

# Encrypted Search

Senior Thesis II  
May 2015

*by Jimmy Swanbeck*

# Table of Contents

Background

Related work

Problem/Goals

Solution

Results

Conclusion

# Encrypted Search



Searching against encrypted information

Everything happens server-side

Secure way of searching through ciphertext

No need for client-side decryption

Encrypted search can be compatible with a cloud server

# The Cloud

Many new software releases today are cloud-based

Email providers store their data in the cloud, allowing you to access it from any location

Security is the main inhibitor of companies adopting cloud storage

If the cloud server is compromised, everybody's data is compromised at the same time



# Encryption

Every character in a character set (ASCII, Unicode, etc.) is assigned a “code point”

This is a unique number used to identify the *character* (not the letter)

ASCII:        ‘A’ = 65        ‘a’ = 97

The code point is the starting point for any encryption algorithm

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	\$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

# Normalization

Remove capitalization, punctuation, etc.

“Sample...text” → “SAMPLETEXT”

When you send a server your search term, it normalizes the text and matches it against text in its database which has already been normalized when it was stored

Doesn't work on encrypted text because once it's encrypted there is no connection between the plaintext and the ciphertext

“text” = “VqnpDGjH”      “Text” = “5g+UNPod”

# How Searching Works

Send a search term “Text” to the server

Client side normalizes it, sends it to the server, the server searches through its database to find matches



Search Term → Normalized Search Term  
Text → TEXT



**Problem:** This will not work with encrypted data as you cannot normalize a string after it has been encrypted

# Client-Side Searching

## **Existing Solution:**

- Client-side decryption

- Involves downloading everything from the server, decrypting it, and then searching

- Not technically searching over encrypted data, rather searching over decrypted data

## **Problems:**

- Defeats the purpose of a cloud server

- Not practical for large data sets



# Related Work

## A Better Solution:

The creation of an **encrypted index**

This involves normalizing before encrypting, then encrypting each word separately to allow a search engine to scan through it

References: 1, 2, 3, 4, 5, 7, 8, 9

# Other Things to Consider

## Perfecting algorithms

- Ranked keyword search

- Conjunctive keyword search

- Fuzzy keyword search

- Wildcard search

## Loss of security

# Related Work

## Fuzzy Keyword Search

- Add support for common misspellings

  - Specifically one letter errors (add, drop, or replace)

- Will increase the size of the index, potentially by a huge amount

- Need to find a balance between functionality and the index's increase in size

References: 1

# Related Work

CASTLE:

{AASTLE, BASTLE, ... YASTLE, ZASTLE},  
{CBSTLE, CCSTLE, ... CYSTLE, CZSTLE},  
{CAATLE, CABTLE, ... CAYTLE, CAZTLE},  
etc.

# Related Work

This particular method would add  $n(25n + 1)$  letters to the index for every word of length  $n$

This example only accounts for character dropping and replacement (not addition)

For a word of length 6, that's 906 additional letters to add to the index

Obviously **not practical**, but this is what fuzzy keyword search is trying to accomplish

# Related Work

## Wildcard Search

Search for any words that contain a specified set of letters

ex. Searching “Cas\*le” will return “Castle” as well as any other words that start with “cas” and end with “le”

References: 1

# Related Work

CASTLE:

{CASTLE, \*CASTLE, \*ASTLE, C\*ASTLE,  
C\*STLE, CA\*STLE, CA\*TLE, CAS\*TLE,  
CAS\*LE, CAST\*LE, CAST\*E, CASTL\*E,  
CASTL\*, CASTLE\*}

# Related Work

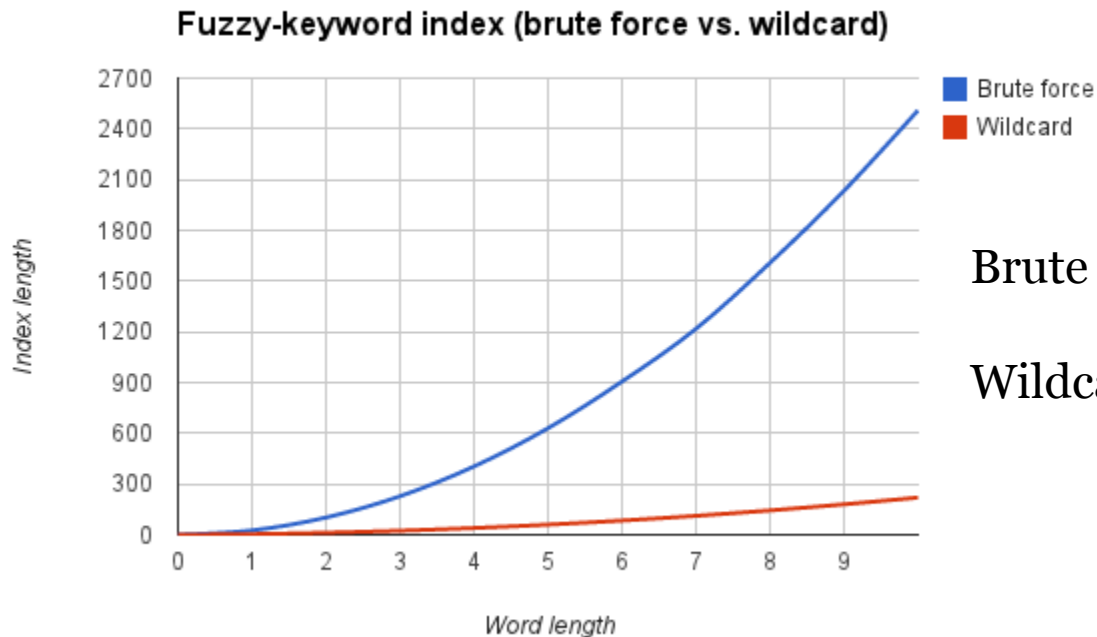
Using just the previous method will increase the index by  $n(2n+2)$  for every word of length  $n$

CASTLE, a word of length 6, would add 84 additional letters to the index

While this isn't perfect, it does solve the problem with fuzzy keyword search



# Related Work



Brute force:  $n(25n + 1)$

Wildcard:  $n(2n+2)$

# Problem/Goals

The goal of this project has been to combine case-insensitive search (encrypted index) and wildcard support to create a unique form of encrypted search that is compatible with a cloud server.

# Solution

First, scan through each word in the email and create a normalized index

- Hash each word separately

- Will consider case-insensitivity as well as wildcard search

Next, encrypt the body of the email message

- Do this all as one string (will minimize information leakage)

# Building the Index

Here is a sample email message.

# Building the Index

Here is a sample email message.

**HERE**

# Building the Index

Here is a sample email message.

**HERE**

# Building the Index

Here is a sample email message.

**HERE**

# Building the Index

Here is a sample email message.

**HERE SAMPLE**



# Building the Index

Here is a sample email message.

HERE SAMPLE EMAIL

# Building the Index

Here is a sample email message.

HERE SAMPLE EMAIL MESSAGE

# Building the Index

Here is a sample email message.

HERE SAMPLE EMAIL MESSAGE

# Algorithms

Fuzzy-keyword search only supports single letter misspellings (add, drop, and replace), not full wildcard support (e.g. “cas\*” returning “castle”)

Duplicates and words less than 3 characters will not be added to the index

Will check against a list of common 3-character words to exclude from the index

# Algorithms

*Demonstration*

# Architecture

Two applications:

Web app:

- Programmed in Python, HTML, CSS, and Javascript

- Can perform operations to store, view, and search documents

- Deals with non-encrypted data (acts as project control and to help visualize data)

Client app:

- Programmed in Visual C#

- Can perform operations to store and search documents

- Deals with encrypted data (acts as proof of concept)

# Architecture

Two virtual machines:

- Linux VM runs the server and database

- Windows VM runs the client application

Web application uses HTTP to perform GET and POST operations

Client application connects directly to MySQL database

# Conclusion

Concept was proven to work

My takeaways from the project:

- Engineering for security

- HTTP/RESTful APIs

- Full-stack development

Achieved my main goals for the project (case-insensitive, fuzzy-keyword), but didn't reach a couple of secondary goals



# Future Work

## Ranked Keyword Search

Record number of matched keywords in a message

Order the search results by rank (e.g. sort by relevance)

References: 2, 3, 4, 9

# Future Work

## Conjunctive Keyword Search

Take a string of words as search input

Search each word individually (unless multiple words are enclosed by quotation marks) and give results based on the highest number of matched keywords

Extension of ranked search

References: 3, 4

*Questions*

# References

1. Li, Jin, et al. "Fuzzy keyword search over encrypted data in cloud computing." *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010.
2. Wang, Cong, et al. "Secure ranked keyword search over encrypted cloud data." *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*. IEEE, 2010.
3. Cao, Ning, et al. "Privacy-preserving multi-keyword ranked search over encrypted cloud data." *Parallel and Distributed Systems, IEEE Transactions on* 25.1 (2014): 222-233.
4. Golle, Philippe, Jessica Staddon, and Brent Waters. "Secure conjunctive keyword search over encrypted data." *Applied Cryptography and Network Security*. Springer Berlin Heidelberg, 2004.
5. Li, Ming, et al. "Authorized private keyword search over encrypted data in cloud computing." *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011.

# References

6. Boneh, Dan, et al. "Public key encryption with keyword search." *Advances in Cryptology-Eurocrypt 2004*. Springer Berlin Heidelberg, 2004.
8. Lu, Wenjun, et al. "Enabling search over encrypted multimedia databases." *IS&T/SPIE Electronic Imaging*. International Society for Optics and Photonics, 2009.
10. Song, Dawn Xiaoding, David Wagner, and Adrian Perrig. "Practical techniques for searches on encrypted data." *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*. IEEE, 2000.
12. Swaminathan, Ashwin, et al. "Confidentiality-preserving rank-ordered search." *Proceedings of the 2007 ACM workshop on Storage security and survivability*. ACM, 2007.
14. Goh, Eu-Jin. "Secure Indexes." *IACR Cryptology ePrint Archive* 2003 (2003): 216.