

Searchable Symmetric Encryption

Ian Van Houdt & Charles Wright

Problem

- Who really owns your data?
 - SSL/TLS encrypts data over the wire.
 - Once across, who knows?
 - Even if it is encrypted at the server, whose data is it?
 - You don't have the keys to decrypt.
 - Server has the keys and can decrypt your data without your knowledge.
- You have no control of your data.
- No protection against external attacks.
- Server can release your data without permission
 - Solicitors, Governments, and other parties.

Solution ?

- Let's just encrypt everything!
 - Ok, but how do you locate the data you're looking for?
 - Server can't/won't be able to help you.
 - Just download entire contents of stored data and decrypt locally?
 - What about performance?

Searchable Symmetric Encryption

- Based off the 2014 research of David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner.
- Provides a scheme in which to encrypt text data such that a server can search and return encrypted data with learning minimal details about the contents of that data.

Searchable Symmetric Encryption

General Idea

Searchable Symmetric Encryption

Client Index:

Clear words, clear document IDs

word -> num of occurrences (at most: 1-per-file)

Word 0	c(0)
Word 1	c(1)
Word 2	c(2)
...	...
Word n	c(n)

Server Index:

Hashed words and encrypted document IDs

word -> document ID mapping

c18d3a0	CipherTxt
0a6278	CipherTxt
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

Searchable Symmetric Encryption

Client Index

Word 0	$c(0)$
Word 1	$c(1)$
Word 2	$c(2)$
...	...
Word n	$c(n)$

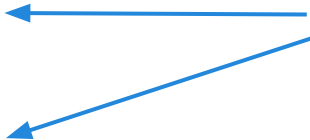
- Client keeps track of word allotment among messages

Searchable Symmetric Encryption

Client Index

...	...
Word "cat"	c("cat")++
Word "two"	c("two")++
...	...
...	...

- Client keeps track of word allotment among messages
- Upon creating new message, client maps each word of the message to the new message:
 - For each word in the new message increment count of that word



Searchable Symmetric Encryption

Client Index

...	...
Word "cat"	c("cat")
Word "two"	c("two")
...	...
...	...

- Client keeps track of word allotment among messages
- Upon creating new message, client maps each word of the message to the new message:
 - For each word in the new message increment count of that word
 - Using a password, input each word into a pseudorandom function to generate a pre-image resistant hash of that word

$PRF(\text{password}, c(\text{"cat"}) \parallel \text{"cat"}) \rightarrow \text{ccb215ad2018660ad49}$

Searchable Symmetric Encryption

Client Index

...	...
Word "cat"	c("cat")
Word "two"	c("two")
...	...
...	...

- Client keeps track of word allotment among messages
- Upon creating new message, client maps each word of the message to the new message:
 - For each word in the new message increment count of that word
 - Using a password, input each word into a pseudorandom function to generate a pre-image resistant hash of that word

$PRF(\text{password}, c(\text{"cat"}) \parallel \text{"cat"}) \rightarrow \text{ccb215ad2018660ad49}$

- Encrypt Doc ID and send new tuple to server

$(\text{ccb215ad2018660ad49}, \$2b\$12\$dd) \rightarrow \text{Server}$

Searchable Symmetric Encryption

- Encrypted message and set of tuples are sent to server.


Server Index

Terms Document IDs

c18d3a0	CipherTxt
...	...
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

Searchable Symmetric Encryption

- Encrypted message and set of tuples are sent to server.
- Server stores files and adds tuples to it's own local index.

(ccb215ad2018660ad49, \$2b\$12\$dd) 

- Server knows nothing about the contents of any messages, maintaining only an index of hashes and encrypted doc IDs.

Server Index

Terms Document IDs

c18d3a0	CipherTxt
ccb215a...	\$2b\$12...
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

Searchable Symmetric Encryption

- Encrypted message and set of tuples are sent to server.
- Server stores files and adds tuples to it's own local index.
 - Server knows nothing about the contents of any messages, maintaining only an index of hashes and encrypted doc IDs.
- Client can then search by generating multiple PRFs of the search term and sending them to the server, who can use these hashes to compute the matching index key (term) and return the value (Document ID).
 - The server never learns anything about the message contents.
 - More on this shortly.

Server Index

Terms Document IDs

c18d3a0	CipherTxt
ccb215a...	\$2b\$12...
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

Searchable Symmetric Encryption

Our Implementation

Searchable Symmetric Encryption

Client Indexes

Word 0	c(0)
Word 1	c(1)
Word 2	c(2)
...	...
Word n	c(n)

Word 0	Docs[]
Word 1	Docs[]
Word 2	Docs[]
...	...
Word n	Docs[]

- Client maintains two indexes:
 - One for word->c(word)
 - One for word->list of corresponding documents

Searchable Symmetric Encryption

Client Indexes

Word 0	c(0)
Word 1	c(1)
Word 2	c(2)
...	...
Word n	c(n)

Word 0	Docs[]
Word 1	Docs[]
Word 2	Docs[]
...	...
Word n	Docs[]

- Client maintains two indexes:
 - One for word->c(word)
 - One for word->list of corresponding documents
- When documents are added, the count is increased in the first index, and the document lists are appended according in the second index.
 - This provides a performance benefit later when performing a search.

Searchable Symmetric Encryption

Server Index

c18d3a0	CipherTxt
ccb215a...	\$2b\$12...
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

- Server maintains single index of mappings
 - $\text{PRF}(w \parallel c(w)) \rightarrow \text{Enc}(\text{Doc ID})$

Searchable Symmetric Encryption

UPDATE

SSE: Update (Client)

Client Indexes

Word 0	c(0)
Word 1	c(1)
Word 2	c(2)
...	...
Word n	c(n)

Algorithm

- Input: File “Message1” (email)

Word 0	Docs[]
Word 1	Docs[]
Word 2	Docs[]
...	...
Word n	Docs[]

SSE: Update (Client)

Client Indexes

Word 0	$c(0)$
Word 1	$c(1)$
Word 2	$c(2)++$
...	...
Word n	$c(n)$

Algorithm

- Input: File "Message1" (email)
- Parse text in file body and header and for each word:
 - If in index 1, increment $c(w)$



SSE: Update (Client)

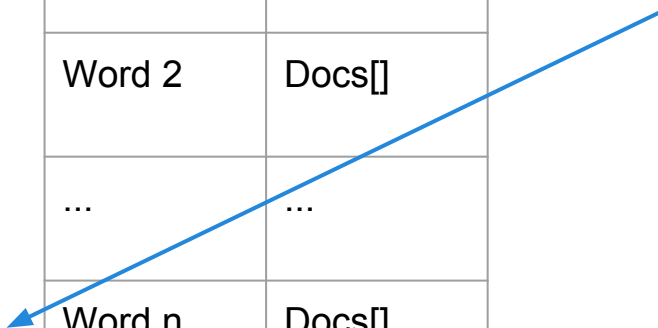
Client Indexes

Word 0	$c(0)$
Word 1	$c(1)$
Word 2	$c(2)$
...	...
Word $n+1$	$c(n+1) = 1$

Word 0	Docs[]
Word 1	Docs[]
Word 2	Docs[]
...	...
Word n	Docs[]

Algorithm

- Input: File "Message1" (email)
- Parse text in file body and header and for each word:
 - If in index 1, increment $c(w)$
 - Else, add to index 1



SSE: Update (Client)

Client Indexes

Word 0	$c(0)$
Word 1	$c(1)$
Word 2	$c(2)$
...	...
Word $n+1$	$c(n+1) = 1$

Word 0	Docs[]
Word 1	Docs[]
Word 2	Docs[]++
...	...
Word n	Docs[]

Algorithm

- Input: File "Message1" (email)
- Parse text in file body and header and for each word:
 - If in index 1, increment $c(w)$
 - Else, add to index 1
 - If in index 2, append file ID to Docs list



SSE: Update (Client)

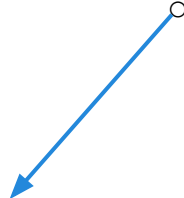
Client Indexes

Word 0	$c(0)$
Word 1	$c(1)$
Word 2	$c(2)$
...	...
Word $n+1$	$c(n+1) = 1$

Word 0	Docs[]
Word 1	Docs[]
Word 2	Docs[]++
...	...
Word $n+1$	Docs[]++

Algorithm

- Input: File “Message1” (email)
- Parse text in file body and header and for each word:
 - If in index 1, increment $c(w)$
 - Else, add to index 1
 - If in index 2, append file ID to Docs list
 - Else add to index 2



SSE: Update (Client)

Client Indexes

Word 0	$c(0)$
Word 1	$c(1)$
Word 2	$c(2)$
...	...
Word $n+1$	$c(n+1) = 1$

Word 0	Docs[]
Word 1	Docs[]
Word 2	Docs[]++
...	...
Word $n+1$	Docs[]++

Algorithm

- Input: File “Message1” (email)
- Parse text in file body and header and for each word:
 - If in index 1, increment $c(w)$
 - Else, add to index 1
 - If in index 2, append file ID to Docs list
 - Else add to index 2
- $k1 = \text{PRF}(\text{password}, 1 \parallel w)$
 $k2 = \text{PRF}(\text{password}, 2 \parallel w)$

SSE: Update (Client)

Client Indexes

Word 0	$c(0)$
Word 1	$c(1)$
Word 2	$c(2)$
...	...
Word $n+1$	$c(n+1) = 1$

Word 0	Docs[]
Word 1	Docs[]
Word 2	Docs[]++
...	...
Word $n+1$	Docs[]++

Algorithm

- Input: File “Message1” (email)
- Parse text in file body and header and for each word:
 - If in index 1, increment $c(w)$
 - Else, add to index 1
 - If in index 2, append file ID to Docs list
 - Else add to index 2
- $k1 = \text{PRF}(\text{password}, 1 \parallel w)$
 $k2 = \text{PRF}(\text{password}, 2 \parallel w)$
- $l = \text{PRF}(k1, c)$
 $d = \text{Enc}(k2, \text{file ID})$

SSE: Update (Client)

Client Indexes

Word 0	$c(0)$
Word 1	$c(1)$
Word 2	$c(2)$
...	...
Word $n+1$	$c(n+1) = 1$

Word 0	Docs[]
Word 1	Docs[]
Word 2	Docs[]++
...	...
Word $n+1$	Docs[]++

Algorithm

- Input: File “Message1” (email)
- Parse text in file body and header and for each word:
 - If in index 1, increment $c(w)$
 - Else, add to index 1
 - If in index 2, append file ID to Docs list
 - Else add to index 2
 - $k1 = \text{PRF}(\text{password}, 1 \parallel w)$
 $k2 = \text{PRF}(\text{password}, 2 \parallel w)$
 - $l = \text{PRF}(k1, c)$
 $d = \text{Enc}(k2, \text{file ID})$
 $l_{\text{prime}} = \text{PRF}(k1, c-1)$
- Send list with each (l, d, l_{prime}) and send $\text{Enc}(\text{“Message1”})$ to server

SSE: Update (Server)

Server Index

l

d

c18d3a0	CipherTxt
ccb215a...	\$2b\$12...
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

Algorithm

- Receives:
 - $L = [(l_0, d_0, l_{\text{prime}_0}), (l_1, d_1, l_{\text{prime}_1})...]$
 - Encrypted message.

SSE: Update (Server)

Server Index

l

d

c18d3a0	CipherTxt
ccb215a...	\$2b\$12...
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

Algorithm

- Receives:
 - $L = [(l_0, d_0, l_{\text{prime}_0}), (l_1, d_1, l_{\text{prime}_1})...]$
 - Encrypted message.
- For w in L :
 - If l_{prime} in index, delete that entry
 - Add each (l, d) to index

Searchable Symmetric Encryption

SEARCH

SSE: Search (Client)

Client Indexes

Word 0	c(0)
Word 1	c(1)
Word 2	c(2)
...	...
Word n	c(n)

Algorithm

- Input list of search terms $W[]$

Word 0	Docs[]
Word 1	Docs[]
Word 2	Docs[]
...	...
Word n	Docs[]

SSE: Search (Client)

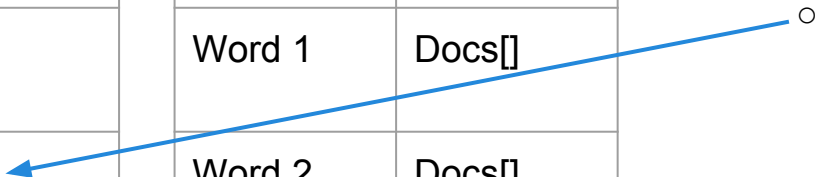
Client Indexes

Word 0	c(0)
Word 1	c(1)
Word 2	c(2)
...	...
Word n	c(n)

Algorithm

Word 0	Docs[]
Word 1	Docs[]
Word 2	Docs[]
...	...
Word n	Docs[]

- Input list of search terms $W[]$
- For each w in $W[]$:
 - If w in index 1 (ie: it exists in some message), get c



SSE: Search (Client)

Client Indexes

Word 0	c(0)
Word 1	c(1)
Word 2	c(2)
...	...
Word n	c(n)

Word 0	Docs[]
Word 1	Docs[]
Word 2	Docs[]
...	...
Word n	Docs[]

Algorithm

- Input list of search terms $W[]$
- For each w in $W[]$:
 - If w in index 1 (ie: it exists in some message), get c
 - Else, $c = \text{None}$

SSE: Search (Client)

Client Indexes

Word 0	c(0)
Word 1	c(1)
Word 2	c(2)
...	...
Word n	c(n)

Word 0	Docs[]
Word 1	Docs[]
Word 2	Docs[]
...	...
Word n	Docs[]

Algorithm

- Input list of search terms $W[]$
- For each w in $W[]$:
 - If w in index 1 (ie: it exists in some message), get c
 - Else, $c = \text{None}$
 - $k1 = \text{PRF}(\text{password}, 1 \parallel w)$
 $k2 = \text{PRF}(\text{password}, 2 \parallel w)$

SSE: Search (Client)

Client Indexes

Word 0	c(0)
Word 1	c(1)
Word 2	c(2)
...	...
Word n	c(n)

Word 0	Docs[]
Word 1	Docs[]
Word 2	Docs[]
...	...
Word n	Docs[]

Algorithm

- Input list of search terms $W[]$
- For each w in $W[]$:
 - If w in index 1 (ie: it exists in some message), get c
 - Else, $c = \text{None}$
 - $k1 = \text{PRF}(\text{password}, 1 \parallel w)$
 $k2 = \text{PRF}(\text{password}, 2 \parallel w)$
- Send list with each $(k1, k2, c)$ to server

SSE: Search (Server)

Server Index

<u>l</u>	<u>d</u>
c18d3a0	CipherTxt
ccb215a...	\$2b\$12...
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

Algorithm

- Receives:
 - $L = [(k1_0, k2_0, c_0), (k1_1, k2_1, c_1) \dots]$

SSE: Search (Server)

Server Index

<u>l</u>	<u>d</u>
c18d3a0	CipherTxt
ccb215a...	\$2b\$12...
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

Algorithm

- Receives:
 - $L = [(k1_0, k2_0, c_0), (k1_1, k2_1, c_1) \dots]$
- For w in L :
 - $F = \text{PRF}(k1, c)$ // for example: ccb215a...

SSE: Search (Server)

Server Index

<u>l</u>	<u>d</u>
c18d3a0	CipherTxt
ccb215a...	\$2b\$12...
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

Algorithm

- Receives:
 - $L = [(k1_0, k2_0, c_0), (k1_1, k2_1, c_1)...]$
- For w in L :
 - $F = \text{PRF}(k1, c)$ // for example: ccb215a...
 - $d = \text{index}[F]$ // \$2b\$12...



SSE: Search (Server)

Server Index

<u>l</u>	<u>d</u>
c18d3a0	CipherTxt
ccb215a...	\$2b\$12...
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

Algorithm

- Receives:
 - $L = [(k1_0, k2_0, c_0), (k1_1, k2_1, c_1) \dots]$
- For w in L :
 - $F = \text{PRF}(k1, c)$ // for example: ccb215a...
 - $d = \text{index}[F]$ // \$2b\$12...
 - If not d , not in index
 - $D.\text{append}(d)$

SSE: Search (Server)

Server Index

<u>l</u>	<u>d</u>
c18d3a0	CipherTxt
ccb215a...	\$2b\$12...
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

Algorithm

- Receives:
 - $L = [(k1_0, k2_0, c_0), (k1_1, k2_1, c_1)...]$
- For w in L :
 - $F = \text{PRF}(k1, c)$ // for example: ccb215a...
 - $d = \text{index}[F]$ // \$2b\$12...
 - If not d , not in index
 - $D.\text{append}(d)$
- For d in D :
 - $m = \text{DEC}(k2, d)$ // decrypts names/IDs
 - $M.\text{append}(m)$ // append to list of names/IDs

SSE: Search (Server)

Server Index

<u>l</u>	<u>d</u>
c18d3a0	CipherTxt
ccb215a...	\$2b\$12...
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

Algorithm

- Receives:
 - $L = [(k1_0, k2_0, c_0), (k1_1, k2_1, c_1)...]$
- For w in L :
 - $F = \text{PRF}(k1, c)$ // for example: ccb215a...
 - $d = \text{index}[F]$ // \$2b\$12...
 - If not d , not in index
 - $D.\text{append}(d)$
- For d in D :
 - $m = \text{DEC}(k2, d)$ // decrypts names/IDs
 - $M.\text{append}(m)$ // append to list of names/IDs
- Send $M[]$ back to client

SSE: Search (Server)

Server Index

l

d

c18d3a0	CipherTxt
ccb215a...	\$2b\$12...
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

Algorithm

- Server can either send encrypted messages back (now that it has decrypted the IDs and knows which have been requested).
 - The current implementation of our scheme

SSE: Search (Server)

Server Index

l

d

c18d3a0	CipherTxt
ccb215a...	\$2b\$12...
ee2064	CipherTxt
...	...
7b13cb	CipherTxt

Algorithm

- Server can either send encrypted messages back (now that it has decrypted the IDs and knows which have been requested).
 - The current implementation of our scheme
- Or, server can send back list of IDs, and the client can send a separate request for the messages themselves.

Searchable Symmetric Encryption

Issues/Concerns:

- Update sends 'lprime' across with 'l'. This means it's giving up the PRF of the term with 'c' and 'c-1'. Could be an issue.
- Search sends 'c' in the clear. Server and listeners would know the count of the hash value, but we don't think that is an issue.
 - Performance is abysmal without sending 'c' with query.
- How to implement encryption when the message originates from outside of the client? Won't have same password for PRF.