A Practical Security Framework for Cloud Storage and Computation

Kavya Premkumar[1]*, Aditya Suresh Kumar[1], Saswati Mukherjee[2]
[1]The Department of Computer Science Engineering, Guindy, Chennai, India.
[2]The Department of Information Technology, College of Engineering, Guindy, Chennai, India.

* Corresponding author. Tel.: +919600179164; email: pkavya8@gmail.com
Manuscript submitted September 30, 2014, accepted April 15, 2015.
doi: 10.17706/ijcce.2015.4.5.336-345

Abstract: Cloud computing is a fast growing technology that transforms the way IT services is used in businesses. Collaborative cloud or federated cloud extends this potential even more by offering true unlimited capacity. Despite its potential to provide millions of organizations with storage and computational power at low costs and great speed, many organizations are still skeptical to use these services due to security concerns. This is because "privacy" in cloud has not been well defined and involves trust on the Cloud Service Provider. The objective of this paper is to propose a novel method, which provides a secure way of storing data in the cloud, and at the same time, allows the user to perform some computation on the stored data without compromising the security of the data. This method combines the ideas of searchable encryption with partial homomorphism. The applicability of this idea and its strengths has been tested with appropriate experiments.

Key words: Cloud computing, cloud security, homomorphic encryption, searchable symmetric encryption.

1. Introduction

The growing popularity of cloud computing among the business world is an undeniable reality. Cloud computing has immense potential, promising innovations beyond imagination. Organizations are both excited and nervous in using this new cloud services. They are overwhelmed by the on-demand offerings of storage and computational power. However customers are also concerned about storing their data on the cloud and/or sending data to the cloud for computation. This is because cloud computing if not properly secured, could mean the compromise of our data making it highly vulnerable. A highlighted by the Cloud Security Alliance (CSA)[1], security threats in the form of data breaches and malicious insiders are not solved and are still an obstacle in providing secure cloud services. The Cloud Service Provider (CSP) is a third party that maintains and manages information about another entity. Keeping raw data on the cloud would mean banking an immense amount of trust on the CSP. Trusting the third party requires taking the risk of assuming that the third party will always act like they are supposed to. This also means we lose direct control of our data. Handing out sensitive information to another company is a serious concern. Are data stored in cloud as protected as data stored in an individual computer or network? A cloud could be used by millions of users; keeping raw sensitive data in it could compromise the privacy of the data. Once the users have stored their data on the cloud, they are not aware of what happens to their data and who has accessed it. The location of data is unknown. Data could be physically stored on a computer that is situated

on a geographically different continent. This fear is further multiplied by the existence of multi- or collaborative cloud since, in this case, there may be the unknown factor of the involvement of multiple clouds that would be handling the sensitive data. Therefore, if there is any breach in security and data is stolen from the CSPs, or even if an insider in any one of these CSPs involved sells the data stored on the cloud, it could have serious repercussions. Revealing sensitive information of a company to a rival is every company's nightmare as it creates a serious business risk and could result in huge losses to the company. Due to this scenario, all potential cloud users were looking for a way by which data stored in cloud can be as protected as data stored in an individual computer or in a private network.

A solution to the data security problem in cloud would be to encrypt the data before uploading it to the cloud. However, this method requires the user to download his entire encrypted data from cloud to his system and decrypt it in order to perform any search over the data (e.g., a simple retrieval of a record from a database). Clearly, this method costs the user considerable overhead in terms of time and makes even a simple search query a very tedious task. To overcome this overhead, SSE (searchable symmetric encryption) [2]-[5] was proposed. The idea of SSE is that, users can encrypt their data in such a way that it allows the user to search for a particular word or field on the encrypted data and it would let the user know if that word which is being searched for is present in the data. All this is done without revealing any of the user's data to the cloud provider since the data remain encrypted during the search and retrieval. This method is ideal in situations where the cloud is used only for storage of data but does not consider the need to perform any computation on this stored data. Encrypting the data makes the data unreadable and hence useless to any hacker who gets their hands on this data. However there is a downside to this approach. Encryption of data also makes computation not possible, thereby prohibiting the user from using the computational power of the cloud. When the user wants computation to be done by the cloud, the user is stymied from data security, as raw data (decrypted) is needed by the cloud to perform computation. This exposes the user's data to various threats and hence compromises the security. Homomorphic Encryption (HE) [6] schemes have been proposed to solve this problem. The idea of HE is that, it encrypts the data in a way that computations can be performed on the encrypted data to yield the same correct results on decryption.

Homomorphic encryption schemes can be broadly classified as fully homomorphic encryption (FHE) or partial Homomorphic encryption (PHE). Though FHE schemes are ideally, their practical viability is still a fantasy. Hence we resort to use a PHE scheme for our proposal. For the choice of PHE we have RSA, Elgamal, Paillier system. Of these we have chosen to use Elgamal [7] in our proposal. This is because Elgamal is said to be semantically secure. Semantic security also called IND-CPA stands for IN-Distinguishability under Chosen Plaintext Attacks. In-distinguishability means given an encryption of a message from a two message space, an adversary would not be able to identify the message choice with a probability significantly better than that of random guessing (ó). This encompasses the notion that the adversary can learn nothing from the cipher text. Since RSA is a deterministic encryption scheme, the same plain text gets converted to the same cipher texts upon every encryption. This could give the adversary some knowledge about the plain text from cipher texts. Hence RSA is not semantically secure. When compared with Paillier, Elgamal allows multiplication to be computed on encrypted text and its variant also supports addition [8]. Paillier is said to be only additively homomorphic. Hence we concluded to use Elgamal to test our proposal.

However, all existing encryption based algorithms offer solutions to the risk of data security either for storage cloud or for computational cloud since SSE does not allow operations to be performed on cipher texts and partial homomorphic systems are not searchable. This makes the whole concept of cloud security incomplete since the use of cloud services are optimized by the end users when both storage and computation services can be consumed on the same data. Since some CSP's offer better storage solutions

while others offer higher computational power from either the cost or efficiency perspective, customers often prefer to use more than one cloud to meet their needs as they offer less expensive and more efficient solutions. Further, it is observed that encrypting every field of the record, or the entire document is prodigal and onerous since it is enough to obscure the sensitive part of the data. In this way, even if a malicious user gets hold of the data, the entire document still is not lucid. In this paper, we propose an encryption-based solution, which allows user to store encrypted data, search on it and perform computation albeit with certain restrictions, all in a secured environment. The storage of data and computation could be done with the same CSP or multiple ones. Our approach allows the user decide which information he/she wants to reveal to the cloud as well as choose who can access the data. This defines "privacy" in cloud. The current research proposes to combine the power of searchable symmetric encryption with the advantage of Partially Homomorphic solution thereby rendering the proposed solution a complete solution for security in cloud. Our paper is organized as follows: Section 2 reviews related work in cloud computing security. In Section 3 we detail our approach and architecture. Section 4 explains the experimental results. In Section 5 & 6 we derive the conclusion and discuss the future research possibilities of our work.

2. Related Work

In this section, various proposed mechanisms for both homomorphic encryption as well as different searchable encryption schemes are surveyed.

Partially Homomorphic cryptosystems, i.e. systems, which have a limitation on the type of operation or the number of operations that can be performed on the cipher texts, have been developed in various forms. RSA [9], Elgamal [7] and Paillier [10] are some of the partially Homomorphic cryptosystems. Unpadded RSA and Elgamal are multiplicatively homomorphic, while Paillier is additively homomorphic. A variant of Elgamal [8] is proposed where it could be made additive but has a restriction on the size of the data that is to be encrypted. Fully Homomorphic systems, i.e. systems that do not have any limitation on the type of operation nor the number of operations that can be performed on the cipher texts, were considered as not implementable in practice for quite some time. Boneh–Goh–Nissim [11] proposed a system, which evaluates unlimited number of addition operations but only at most one multiplication operation. It wasn't until 2009, when Craig Gentry [12], using lattice based cryptography showed the first fully homomorphic system. However, this scheme did not allow for it's implementation to be practical as the complexity and length of cipher texts increases with increase in security level. A second fully Homomorphic encryption scheme was presented by Marten van Dijk *et al*. [13]. This scheme uses many of the tools proposed in Gentry's construction, but does not require ideal lattices. This technique has almost the same efficiency as Gentry's original proposition. The HElib, a library released in GitHub [14], implements the Brakerski-Gentry-Vaikuntanathan (BGV) [15] homomorphic encryption scheme, along with many optimizations to make homomorphic evaluations run faster.

Yanjiang Yang *et al*. [16], Dan Boneh *et al*. [2] and others proposed research work in the Searchable Encryption field. Most of these researches are based on creating an index of keywords for the searchable encrypted file and mapping the indexes to the words when searched. When data users input a keyword, a trapdoor is generated for this keyword and then submitted to the cloud server. Upon receiving the trapdoor, the cloud server executes comparison between the trapdoor and index, and finally returns all files that contain this keyword to the data users. Song *et al*. [17] proposed a scheme using sequential scan and controlled searching. This approach reduces the complexity of searching and can be used for practical implementations. Wang *et al*. [18] proposed a secure ranked keyword search scheme. Their solution combines inverted index with order-preserving symmetric encryption (OPSE). In terms of ranked search, the order of retrieved files is determined by numerical relevance scores. The relevance score is encrypted

by OPSE to ensure security. It enhances system usability and saves communication overhead. This solution only supports single keyword ranked search. In our work, we have implemented the system proposed by Song *et al*.

3. Proposed Work

Let us consider a situation where Alice has a set of documents, say records of a database, and she wants to store them on cloud Bob whose storage capacity is high. Alice is aware of the vulnerabilities of cloud security and hence wishes to encrypt her data before storing it. However encrypting her entire database would mean decrypting it completely before she could access a particular record for example. This encryption and decryption process consumes a large amount of time and contributes to a large overhead. Hence Alice decides to use searchable symmetric encryption (SSE). This scheme allows her to search for specific keywords while the file remains encrypted. Now she can search for a particular word/data and retrieve those records where they occur. This saves her from the risk of storing raw data in the cloud and also minimizes the overhead in decryption. This solution is suitable for storage and retrieval purposes.

Let Charlie be another cloud service provider (CSP) whose computational power is high. Alice wants to do some computation on data stored at Bob. Hence Alice specifies what data she wants and what operation to perform on them to Charlie. Charlie searches data on Bob and retrieves them. However for computation to be performed, the data must be raw. This means Alice has to provide Charlie the key to decrypt the data thereby revealing her data to Charlie. This poses a serious privacy problem and defeats the purpose of encryption in the first place. For this reason, Alice uses a partially homomorphic algorithm to pre-encrypt her data. Since it takes a lot of time and power to encrypt the whole data set, we propose that Alice only encrypts only the sensitive parts of the data rather than the whole database.

Thus our approach involves two layers of encryption:

1) Sensitive data fields using Elgamal.
2) Searchable encryption layer.

By using this method, Alice is saved from two major security threats: Data Breaches and Malicious insiders. If there is a security breach at the CSP Bob and Alice's data is stolen or an insider at Bob sells this data for a profit, there would be no concerns for Alice as her data is encrypted by the SSE and without the key (which is only known to Alice) this stolen data is of no value to the attacker or any potential buyer. During computation, Alice would have sent a very few of her data (few hundreds of records) to the CSP Charlie and if there happens to be a security breach at this point in time then normally there would be major concerns but this too is thwarted as parts of Alice's data which she knew were of importance was pre-encrypted using Elgamal thus ultimately making the data useless to the attacker.

Thus, in order to perform operations on the encrypted data, Charlie has to remove the layer of Searchable Encryption. The remaining Elgamal layer obscures sensitive data but allows certain computations to be done on it.

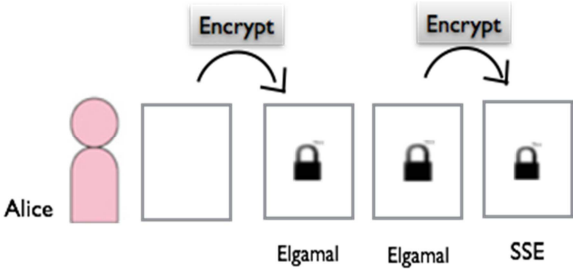

Fig. 1. Encryption.

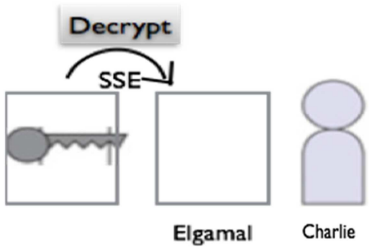

Fig. 2. Decryption.

Fig. 1 and Fig. 2 explain the encryption and decryption process. In order for secure transmission of the searchable encryption key to Charlie for retrieval of data, we suggest to use SSL (secure socket layer) [19] for transport layer security. Using SSL, the user can establish a secure session with Cloud Charlie.

Before establishing this session, both parties enter into a handshake procedure where different cryptographic parameters are negotiated. A pre master key is exchanged that is used by both Alice and Charlie to generate the master session key. Also during this handshaking procedure, Alice could demand for the certificate of Charlie to validate his authenticity. In order to verify the authenticity of different cloud service providers, a PKI-like certificate trust model called Inter-cloud Trust model as suggested by D Bernstein *et al*. can be used [20]. Once Charlie is authenticated and the session is established, Alice can securely transmit her searchable encryption key to Charlie. A similar mechanism of inter-cloud trust will also allow Charlie and Bob to share a trust agreement letting Charlie access the data stored at Bob. Fig. 3 explains the architecture of the proposed solution.

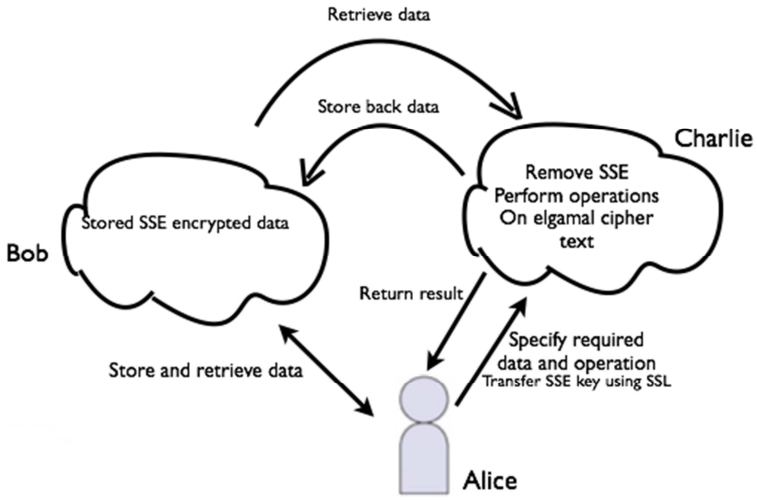

Fig. 3. Architecture.

The following sub-sections explain the theoretical proof of Elgamal and SSE.

3.1 Partial-Homomorphism Using Elgamal

Elgamal [7] is an asymmetric key encryption algorithm used for public key encryption based on Diffie-Hellman key exchange. 0 can be defined over any cyclic group G. Its security depends upon the difficulty of a certain problem in G relating to solving of discrete logarithms. Elgamal is multiplicatively homomorphic. Homomorphic encryption [6] allows specific type of operations to be carried out on cipher text and produces an encrypted result, which when decrypted matches the result of operations done on the plaintext. All computations of the Elgamal Cryptosystem take place over a group G such as a sub-group of Z_p^* or an elliptic curve. We assume that $|G| = q$ is a prime and there is a system wide generator g of G. Elgamal encryption involves the following algorithms:

Algorithm 1: Key Generation

Step 1: Generate a group G of order q using generator g where q is a prime number and g is a primitive root of q.

Step 2: Choose a number x between $\{0 \ldots q\text{-}1\}$.

Step 3: Compute $h = g^x$.

Step 4: Publish G, g, q, h as public key and retain x as secret key.

Algorithm 2: Encryption:
Step 1: Choose y between {0…q-1}.
Step 2: Compute $s=h^y$.
Step 3: Multiply the message M and s.
Step 4: Create the cipher text as { g^y, m. h^y}

Upon receiving a cipher pair {$c1$, $c2$}, decryption is done as follows

Algorithm 3: Decryption
Step 1: Calculate s= $c1^x$.
Step 2: M. s^{-1}.

The decryption can be expressed as

$$2 \,.\, \boldsymbol{s^{-1}} = \boldsymbol{m}.\boldsymbol{h^y}.(\boldsymbol{g^{xy}})^{-1} = \boldsymbol{m}\,.\boldsymbol{g^{xy}}.\boldsymbol{g^{-xy}} = \boldsymbol{m} \tag{1}$$

Multiplicative homomorphism property:

$$E\ (x1) = \{\ g^{y1}\ , x1\ .h^{y1}\ \} \tag{2}$$

$$E\ (x2)\ = \{\ g^{y2}, x2\ .h^{y2}\ \} \tag{3}$$

$$E\ (x1).E\ (x2) = \{\ g^{y1+y2}, (x1.x2)h^{y1+y2}\ \} = \ E\ (x1.x2) \tag{4}$$

3.2 Searchable Symmetric Encryption

The main purpose of using searchable encryption is to secure data during storage and make retrieval easier and less expensive. The scheme that we have used allows the user to search for Keywords in a document. This could, for example, be used to search for a name and retrieve a record accordingly. This approach for searchable encryption uses deterministic encryption in which case the same word gets encrypted to the same cipher text each time. Unlike usual approaches of using indexes for searching keywords, implementation of searchable encryption by Song *et al*. [16] is based on sequential scan of the entire document for the keyword.

Algorithm 4: Searchable Encryption
Step 1: Generate a sequence of pseudo random value $S_{1...}S_l$ using some stream cipher (pseudorandom generator G).
Step 2: To encrypt a word W_i at position i, take S_i and set T_i = {S_i , $F_{ki}(S_i)$ } where F is a pseudorandom function.
Step 3: Output Cipher text as $C_i = W_i \oplus T_i$.

A user who wants to store data in the cloud would first encrypt the sensitive fields using Elgamal (Algorithm 2) and then encrypt the entire document using searchable encryption (Algorithm 4). In this case, the entire encrypted document is stored in one cloud. Retrieval of data is possible through keyword searches followed by decryption at the user side. When there is a necessity to perform computation on the same/another cloud, the searchable encryption layer is removed but the sensitive data is still protected

using Elgamal. This data can be multiplied or added and the new result could replace the old entry and then encrypted again with SSE and stored back.

In cases where authorized users want to search for the sensitive data, although Elgamal is probabilistic and would produce different cipher texts each time, in order to generate the same cipher text it is enough to have saved the value of y. The values of g, q, and h remain the same since we r using the same key. To get the same cipher text for a given plaintext:

Step 1: Compute g^y.

Step 2: Compute $s = h^y$.

Step 3: Create cipher text as $\{ g^y , m. h^y \}$.

Thus it can be seen that, using the same value of y and the same keys on a given text would produce the same cipher text every time. This can be used to figure out the Elgamal-generated cipher text for a given message and search it on the cloud.

This proposed framework is useful in situations like calculation of interests over a period of years for a bank deposit, a percentage salary rise for workers in a firm, insurance firm, any sales force in order to calculate the total sales/cost. In case of bank deposits, interest can be added/subtracted to the deposits of the customers without needing to decrypt the data. (E.g. a "20" percent interest on a initial amount of "100" can simply be added by multiplying the Elgamal encrypted form of "100" with Elgamal encrypted form of "1.20" and similarly subtracted by using "0.80"). This method will help save time, as it would not require the company or user to retrieve record and perform these operations. It is enough to perform decryption at the end of the deposit maturity period.

4. Experimental Setup

The experiments are carried out in a 5-node cluster. Each node has 2.7 GHz Intel (R) Core TM Processors, i-7 2600, CPU @ 3.40GHZ, 8 GB of RAM and 512 GB hard disks.

Table 1 represents a simple bank database where we have identified "Balance" as the sensitive field. Upon first layer of encryption we will have two fields for Balance since Elgamal encryption produces a pair of cipher texts. Followed by Elgamal, we feed the database to the SSE code. Upon searching for a given name or attribute, the record is retrieved if found. Upon retrieval the SSE layer is decrypted.

Table 1. Sample Table

Name	**AccNo**	**Balance**
Varchar	Varchar	Varchar

Suppose a 20% interest is to be added to a customer. His/hers record can be retrieved by searching for his name and the increment can be done homomorphically on the cipher texts of balance by multiplying them with encrypted form of 1.20 .The result can be stored back to the database. Following this update, SSE is run again on the updated fields. Time taken for generating keys, encrypting, decrypting and performaing multiplication using Elgamal with different key sizes is shown in Table 2.

Time taken for searching for a word depends on the location of the word since the proposed solution uses sequential scan. The searching of a word is tested on a local copy of the database due to limitations in the programming language chosen. However, this can be improved to work directly on the database itself. Table 3 indicates the time taken for encrypting, decrypting and searching with SSE.

Since a conventional approach of encrypting the entire database with Elgamal or any equivalent and also the need for decrypting it upon retrieval, followed by searching for a particular record and computing on it would obviously consume comparatively much more time, our approach is extremely efficient and requires

very less effort from the user.

Table 2. Time Taken (In Seconds) for Various Key Sizes Using Elgamal

Experiment	**256 bit**	**512 bit**	**1024 bit**
Generate Elgamal key	8	119	527
Encrypt one sensitive column of 200 records using Elgamal	0.332	~1	~5
Decrypt one sensitive part of record using Elgamal	0.0015	0.004	~0.03
Perform homomorphic multiplication on the retrieved record (e.g. multiply by 1.2 for 20% increase)	~0.0002	~0.0003	~0.0004

Table 3. Time Taken (In Milliseconds) for Searchable Encryption

Experiment	**Time (in milliseconds)**
Encrypting With SSE	25
Decrypting with SSE	24
Searching	~5

Fig. 4 indicates the total time taken for the whole process for different key sizes.

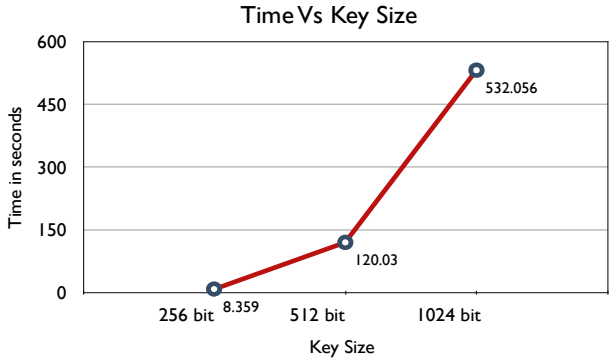

Fig. 4. Time vs. key size.

An example of multiplication has been taken here. It is also possible to retrieve two records and multiply their cipher texts. Experiment was carried out on considerable huge real world numbers and the time taken was found to be in order of milliseconds.

5. Conclusion

As users of both private and public sectors become more and more aware of cloud and it's plethora of services, they are searching for ways of making it more flexible and cost efficient. This immense growth of cloud computing increases the concern and importance of cloud security. The strength of the cloud lies in that it offers both storage and computational power. Few CSP's provide better storage solutions while others provide better computational offerings. Ideally, a company would want to use the CSP, which provides the best package in their field of service. Hence a customer would opt to use multiple CSPs to make the best use of resources. In such situations where data could be in three states, viz, in storage, in transit and in computation, obfuscating the sensitive data is crucial. In this paper, we proposed an approach that

successfully combines two useful solutions (SSE and PHE) for cloud security in order to provide an elaborated security framework. Cloud services are used to the maximum when we store as well as compute data on the cloud. Limitations of the existing solutions for either offering a solution for storage or computation are taken care of by combining two successful solutions. With the proposed solution, the customer is given the power to decide which part of the data he doesn't mind revealing while still the sensitive part of the same data keeping obscure. By combining the strength of encrypted search and the malleability of few cryptographic algorithms, it was possible to provide an appreciable security for data stored on the cloud as well as compute on them. We have taken the homomorphic property of Elgamal algorithm as an example to run our suggestion. This solution uses simple existing algorithms that are neither time consuming nor highly convoluted. The testing of each module was successfully carried out and the results were as desired. Due to the limitations of the chosen programming language, the searching on encrypted data was done on a document rather than a database during the experiments. This can be easily extended to allow MySQL queries on encrypted data.

References

[1] CSA. (July 2014). The notorious nine: Cloud computing top threat in 2013. From http://www.cloudsecurityalliance.org/download/the-notorious-nine-cloud-computing-top-threats-in-2013/

[2] Boneh, D., Crescenzo, G. di, Ostrovsky, R., & Persiano, G. (2004). Public key encryption with keyword search. *Proceedings of Eurocrypt 2004: Vol. 3027. Lecture Notes in Computer Science* (pp. 506-522). Springer.

[3] Chang, Y.-C., & Mitzenmacher, M. (2005). Privacy preserving keyword searches on remote encrypted data. In J. Ioannidis, A. D. Keromytis, & M. Yung (Eds.), *Proceedings of ACNS*: *Vol. 3531*. *Lecture Notes in Computer Science* (pp. 442–455).

[4] Curtmola, R., Garay, J., Kamara, S., & Ostrovsky, R. (2006). *Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions*. Cryptology ePrint Archive. Report 2006/210. From http://eprint.iacr.org/

[5] Kamara, S., Papamanthou, C. & Roeder, T. (2012). Dynamic searchable symmetric encryption. *Proceeding of the 19th Conf. on Computer and Comm. Security* (pp. 965-976). ACM.

[6] Rivest, R. L., Adleman, L., & Dertouzos, M. L. (1978). On data banks and privacy homomorphisms. *Foundations of Secure Computation*.

[7] ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, *31(4)*, 469-472.

[8] Jakobsson, M., & Juels, A. (December 3-7, 2000,). Addition of ElGamal Plaintexts. *Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology* (pp. 346-358).

[9] Rivest, R., Shamir, A., & Adleman, L. (Feb. 1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21(2)*, 120-126.

[10] Pallier, P. (May 1999). Public-key cryptosystems based on composite degree residuosity classes. In J. Stern (Eds.), *Proceedings of Eurocrypt 1999*: *Vol. 1592*. *LNCS* (pp. 223–238). Springer-Verlag.

[11] Boneh, D., Goh, E.-J., & Nissim, K. (2005). Evaluating 2-DNF formulas on cipher-texts. *Proceedings of Theory of Cryptography - TCC'05: Vol. 3378. Lecture Notes in Computer Science* (pp. 325-341). Springer.

[12] Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. *Proceeding of Symposium on the Theory of Computing* (pp. 169-178).

[13] Dijk, M. van, Gentry, C., Halevi, S., & Vaikuntanathan, V. (2009). Fully homomorphic encryption over the

integers. From http://eprint.iacr.org/2009/616

[14] Halevi, S. (April 2013). An implementation of homomorphic encryption. Git Hub Repository. From https://github.com/shaih/HElib

[15] Brakerski, Z., Gentry, C., & Vaikuntanathan, V. (2011). *Fully Homomorphic Encryption without Bootstrapping*. Cryptology ePrint Archive. Report 2011/277. From http://eprint.iacr.org/

[16] Yang, Y. J., Lu, H. B., & Weng, J. (2011). Multi-user private keyword search for cloud computing. *2011 IEEE Third International Conference on Proceedings of Cloud Computing Technology and Science* (pp. 264-271).

[17] Song, D. X., Wagner, D., & Perrig, A. (2000). Practical techniques for searches on encrypted data. *Proceedings of IEEE Symp. Security and Privacy* (pp. 44-55).

[18] Cao, N., Wang, C., Li, M., Ren, K., & Lou, W. J. (March 2011). Privacy-preserving multikeyword ranked search over encrypted cloud data. *Proceedings of 30th IEEE Conference on Computer Communications* (pp. 829-837).

[19] Elgamal, T. (April 1995). The secure sockets layer protocol (SSL). from http://www3.ietf.org/proceedings/95apr/sec/cat.elgamal.slides.html

[20] Bernstein, D., & Vij, D. (2010). Intercloud directory and exchange protocol detail using XMPP and RDF. *Proceedings of 2010 6th World Congress on Services* (pp. 431-438).

Kavya Premkumar is from Chennai, India. She is currently pursuing her final year of undergraduate studies in the field of computer science engineering in Anna University, Guindy, Chennai, India. She has worked as an intern in Defiance Technologies for a span of one month during summer 2013, where she worked with database systems. She has also been on an exchange program to KTH Royal Institute of Technology, Sweden where she studied graduate courses for one semester.

Ms. Kavya did her research in cloud security under Prof. Saswati Mukherjee, Anna University, Chennai during Summer 2014.

Aditya Suresh Kumar is from Chennai, India. He is currently doing his final year of undergraduate studies in the field of computer science engineering in Anna University, Guindy, Chennai, India.

Mr. Aditya did his research in cloud security under Prof. Saswati Mukherjee, Anna University, Chennai during Summer 2014.

Saswati Mukherjee is currently a professor in the Department of Information Science and Technology, CEG, Anna University, Chennai, India. Her fields of research interest are distributed systems, grid and cloud computing and information retrieval and natural language processing. She is currently guiding many fellows towards their Ph.D. and M.S. (by research) in Anna University.