- Puncturable Encryption
- Incremental Puncture

Puncturable Encryption and Incremental Puncture:

- Imagine having a key ring with all the keys to a building
- These keys can be taken off of the ring
- Security at the entrance can make you take off certain keys
  - We will call the instructions to take off keys a *key part*

Janus

How Janus is set up:

Janus

How Janus is set up:

- Uses 2 instances of the forward private scheme
  - Used for additions - stores a pair of keyword and encrypted index
  - Used for deletion - stores a pair of keyword and key part (which key to take off the ring)

Janus

How Janus is set up:

- Uses 2 instances of the forward private scheme
  - Used for additions - stores a pair of keyword and encrypted index
  - Used for deletion - stores a pair of keyword and key part (which key to take off the ring)
- Each keyword has its own puncturable key (*key ring*)
  - the client stores the *full key ring* for each of these keys

Janus

An example of what our database would look like:

Keyword	Encrypted Indexes
pepperoni	$e(1)$
pepperoni	$e(2)$
pineapple	$e(2)$
pineapple	$e(4)$
sausage	$e(3)$
ham	$e(4)$

Keyword	Key Part

Index	Document
1	Pizza Box
2	Pizza Box
3	Pizza Box
4	Pizza Box

Janus

How Janus inserts new documents:

Janus

How Janus inserts new documents:
For example: a ham pizza

Janus

How Janus inserts new documents:
For example: a ham pizza

- Client encrypts the new document's index with the key corresponding to its keyword
- Document is then sent to the database
- Keyword and encrypted-index pair is inserted in the addition instance

Janus

How Janus inserts new documents:

Keyword	Encrypted Indexes
pepperoni	$e(1)$
pepperoni	$e(2)$
pineapple	$e(2)$
pineapple	$e(4)$
sausage	$e(3)$
ham	$e(4)$
ham	**e(5)**

Index	Document
1	Pizza Box
2	Pizza Box
3	Pizza Box
4	Pizza Box
5	**Pizza Box**

Janus

How Janus deletes documents:

Janus

How Janus deletes documents:
For example, the ham pizza we just added:

Janus

How Janus deletes documents:
For example, the ham pizza we just added:

- The client calculates its key part (the instructions to remove a key from the ring)
- The client then inserts the key part to the deletion instance

Janus

How Janus deletes documents:
For example, the ham pizza we just added:

- The client calculates its key part (the instructions to remove a key from the ring)
- The client then inserts the key part to the deletion instance

Keyword	Key Part
Ham	sk_1^{ham}

Janus

How Janus searches documents:

Janus

How Janus searches documents:
For example, ham pizzas:

Janus

How Janus searches documents:

For example, ham pizzas:

- The client sends a search query with keyword (ham) and the corresponding key ring (sk_0^{ham})

Janus

How Janus searches documents:
For example, ham pizzas:

- The client sends a search query with keyword (ham) and the corresponding key ring (sk_0^{ham})
- Both instances are searched for the keyword

Janus

How Janus searches documents:
For example, ham pizzas:

- The client sends a search query with keyword (ham) and the corresponding key ring (sk_0^{ham})
- Both instances are searched for the keyword

Keyword	Encrypted Indexes
pepperoni	$e(1, sfadsa)$
pepperoni	$e(2, affdsa)$
pineapple	$e(2, lykuty)$
pineapple	$e(4, lfggry)$
sausage	$e(3, gregff)$
ham	**e(4, ytrhgg)**
ham	**e(5, yiperg)**

Keyword	Key Part
ham	sk_1^{ham}

Janus

How Janus searches documents:

Keyword	Encrypted Indexes
pepperoni	$e(1, sfadsa)$
pepperoni	$e(2, affdsa)$
pineapple	$e(2, lykuty)$
pineapple	$e(4, lfggry)$
sausage	$e(3, gregff)$
ham	**e(4, ytrhgg)**
ham	**e(5, yiperg)**

Keyword	Key Part
ham	sk_1^{ham}

- The server:
  - obtains the encrypted indexes from the addition instance
  - obtains the corresponding key parts from the deletion instance

Janus

How Janus searches documents:

Keyword	Encrypted Indexes
pepperoni	$e(1, sfadsa)$
pepperoni	$e(2, affdsa)$
pineapple	$e(2, lykuty)$
pineapple	$e(4, lfggry)$
sausage	$e(3, gregff)$
ham	**e(4, ytrhgg)**
ham	**e(5, yiperg)**

Keyword	Key Part
ham	sk_1^{ham}

- The server:
  - obtains the encrypted indexes from the addition instance
  - obtains the corresponding key parts from the deletion instance
  - can then remove the required keys from the key ring and decrypt the indexes that it still has keys for

Janus

How Janus searches documents:

Keyword	Encrypted Indexes
pepperoni	$e(1)$
pepperoni	$e(2)$
pineapple	$e(2)$
pineapple	$e(4)$
sausage	$e(3)$
ham	**e(4)**
ham	**e(5)**

Index	Document
1	Pizza Box
2	Pizza Box
3	Pizza Box
4	**Pizza Box**
5	Pizza Box

- With this, the server can retrieve one document

Janus

How Janus searches documents:

Index	Document
1	Pizza Box
2	Pizza Box
3	Pizza Box
4	**Pizza Box**
5	Pizza Box

Index	Document
1	Pepperoni Pizza
2	Pepperoni Pineapple Pizza
3	Sausage Pizza
4	**Ham Pineapple Pizza**
5	Ham Pizza

Janus

After Janus searches documents:
The server has now learned the indexes matching a keyword and its secret key.

Janus

After Janus searches documents:
The server has now learned the indexes matching a keyword and its secret key.

- For security any future insertions of that keyword will be encrypted with a new key

Janus

After Janus searches documents:
The server has now learned the indexes matching a keyword and its secret key.

- For security any future insertions of that keyword will be encrypted with a new key
  - key ring (sk^{ham}) replaced by a new key ring($sk^{new-ham}$)
  - necessary because the server would be able to use an old key ring to decrypt new entries and deletions

Janus

After Janus searches documents:

The server has now learned the indexes matching a keyword and its secret key.

- For security any future insertions of that keyword will be encrypted with a new key
  - key ring (sk^{ham}) replaced by a new key ring($sk^{new-ham}$)
  - necessary because the server would be able to use an old key ring to decrypt new entries and deletions
- Previous searches will be cached
  - cached results are no longer encrypted

Janus

After Janus searches documents:
The server has now learned the indexes matching a keyword and its secret key.

- For security any future insertions of that keyword will be encrypted with a new key
  - key ring (sk^{ham}) replaced by a new key ring($sk^{new-ham}$)
  - necessary because the server would be able to use an old key ring to decrypt new entries and deletions
- Previous searches will be cached
  - cached results are no longer encrypted

Keyword	Indexes
ham	4

Privacy

Privacy

Forward Privacy:
The scheme is forward private because it requires the use of a forward private scheme

Privacy

Forward Privacy:
The scheme is forward private because it requires the use of a forward private scheme

Backward Privacy:
The scheme has weak backward privacy, because:

- The server only has access to the key ring during a search query
- The key ring used for a keyword changes after every search

So, deleted indexes remain hidden. However, the server is able to tell which inserted entries were later deleted.

Janus' Other Privacy Considerations

Janus' Other Privacy Considerations

Janus is vulnerable to weaker adversaries.

- Persistent - constantly monitors from the beginning
- Late Persistent - constantly monitors from a given point in time
- Snapshot - only gets to view the database at one given point in time

Conclusions

SE Scheme	FP	BP	Other Considerations
Dual Dict.	✓	X	Two dictionaries takes twice the space
Fides	✓	With update pattern	Two roundtrips increases communication cost
Janus	✓	Weak	Vulnerable to weak adversaries

Acknowledgments

- Thanks to Elena Machkasova, my senior seminar professor and advisor

Discussion

Questions?

References

- R. Bost. $\Sigma o\phi o\varsigma$: Forward secure searchable encryption, 2016
- J. P., T. R., David Cash, Paul Grubbs. Leakage-abuse attacks against searchable encryption, 2015
- D. L., J. H., P. W., H. K., Kee sung Kim, Minkyu Kim. Forward Secure Dynamic Searchable Symmetric Encryption with Efficient Updates, 2017
- H. D., H. Knebl. Introduction to cryptography: Principles and applications. Springer Berlin Heidelberg, Heidelberg, Germany, 2007
- O. O., Raphael Bost, Brice Minaud. Forward and backward private searchable encryption from constrained cryptographic primitives 2017