Preserving Data Privacy with Searchable Symmetric Encryption

Shaun Mc Brearty, William Farrelly, ¥Kevin Curran

Letterkenny Institute Of Technology, Co. Donegal, Ireland
¥Ulster University, Northern Ireland

Abstract - New techniques such as Searchable Encryption are being deployed to enable data to be encrypted online. Searchable Encryption is now at the point that it can be deployed and used within the Cloud. In the Cloud, Searchable Encryption has the ability to allow CSP customers to store their data in encrypted form, while retaining the ability to search that data without disclosing the associated decryption key(s) to CSPs that is, without compromising data security on the Server. We present an SSE scheme and evaluate the efficiency of storing and retrieving data from the cloud. The results showed that carrying out a task using SSE is directly proportional to the amount of information involved.

Keywords - Security, Encryption, Cloud Computing, Search, WWW

I. INTRODUCTION

The benefits of Cloud computing are significant: reduced costs, high reliability, as well as the immediate availability of additional computing resources as and when needed. Despite such advantages, Cloud Service Provider (CSP) consumers need to be aware that the Clouds poses its own set of unique risks that are not typically associated with storing and processing one's own data internally using privately owned infrastructure [1, 2]. Perhaps the most severe risk facing CSP consumers at present is the threat of data disclosure or data loss. Recent years have seen a number of such incidents occur, whereby organisations customer data – hosted on the Cloud - has been leaked online (for hacktivism or vandalism purposes) or stolen for criminal purposes. Cloud computing is made possible through the use of many technologies, including internet access, virtualisation and third party data centres.

In the case of online access to the CSP, such access controls typically take the form of usernames and passwords; In the case of virtualisation, such access controls typically take the form of logical data separation; and in the case of third party data centres, such access controls typically take the form of physical access controls (*For Example: Locks, Keypads*) (as well as software based access control) that prevent unauthorised CSP personnel from gaining access to user data. In principle, all of the aforementioned access controls are sound; however in practice, such controls have been circumvented. In the event that any of the aforementioned access controls are compromised maliciously, the chances of a data breach occurring are high. Should a data breach occur and the associated data is retrieved in encrypted form, the data is essentially useless to an attacker (unless the encryption algorithm utilised is weak and/or the attacker has some foreknowledge of the associated decryption key) [2, 3]; however, in the event that a data breach occurs and the associated data is retrieved in plaintext form, an organisations worst nightmare has become a reality. What follows is typically a slew of press releases, negative publicity, damaged business reputations, and fines under various data protection laws [4, 5] . To reduce the impact of potential data breaches (and to provide privacy for CSP consumer data) CSPs typically employ the use of cryptography. In a Cloud environment, cryptography is typically utilised for two purposes: security while data is at rest; and security while data is in transit. Unfortunately the Cloud cannot guarantee the security of data during processing as the current limitations of cryptography prevent data from being processed in encrypted form. Given the fact that data is processed in unencrypted form, it is quite common for attackers to target data in use, rather than targeting data which is encrypted during storage and transit. An entity wishing to store its data within the Cloud must choose to (1) Store Data in Encrypted Form or Store Data in Unencrypted Form. If storing data in encrypted form then 2 Options exist which are to 1. Disclose Decryption Key(s) to Cloud Service Provider (CSP) or 2. Keep Decryption Key(s) Private.

Option 1A requires encrypted data owners to disclose their decryption key(s) to CSPs. This is due to the fact that data cannot be searched or operated on while in encrypted form. In order to provide CSP customers with such functionality, CSPs require access to the necessary decryption key(s). Option 1B (Keeping Decryption Key(s) Private) represents the most secure sub-option; however, as previously mentioned, CSP customers lose the ability to search or operate on their data while it is in encrypted form. In order to utilise such functionality using Option 1B, CSP customers must download their data, decrypt it, and only then can it be searched and/or operated on. While this approach may be fine for small amounts of data, it becomes increasingly inefficient and unwieldy as the amount of data increases. In addition, should any changes be made to the data after it has been downloaded; the customer must then re-encrypt and re-uploaded the entire dataset to the Cloud. Option 2 avoids the use of encryption for data security. Rather than relying on cryptography for data security; *that is, the traditional approach to data security*, this approach utilises the aforementioned approach of logically separating data

[6]. Evidently, none of the options available at present provide an adequate balance of data security and functionality. Option 1A and Option 2 offer full functionality at the expense of data security, while Option 1B provides data security at the expense of any and all functionality. The ideal solution to achieving an optimal balance of data security and functionality within the Cloud involves the CSP having the ability to search and operate on data while it is in encrypted form – without having any knowledge of the associated decryption key(s), or the associated plaintext(s) [6].

SSE represents one of the few forms of Searchable Encryption that is achievable using established standardised encryption algorithms. Alternative forms of Searchable Encryption require the use of non-standardised, special purpose encryption algorithms [7]. SSE is considered one of the least secure forms of Searchable Encryption (see figure 1) primarily due to Information Leakage [9, 11]. Solutions exist to eradicate and obfuscate all forms of Information Leakage in SSE; however existing solutions have a significant effect on the search efficiency of SSE [18]. Evidently, the challenge for researchers is to improve the security of SSE while maintaining its superior search efficiency. Figure 1 lists all known solutions to the problem of searching on encrypted data; *that is, symmetrically encrypted data, as well as public key encrypted data*. The y-axis of figure 1 lists all Searchable Encryption solutions with respect to their efficiency, while the x-axis lists all solutions with respect to security. As regards efficiency, the SSE literature defines efficiency as the time-complexity associated with finding a given Encrypted Search String (ESS) within a body of encrypted data (expressed in Big O Notation).

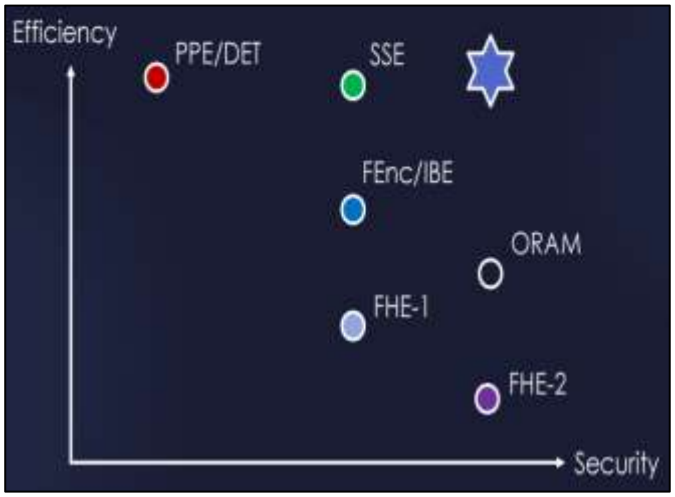


Figure 1: Efficiency Vs. Security Trade-off For SE Schemes (Kamara 2013)

In terms of security, the SSE literature defines security as the amount of Information Leakage associated with using a given Searchable Encryption scheme; *that is, what the Server learns (or can deduce) about the ciphertext by searching over it* (expressed in Terms of the numerous categories of Information Leakage) [19].

II. SEARCHABLE ENCRYPTION

Searchable Encryption operates on the assumption that a given Term - whether in plaintext form or encrypted form - is located in the same position in both the plaintext version of the Document and the encrypted version of the same Document. *For Example: Given a plaintext Document beginning with the Term 'The', the description provided by [8] assumes that the first three characters of both the plaintext version of the Document and the encrypted version of the Document correspond to the Term 'The'*. Essentially this description assumes that symmetric ciphers encrypt data one character at a time, when in reality, this is not the case. Modern symmetric ciphers encrypt data in blocks of a fixed size, rather than character by character [19]. The effect of using such ciphers is that the ciphertext associated with a given plaintext Term is spread across the entire ciphertext block, rather than appearing in the same position as the plaintext Term; thus preventing traditional Sequential Searching. In addition, modern symmetric ciphers typically operate using advanced block cipher modes (another mechanism to counter cryptanalysis) which 'chain' the ciphertext of previously encrypted blocks to the current plaintext block (by means of a bitwise XOR operation); thus further complicating the problem of searching ciphertext for the presence of an encrypted version of a plaintext Search String. Recognising the inherent difficulty in achieving Searchable Encryption as originally described by [8], subsequent work in the area focussed on developing solutions to the problem as originally conceived; albeit without actually using Sequential Searching [16]. Specifically, researchers focussed on adapting the Inverted Index – a mechanism that has been used in plaintext Information Retrieval for decades – for use in Searchable Encryption [12, 13]. In its most basic form, an Inverted Index is a Data Structure that maps Terms to the Document(s) they occur in; therefore eradicating the need to Sequentially Search Documents [15]. When adapted for use with an encrypted Document Collection, the resulting Inverted Index is titled Searchable Symmetric Encryption (SSE) [12, 14].

The topic of Information Leakage forms an Integral part of SSE. When the idea of Searchable Encryption was first proposed, one of its founding principles was the assumption that the Server storing the encrypted Document Collection is an adversary that is actively working on subverting the security of the Document Collection it possesses (with the ultimate goal of gaining access to the Document Collection in plaintext form) [8]. As such, the SSE Inverted Index is constructed and operates in a manner that takes significant steps to reduce the Leakage of potentially useful Information to the Server. In practice, this involves the use of encryption for the Document Collection, the Lexicon, Posting Lists and Search Strings; as well as the use of Data Structures that hinder the Servers efforts in achieving its malicious goals [12,16].

Responsibility for creating the SSE Inverted Index is offloaded to the Client. In order for the Server to construct the SSE Inverted Index, decryption keys must be disclosed to the Server (as mentioned previously, this is undesirable from a data security perspective). Rather than reveal sensitive information to the Server, SSE delegates responsibility of constructing the SSE Inverted Index to the Client. Given that the Client is responsible for constructing the SSE Inverted Index, it is therefore expected that the Client forwards the SSE Inverted Index to the Server along with the encrypted Document Collection whenever the latter is forwarded to the Server for storage [16].

The steps involved in constructing an SSE Inverted Index are exactly the same as those involved in constructing an IR Inverted Index, albeit the Client has responsibility for generating the SSE Inverted Index, and various forms of encryption are applied to each dataset after they have been compiled; *that is, the Document Collection, the Lexicon and the Postings List* [16, 17]. In addition to the use of encryption, a different Data Structure – namely, an Array - is utilised to store Postings instead of a Linked List (as is used in the IR Inverted Index) [12].

An Inverted Index as typically utilised in plaintext Information Retrieval (IR) contains Data Structures commonly used to store the three data sets that make up the Inverted Index, as well as what form of computer memory is typically used to store each Data Structure.

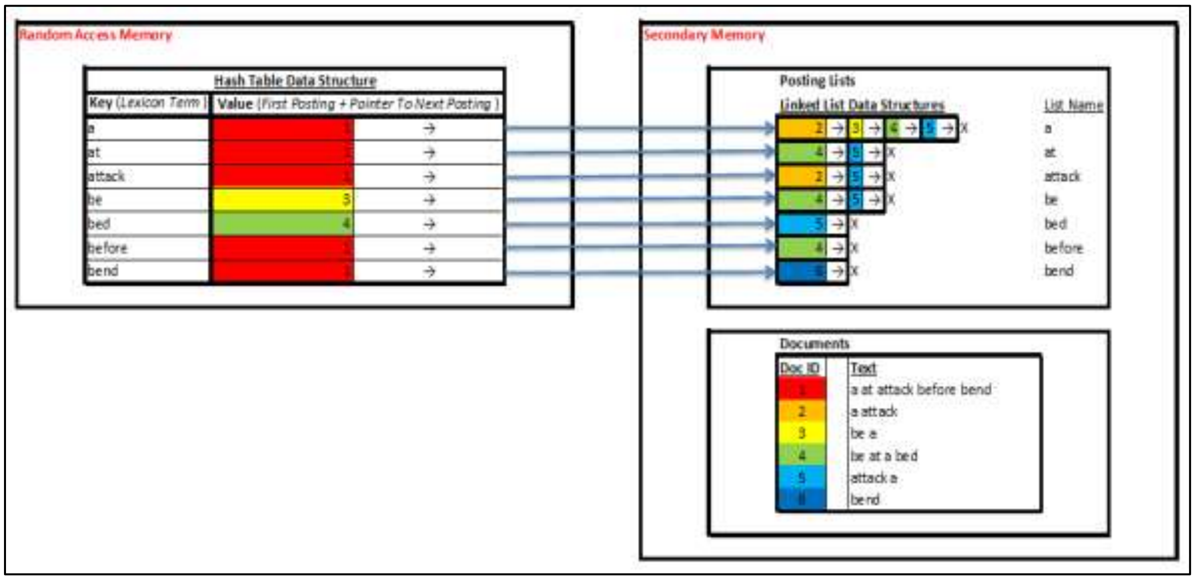


Figure 2: Inverted Index Visualisation (Including Data Structures and Memory

Rather than storing Lexicon Terms in plaintext form, SSE requires that a keyed-hash of each Term be stored instead [10, 20]. The use of a keyed hash function for this purpose - instead of traditional reversible encryption - may seem curious at first; however researchers have successfully argued that the Lexicon's sole purpose within the Inverted Index is to provide the Client with the ability to carry out searches and nothing more. Given that the Lexicon is unlikely to be downloaded to the Client (and is therefore unlikely to be decrypted - unlike the actual Documents), the use of reversible encryption for encrypting Lexicon Terms has largely been abandoned. Aside from the aforementioned reasons, the use of a keyed hash function for this purpose has a number of advantages in terms of reduced Information Leakage and improved data security, including the following [19].

- First and foremost, the use of a hash function (keyed or non-keyed) ensures that all encrypted Lexicon Terms within the SSE Inverted Index are of equal length (*a hash function produces a Hexadecimal String of fixed length*); therefore masking the length of all underlying plaintext Lexicon Terms.

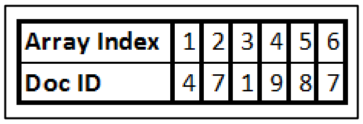
- Secondly, the use of a hash function (again, keyed or non-keyed) ensures that an adversary has no means of decrypting the encrypted Lexicon Term back to its plaintext form.

- Thirdly, ensuring that a keyed hash function is used – instead of a traditional non-keyed hash function – protects SSE from Rainbow Table Attacks; *that is, pre-computed Hash Values of common Dictionary Words*.

The use of Linked Lists for Posting List storage is abandoned in SSE due to Setup Leakage resulting from their modus operandi; *that is, sequential memory access*, with Arrays being preferred instead [12].

Specifically, given the first Link in a Linked List, it is a trivial process to examine all subsequent links due to the fact that each Link in a Linked List contains a pointer to the next Link. Given that each Term in an IR Inverted Index has its own dedicated Linked List to store Postings; it is therefore a trivial process to derive the Term-Document Frequency (TDF) for each Term in the Lexicon in advance of the associated Term being searched for.

Rather than using one Array for each Term in the Lexicon (doing so would also result in TDF Storage Leakage; *that is, the size of the Array would be equivalent to the TDF*), SSE utilises a single one dimensional Array to store all Postings for all Terms (see figure 3). Utilising this approach, Setup Leakage amounts to the total number of Postings for the entire Lexicon*; that is, trivial Leakage*.



Array Index	1	2	3	4	5	6
Doc ID	4	7	1	9	8	7

Figure 3: Postings Stored In an Array.

Given that all Postings are now stored in a single one dimensional Array, some mechanism to keep track of what Postings belong to what Terms is therefore required. The solution to this problem is relatively similar to a Linked List, albeit the solution involved does not utilise pointers (as is the case with Linked Lists). In order to keep track of what Postings are associated with a given Term, SSE requires that the Document ID of the first Posting associated with a given Term is stored alongside the keyed-hash of the Term in the Lexicon Hash Table (in RAM) (*For Example: Doc ID 1*). Alongside this Document ID (in the Lexicon Hash Table) is an Array Index denoting the location of the second Posting associated with the Term (*For Example: 94*). At the Array Index in question is the Document ID of the 2nd Posting, as well as the Array Index denoting the location of the third Posting (*For Example: 79*).

From the Research Results presented in [10] it is apparent that the search time associated with SSE is impressive – to the point that one could argue SSE is efficient enough to be deployed in a Cloud environment. In addition, the work of Cash *et al.* (2013) proves that SSE does indeed scale to large Data Sets whilst maintaining its search efficiency, and also has the ability to support Boolean/Conjunctive Queries in an efficient manner whilst maintaining Data/Query Privacy. Despite such impressive Results, we believe both papers focussed on the performance of a single component of SSE; *that is, searching an SSE Inverted Index*, and not SSE as a whole. Specifically, the author feels that both papers have glossed over the topic of SSE Inverted Index Construction. Given that constructing an SSE Inverted Index is a necessary pre-requisite to searching an SSE Inverted Index; the author feels the topic deserves significantly more attention than that which it has been given in the published literature thus far.

[10] cover the topic briefly, however as indicated previously, the Results presented are somewhat skewed by the fact they only include the Results of converting a pre-existing IR Inverted Index into an SSE Inverted Index – the Results do not include the time taken to generate the initial IR Inverted Index. [11] make no mention of the time taken to generate the SSE Inverted Index used in their work. In addition to largely ignoring the process of constructing an SSE Inverted Index, both papers have also ignored the process of transferring the SSE Inverted Index and the encrypted Document Collection from the Client to the Server.

As [10] correctly points out, the time taken to transfer both the SSE Inverted Index and the encrypted Document Collection from the Client to the Server will vary depending on the underlying system [10] failed to cover this part of SSE for this reason); however the author personally feels that the same can also be argued in relation to cryptographic operations (which are of course reported on in detail in both implementations).

When discussing their Results in relation to searching an SSE Inverted Index, both Kamara *et al.* (2012) and Cash *et al.* (2013) readily acknowledge that their Results only cover searching the SSE Inverted Index and decrypting the Postings associated with the Lexicon Term being searched – their Results do not include the time associated with retrieving and forwarding matching Documents to the Client – another essential component of SSE.
In addition to their failure to examine SSE as a whole, the author is also somewhat disappointed in the quality of information relating to the Test Data Sets and findings of both papers. In relation to Test Data, table 1 summarises the Test Data statistics published (and not published) in both papers.

Information Disclosed	**Kamara *et al.* (2012) [10]**	**Cash *et al.* (2013) [11]**
Number of Documents In Data Set	No	Yes
Number of Terms In Data Set	No	No
Number of Unique Terms In Data Set	No	Yes (Enron Data Set Only)
Number of Postings In Data Set	Yes (Postings In Media File Data Set Not Disclosed)	Yes (Postings In Census Data Set Not Disclosed)
Number of Postings Associated With Highest Frequency Lexicon Term	No	Yes (Not Disclosed For Media File Data Set)
Size of Test Data Set	Yes	Yes (Size Of Census Data Set Not Disclosed)

Table 1: Test Data Statistics

The total number of Terms in the Data Set is relevant in that it dictates the amount of work needed to be performed during Document Tokenisation; *that is, IR Inverted Index Construction*, the number of unique Terms in the Data Set is relevant in that it dictates the number of Terms contained within the Inverted Index (both the IR Inverted Index and the SSE Inverted Index), while the number of Postings in the Data Set is relevant in that it dictates the number of Postings contained within the Inverted Index (both the IR Inverted Index and the SSE Inverted Index).

The number of Postings associated with the highest frequency Lexicon Term is relevant in that the Term in question is typically used to measure the worst case scenario of searching an SSE Inverted Index, while the size of the Test Data Set is relevant in terms of transmitting the Document Collection to the Server from the Client. As can be seen from table 1, a number of these statistics are not disclosed (or are only partially disclosed) by the respective authors; therefore making it difficult to give context to the associated experiment results.

In relation to Inverted Index Construction statistics, Table 2 summarises the Test Data statistics published (and not published) in [10] and [11].

Information Disclosed	**Kamara *et al.* (2012)**	**Cash *et al.* (2013)**
Time Taken To Generate IR Inverted Index	No	No
Size Of IR Inverted Index	No	No
Time Taken To Convert IR Inverted Index To SSE Inverted Index	Yes	No
Size of SSE Inverted Index	No	Yes
Time Taken To Encrypt Document Collection	Yes	No

Table 2: Inverted Index Construction Statistics

The time taken to generate the IR Inverted Index is significant in that the processing time is linear in the number of Terms contained within the Document Collection. The time taken to generate the SSE Inverted Index is significant in that the processing time is linear in the number of Postings contained within the IR Inverted Index, while the size of the SSE Inverted Index is relevant in terms of transmitting the SSE Inverted Index to the Server from the Client.

As can be seen in table 2, neither [10] or [11] disclose any information in relation to IR Inverted Index Construction. When reporting the Results of converting their IR Inverted Index to an SSE Inverted Index, [10] choose to do so by charting their Results against the size of the Test Data Set (in MB)[1]. Personally the author feels this information would be much more informative if it were charted against the number of Postings in the Test Data Set, given that the size of the underlying Data Set in no way reflects the number of unique Terms or Postings in the Data Set. *For Example: a 10MB DOCX file may contain the same Term repeated over and over again; that is, one unique Term => one Posting*. In addition, the author feels that the use of the Document Collection size here is a poor choice given the fact that different file formats can contain the same number of words, but differ greatly in size (such a TXT Files and DOCX Files)[1].

III. EVALUATION

We have therefore identified a number of issues with the information available regarding existing implementations of SSE. The existing SSE literature has failed to cover the whole spectrum of activities associated with SSE [20]. Additionally, the existing published literature has yet to examine the usage of SSE when deployed in a Cloud computing environment. In relation to RQ2, the existing published literature has only compared the performance of SSE with a Database Server, and not a traditional plaintext IR system that utilises an Inverted Index [11].

[1] It should be noted that the chart in question also includes encrypting the associated Document Collection (which is of course dependant on the size of the underlying Document Collection); however the time associated with executing this portion of the task represents only a fraction of the time associated with generating the SSE Inverted Index.

Both software artefacts are examples of personal file hosting applications. Like all file hosting applications, the objective of both the “PlainTXT Storage and Search Engine” and “CipherTXT Storage and Search Engine“ is to allow service users to store their files in the Cloud, and to access/retrieve those files as and when needed (via a web browser). In the case of the “PlainTXT Storage and Search Engine” application, users will be able to store their personal files in plaintext form, as well as having the ability to search and retrieve those files by forwarding queries to the application in plaintext form. In the case of the “CipherTXT Storage and Search Engine” application, users will be provided with the exact same functionality as the “PlainTXT Storage and Search Engine” application, with the exception that both user’s files and queries are encrypted prior to being forwarded to the application for storage/usage.

Given the prototype status of both applications, a number of standard features and functionality typically associated with personal file hosting services have been classified as out of scope for the initial version of both software artefacts. Both the “PlainTXT Storage and Search Engine” and “CipherTXT Storage and Search Engine“ applications were implemented using the Java Programming Language. All Client-Side functionality associated with both applications was implemented in the form of Java Applets, while all Server-Side functionality was implemented in the form of Java Servlets. The SSE scheme underlying the “CipherTXT Storage and Search Engine” application is [10].

The Operating System was Windows Ultimate 64-Bit SP1. The Java Development Kit (JDK) was v.8 and JRE was update 51, build 16. The Web Server (Localhost) was Apache Tomcat 7.0.56. Tests were run on an Intel Core i7 4900MQ @2.8GHz Quad Core laptop with 24GB RAM (3 X 8GB KINGSTON DDR3 @ 800MHz). The Hard Disk was a 925GB SSHD with RAID 1. All tests were conducted using the default Java Virtual Machine (JVM) - no additional runtime parameters were configured. All experiments were performed on the ’20 Newsgroups’ Data Set (Rennie, 2008). In its original form, the ’20 Newsgroups’ Data Set consists of 18,828 files, subdivided into 20 folders. Initially, each file in the Data Set has a numeric file name between 4 and 6 digits in length with no file extension. Prior to being used in the experiments, we first attempted to move all files in the Data Set into a single folder; however at this point we noted that the names of all files in the Data Set are not unique (the contents of each file are unique however [21]. In an effort to avoid duplicate file names, we randomly assigned an 8 digit numeric name to each file in the Data Set. We also appended the TXT file extension to each file in the Data Set. As part of Testing, we tested each aspect of SSE with Data Sets that increased in size by an order of magnitude. As such, it was necessary to derive smaller subsets from the full ’20 Newsgroups’ Test Data Set. In total, 5 subsets were derived (DS1 – DS5). The details associated with each subset – and the full Data Set (DS6) – can be seen in table 6. We present the results associated with SSE Inverted Index Construction, SSE Inverted Index Searching and the comparison of SSE and plaintext Information Retrieval (IR). All results represent average values obtained over ten executions of each experiment.

SSE Inverted Index Construction

The time associated with constructing an IR Inverted Index appears to increase linearly as the number of Terms in the underlying Document Collection increases. In relation to Test Data, an IR Inverted Index was generated for Test Data Set 6 (approximately 5 million Terms) in approximately 7.6 seconds.

Performance of SSE vs. Plaintext IR

We found that the amount of time necessary for SSE uploading increases in a non-linear manner when compared to the amount of time necessary for plaintext IR uploading.

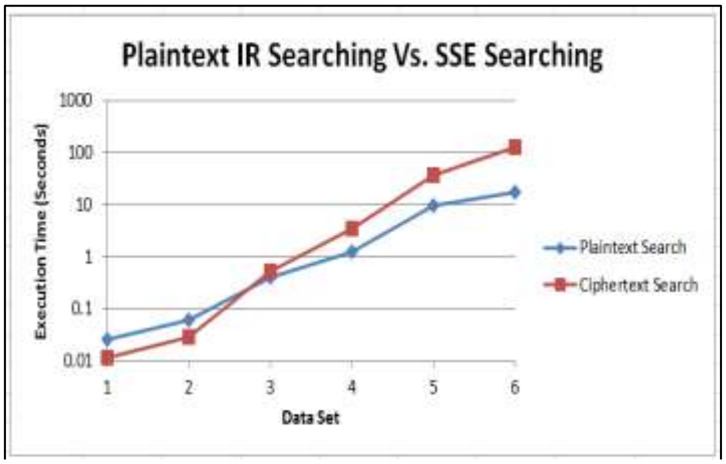


Figure 4: Plaintext IR Querying vs. SSE Querying

Figure 4 denotes the comparison of traditional plaintext Information Retrieval (IR) querying and SSE querying. The Experimental Results presented in figure 4 consist of the time taken to identify the set of all Postings associated with the most frequently occurring Lexicon Term in the underlying Document Collection, and encapsulating the set of all matching Document within a ZIP File which is then returned to the Client. It is obvious from figure 4 that the amount of time necessary for SSE querying increases in a non-linear manner when compared to the amount of time necessary for plaintext IR querying.

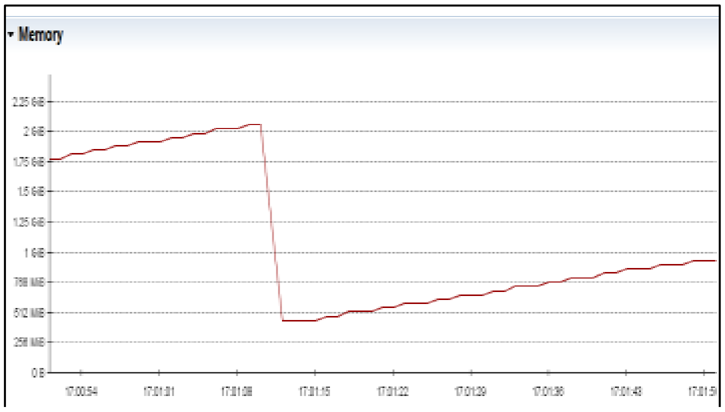


Figure 5: Java Heap Memory Usage and Garbage Collection Statistics for SSE Inverted Index Construction

In relation to searching an SSE Inverted Index, the results provide additional proof of the efficiency of SSE when implemented in software. The implementation of SSE developed as part of this research was able to identify and decrypt a single Posting associated with a given Lexicon Term in approximately 22 microseconds (µs). This performance is comparable with the implementations of SSE developed by Kamara *et al.* (2012) which was 7.3 Microseconds (µs) per Posting and Cash *et al.* (2013) which was 100 Microseconds (µs) per Posting. Regarding the efficiency of constructing an SSE Inverted Index, the results are somewhat inconclusive. Given the five steps involved in constructing an SSE Inverted Index, each step in the implementation of SSE produced as part of this research performed as expected with the exception of the second step: *Converting an IR Inverted Index to an SSE Inverted Index*.

For Test Data Set 1 (DS1) through Test Data Set 4 (DS4), an SSE Inverted Index was generated from an existing IR Inverted Index in a time linear to the number of Postings stored in the IR Inverted Index; however, for DS5 and DS6, this apparent linear performance decreased dramatically. This decrease in performance could be attributed to a combination of one or more of the following: 1) The Java Virtual Machines (JVM) Garbage Collection functionality, 2) Insufficient Java Heap memory, 3) The use of String Objects in the Encrypted_Array_Node Class, 4) The size of the SSE Inverted Index, and 5)

IV. CONCLUSION

The results show that carrying out a task using SSE is directly proportional to the amount of information involved. In the case of constructing an IR Inverted Index, the results show that the time taken to generate an IR Inverted Index is directly proportional to the number of Terms contained in the underlying Document Collection. Converting the same IR Inverted Index to an SSE Inverted Index is directly proportional to the number of Postings contained within the IR Inverted Index, while the time taken to encrypt the underlying Document Collection is directly proportional to the number of Terms contained within the Document Collection. In relation to searching in SSE, the time taken to identify and decrypt the set of Postings associated with a given Lexicon Term is directly proportional to the number of Postings. Regarding the question of whether or not SSE is efficient enough to be deployed in a Cloud environment, the answer is context dependant. If deployed in an environment whereby Search Results only have to be returned to the user in small quantities (such as an Internet Search Engine (*For Example: ten results at a time*)), then SSE would be more than efficient, irrespective of the size of the underlying Data Set (due to the fact that only a small number of Postings would need to be decrypted at a given time). If deployed in an environment whereby all results must be returned at once (as was the case with the implementation of SSE developed as part of this research, SSE would only be suitable for small and medium sized Data Sets.

REFERENCES

[1] Eurostat (2014) Cloud computing - statistics on the use by enterprises http://ec.europa.eu/eurostat/statistics-explained/index.php/Cloud_computing_-_statistics_on_the_use_by_enterprises

[2] Hashizume, K., Rosado, D. G., Fernández-Medina, E. and Fernandez, E. B. (2013) An analysis of security issues for cloud computing http://www.jisajournal.com/content/pdf/1869-0238-4-5.pdf

[3] Nguyen, M., Chau, N., Jung, S. and Jung, S. (2014) A Demonstration of Malicious Insider Attacks inside Cloud IaaS Vendor http://www.ijiet.org/papers/455-F028.pdf

[4] ICO (2015) Monetary Penalty Notice: Staysure.co.uk Limited https://ico.org.uk/media/action-weve-taken/mpns/1043368/staysure-monetary-penalty-notice.pdf

[5] Levick (2015) DATA SECURITY & PRIVACY http://levick.com/experience/specialty/data-security-privacy

[6] Mather, T., Kumaraswamy, S. and Latif, S. (2009) Cloud Security and Privacy, California: O'Reilly.

[7] Gentry, C. (2009) A Fully Homomorphic Encryption Scheme, unpublished thesis (PhD), Stanford University.

[8] Song, D. X., Wagner, D. and Perrig, A. (2000) 'Practical Techniques For Searches On Encrypted Data', in Titsworth, F. M., ed., IEEE Symposium on Security and Privacy, 2000, Berkeley, California, 14-17 May 2000, Washington, D.C.: IEEE Computer Society, 44-55.

[9] Gentry, C., Halvei, S. and Smart, N. P. (2015) Homomorphic Evaluation of the AES Circuit (Updated Implementation) https://eprint.iacr.org/2012/099.pdf

[10] Kamara, S., Papamanthou, C. and Roeder, T. (2012) Dynamic Searchable Symmetric Encryption Proceedings of the 2012 ACM conference on Computer and communications security, pp: 965-976, https://eprint.iacr.org/2012/530.pdf

[11] Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Rosu, M. C. and Steiner, M. (2013) Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries, Crypto 2013, Part 1, LNCS 8042, pp: 353-73

[12] Curtmola, R., Garay, J., Kamara, S. and Ostrovsky, R. (2006) Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions http://eprint.iacr.org/2006/210.pdf

[13] Bosch, C., Hartel, P., Jonker, W. and Peter, A. (2014) A Survey of Provably Secure Searchable Encryption. http://eprints.eemcs.utwente.nl/24788/01/a18-bosch.pdf

[14] Luenberger, D. G. (2006) 'Information Science' in, Princeton, New Jersey: Princeton University Press, 284-300.

[15] Manning, C. D., Raghavan, P. and , S., H. (2008) Introduction to Information Retrieval, Cambridge, England: Cambridge University Press.

[16] Goh, E. (2003) Secure Indexes http://crypto.stanford.edu/~eujin/papers/secureindex/secureindex.pdf

[17] Goldreich, O. and Ostrovsky, R. (1992) Software Protection and Simulation on Oblivious RAMs

[18] Stefanov, E., Papamanthou, C. and Shi, E. (2013) Practical Dynamic Searchable Encryption with Small Leakage, IACR Cryptology ePrint Archive, pp: 832 https://eprint.iacr.org/2013/832.pdf

[19] Stallings, W. (2014) Cryptography and Network Security: Principles And Practices, New Jersey: Pearson Education.

[20] [Chase, M. and Kamara, S. (2010) Structured Encryption and Controlled Disclosure http://eprint.iacr.org/2011/010.pdf

[21] Rennie, J. (2008) The 20 Newsgroups Data Set http://qwone.com/~jason/20Newsgroups/