



Homomorphic Encryption for Secure Data Storage on the Cloud

¹Shubhangi Arora, ²Monika, ³Vinod Kumar¹M.Tech Student, ^{2,3} Assistant Professor^{1,2,3} Department of Computer Science & Applications, Kurukshetra University, Kurukshetra, Haryana, India

Abstract-Industries are showing great interest in storing data on public clouds. This could be a result of the remarkable growth of data recorded in the last few years. However the security issues associated with data storage over cloud is a major demoralizing factor for probable adopters. Hence the polestar of today is to find cryptographic methods that will offer more than confidentiality. Homomorphic encryption is one of the methods that have interesting applications in cloud. The objective is to manage and protect the data from the users of a client organization which wants to store the data on untrusted public clouds. This paper is proposed that addresses the confidentiality and assurance concerns and produces encrypted storage with public clouds. This proposed approach adopts Homomorphic encryption for saving the user data and uses an updation method for modified file to reduce the bandwidth consumption while the transmission of large encrypted files.

Keywords: Cloud Computing, Cryptography, Cloud Storage Security, Fully Homomorphic Encryption.

I. INTRODUCTION

Cloud computing offers a cost-effective solution to manage the IT infrastructure in a flexible and scalable manner. Cloud computing enables software applications, deployment platforms, even the computing resources to be made available on-demand using a pay-as-you-go model. This has drawn a lot of attention towards the domain in recent years. Today a good number of organizations use the cloud for their day to day operations and the adoption rate by others are also high [1]. There have been concerns with respect to confidentiality and privacy of data being stored on the clouds.

Security is the primary attribute of the quality of service and the cloud service provider has to give full assurance of security in terms of confidentiality, privacy and integrity. Among these factors privacy is a primary and uncompromisable factor of security. Encryption is the way to secure the data in the untrusted cloud server. Most of Encryption methods currently available had no effect on real time cloud applications. Homomorphic encryption using EcElGamal algorithm based on additive property is used to provide the security of data. This framework should handle file uploads in an efficient manner to reduce bandwidth consumption.

II. RELATED WORKS

2.1 Homomorphic Cryptosystem:-

Homomorphic encryption is a scheme of encryption that allows some special categories of computations to be transferred on cipher text and obtain an encrypted outcome and then decrypted outcome gives the functions performed on the plaintext. For example, one could add the two numbers which is encrypted and another could decrypt that result, without know the value of the individual numbers. Homomorphic encryption is one of the methods that would allow operation on data without knowing the actual content can help in lot of areas. Users have to trust an entity, human or machine to maintain secrecy of their data. But an attack on the trusted party or vulnerability with the system can expose the user's secret. Hence the necessity of systems where even the service providers have no detailed knowledge of the users data is growing.

Homomorphic encryption originated from the concept of privacy homomorphism [2], introduced by the Rivest et al. In their paper, they discussed about performing operations on the encrypted data. The RSA [3] cryptosystem introduced by them also exhibited the property of partially homomorphic encryption, allowing multiplication of encrypted data, which when decrypted will give the product of the plaintexts. Ever since many schemes with homomorphism properties have been proposed. Another cryptosystem developed during the same period was the ElGamal Cryptosystem [4]. Developed and named after Taher ElGamal, ElGamal cryptosystem also had some homomorphism properties (multiplicative). Homomorphic encryption has great potential for use in scenarios ranging from multi-party communication to secure computation in cloud systems.

EcElGamal cryptosystem with additive property

Assume that here are some elliptic curve C [5] that is described over a finite field Fq where $q = pn$ is large (and p is prime). Suppose that C , g , and a point $G \in C$ are public and inserted in the system $m \leftrightarrow Pm$. When Alice wants to communicate secretly with Bob, they proceed thus:

- Bob chooses a random integer number y , and declares the point yG (while y remains secret).
- Alice chooses her own random integer number x and sends this two points $(xG, Pm+x(yG))$ to Bob (while x remains secret).

- To decrypt the message, Bob calculates $y(xG)$ from the first part of the pair, then subtracts it from the second part to obtain $Pm+x(yG)-y(xG)=Pm+xyG-xyG=Pm$, and then inserted again to obtain the original message m .
- Eve, who can only see yG , xG , and $Pm+x(yG)$ must find x from xG or y from yG to make sense of $Pm+x(yG)$, so her problem is reduced to the ECDLP(Elliptic curve discrete logarithm problem), and she is opposed.

A technical problem is how to encode characters into points of an elliptic curve (note that $M \in C$). There is a variation of the cryptosystem sometimes called MV-ElGamal (MV stands for Menezes and Vanstone) that avoids this technical problem. In this version a message M is divided into two blocks $m1$ and $m2$ modulo p , i.e. $Fp \times Fp$ is the set of plaintext messages (and the encoding is very easy).[6]

This is a successful cryptosystem because every operation that Alice and Bob have to carry out (addition and subtraction on the curve) is relatively easy, while the operation that Eve would have to perform to crack the system is extremely difficult (or for real-life villains without the proper resources, perhaps impossible).

III. PROPOSED FRAMEWORK

This framework usually supports in storing, distributing, and serving files in a secure manner. The comprehensive progress of data proceeds as follows; First, data generated by different users which could consist of data from the employees, faraway colleagues, employees from third-party organizations on contract or individuals using services afforded by the client organization. This data is stored and uploaded in files to the cloud infrastructure of the organization over an interface. When the files remain within the trusted infrastructure of the organization it remains protected, but when it is onward to other public cloud services the issues on trust originates. The proposed framework offers a beneficial method to obtain communication with other public clouds. The uploaded files are stored in provisional confidential storage and are encrypted using EcElGamal Homomorphic Cryptosystem (with additive property) to ensure confidentiality. Based on the configuration, the encrypted files are sent to the public clouds. From this point, the security of the data is an issue but the contents are encrypted, confidentiality of the data is established. Now at the next phase when access to a file is appealed and if the file is accessible within the private storage then it gives a response otherwise the file is downloaded from the third-party cloud service on which it is stored and then decrypted within the trusted private storage. The decrypted file is delivered as response. If there is any modification require in encrypted file then there is no need to download entire file by creating the patch file which contains the new content in encrypted form add to the encrypted file that is already stored in the cloud. Patch file concept improves the bandwidth utilization and reduce time to download and upload the complete file.

3.1 Components

The proposed framework has five major components. A brief description of each component is described below-

Encryption/Decryption Module

This module is important for the key generation, encryption, and decryption of files. It uses the EcElgamal Cryptosystem for various cryptographic procedures. When the file is encrypted, it is divided into two blocks. The content of each block is converted to a numerical form using their hexadecimal representation. Then the encryption and decryption is implemented on the block and the result is appended to the encrypted file. The procedure is defined below in the form of an algorithm.

Algorithm1 Encrypt_File (Ffile, Enfile, public Key)

- 1: Read FFILE till the end, in fixed size blocks
- 2: for all blocks do
- 3: Block Content \leftarrow Content of the block from FFILE
- 4: en \leftarrow encrypt (Block Content, public Key)
- 5: Write en to ENFILE
- 6: end for

Algorithm2 Decryptfile (Enfile, Defile, private Key)

- 1: Read ENFILE till the end, in fixed size blocks
- 2: for all blocks do
- 3: Block Content \leftarrow Content of the block from ENFILE
- 4: de \leftarrow decrypt (Block Content, private Key)
- 5: Write de to DEFILE
- 6: end for

Private Store

This framework is created with a prospect to forward files to other cloud services. But in order to process the files before sending, a provisional storage area is mandatory and private Store provides that storage. Different operations supported by the private store are storing user files, handling file download requests and sending files to public clouds. The details of these operations are described in STORE_FILE, DISPATCH_FILE, and ACCESS_FILE procedures.

FILES database

FILES database involves essential information on each and every file stored. This framework also includes the files created later such as encrypted files and patch files. It also includes the list of cloud services where the files need to be saved. File ID is a unique value referred to identify the file. The file name and file size attribute consist of the name of the file as uploaded by the user and its size respectively.

Algorithm 3 Store_File (Ffile)

- 1: Write FFILE to the Private Store
- 2: Update FILES database
- 3: ENCRYPT_FILE (FFILE)
- 4: Search for existing versions of FFILE
- 5: if found then
- 6: Select the latest existing file version, fold and remove others, if any
- 7: if size (FFILE) > size (Fold) then
- 8: Encrypted Differencing (Fold, FFILE, EFdiff, public Key)
- 9: end if
- 10: end if
- 11: DISPATCH_FILE (FFILE)

Algorithm 4 Dispatch_File (Ffile)

- 1: Search for encrypted version of the file, ENFILE and encrypted patch file, EFdiff
- 2: if EFdiff is found then
- 3: $F \leftarrow \text{EFdiff}$
- 4: else
- 5: $F \leftarrow \text{ENFILE}$
- 6: Get the list of public cloud services for file storage, Location List from the FILES database.
- 7: for cloud Service \in Location List do
- 8: Send (F, cloud Service)
- 9: end for
- 10: [Optional] Remove EFdiff (if any) and ENFILE from Private Store.
- 11: end if

Attribute contains a list of cloud data storage services on which the file is stored. The Public key file attribute is used for encrypting the file. In case same public key file is used for all plaintext files, this attribute will have same value. This attribute is kept for future use. It can be used to encrypt a file with various keys depending on the cloud storage location.

Presentation Module

This is the essential interface for user interaction with the cloud. It is a web interface which serves users a list of their files stored with our framework and grants them to upload or download files. For an administrator, it produced options to change the configuration such as generating new keys and adding support for new cloud services.

3.2 Updation Method

Now one problem is noticed with this framework. After encryption, the file size becomes very large and consumes high bandwidth during the transmission. Further many files get frequent updates in this case transmission of the encrypted version of the updated file are necessary. To reduce the bandwidth consumption the following approaches can be considered-

File Updation Approach:

A modified approach for updating encrypted files that is useful in cloud schemes is described. In this approach, two versions of a plaintext file, say f_1 and f_2 which are states of the file at time t_1 and t_2 , the objective is to find an encrypted patch file EFdiff that can be used to create an encrypted version of f_2 say ENf_2 from the encrypted version of f_1 say ENf_1 . Now EFdiff can be sent to the public clouds where ENf_1 is stored and can be used to update the encrypted file without the need for transmission of the encrypted version of the updated file. The method for searching encrypted difference and performing update using that difference is described in the Encrypted_Differencing and Encrypted_Patching procedures. Here the encrypt Add procedure uses the additive homomorphism property of the EcElgamal Cryptosystem to add the encrypted values. The size of the resulting patch file can be further reduced using compression techniques.

Security Analysis:

Consider M_1 and M_2 are two instances of a message.

Hence $M_1 = me_{11}, me_{12}, me_{13} \dots me_{1n}$

And $M_2 = me_{21}, me_{22}, me_{23} \dots me_{2n}$, where m_i represents a single block of a message.

Let C_1 and C_2 be the cipher texts generated by applying the encryption procedure on M_1 and M_2 respectively.

Algorithm 5 Encrypted_Differencing (f1, f2, EFdiff, public Key)

- 1: Read f_1 and f_2 till the end, in fixed size blocks
- 2: for all blocks do
- 3: $BC_1 \leftarrow$ Content of the block from f_1
- 4: $BC_2 \leftarrow$ Content of the block from f_2
- 5: if $BC_1 = BC_2$ then
- 6: if $BC_2 > BC_1$ then
- 7: $di \leftarrow BC_2 - BC_1$
- 8: $findex \leftarrow$ Block Index
- 9: else
- 10: $di \leftarrow BC_2$
- 11: $findex \leftarrow -$ (Block Index)

```

12: end if
13: en ← encrypt (di, public Key)
14: write (findex, en) to EFdiff
15: end if
16: end for
    
```

Algorithm 6 Encrypted_Patching (Ef1, EFdiff, Ef2, public Key)

```

1: Read EFdiff till end, in pairs (findex, difference)
2: Read Ef1 and write unaffected blocks to Ef2
3: for all pairs do
4: if findex < 0 then
5: ui ← difference
6: else
7: Assign en the encrypted value at findex of Ef1
8: ui ← encrypt Add (en, difference, public Key)
9: end if
10: Write ui to EF2
11: end for
    
```

Now $C1 = c_{i1}, c_{i2}, c_{i3} \dots c_{in}$

and $C2 = c_{i21}, c_{i22}, c_{i23} \dots c_{i2n}$, where c_i represents a single block of a cipher text C. As $M2$ is an updated version of $M1$, hence it can also be expressed as follows

$$M2 = (m_{e_{i1}} + d1). (m_{e_{i2}} + d2). (m_{e_{i3}} + d3) \dots (m_{e_{in}} + dn),$$

Where $d_i = m_{e_{2i}} - m_{e_{i1}}$, is the difference. Now a single block of cipher text can be expressed as,

$$c_{i1} = \text{encrypt}(m_{e_{i1}}, \text{public Key}).$$

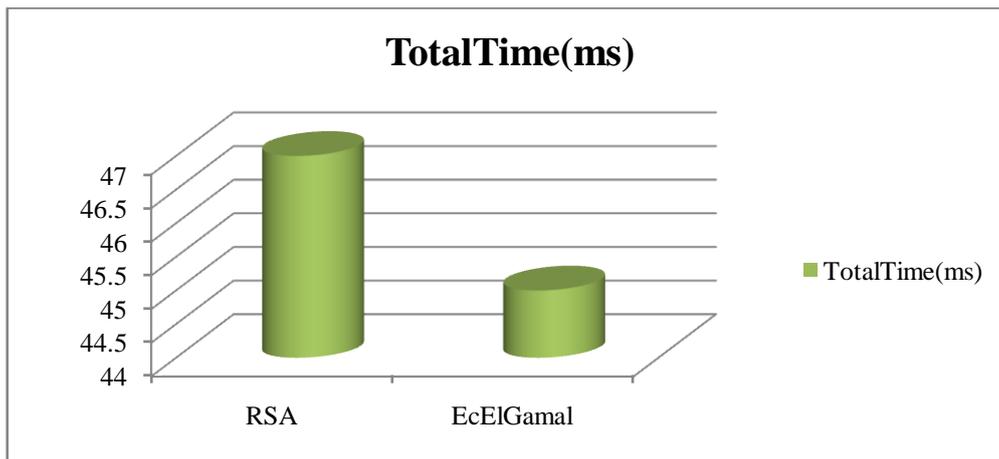
Further the encrypted difference for a block i can be expressed as,

$$en_di = \text{encrypt}(d_i, \text{public Key}).$$

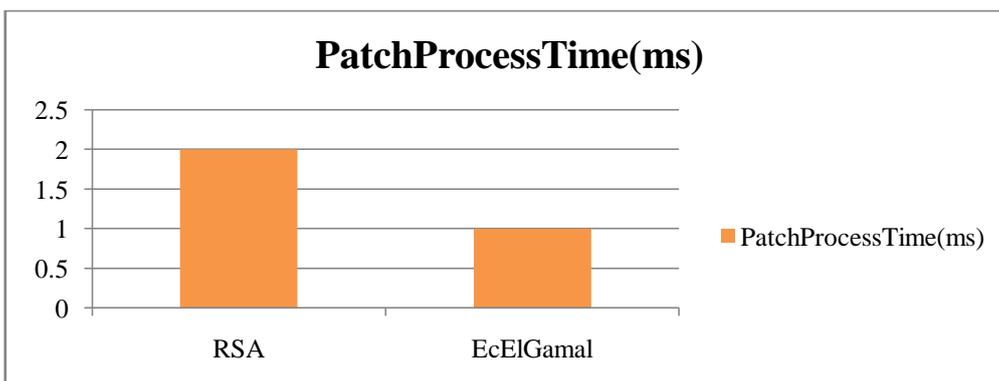
As the EcElGamal Cryptosystem is used for encryption which presents the additive homomorphism property, Hence $\text{decrypt}((c_{i1} * en_di) \text{ mod } n2, \text{private Key}) = m_{i1} + d_i = m_{2i}$.

Similarly when this is enforced over all the blocks, one can update the message $M1$ to $M2$, by using the cipher text $C1$ and the encrypted difference. It is important to note that in this proposed framework the cloud providers do not have the private Key and hence cannot decrypt the cipher texts. Yet they can update the content in their encrypted form itself. All interactions with the untrusted cloud involve encrypted contents only.

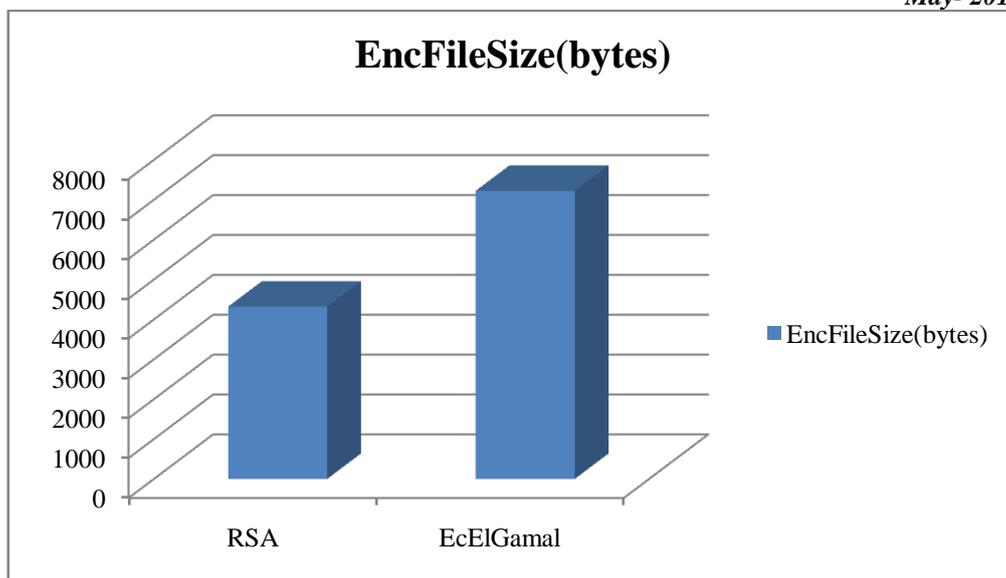
IV. EXPERIMENTS AND RESULTS



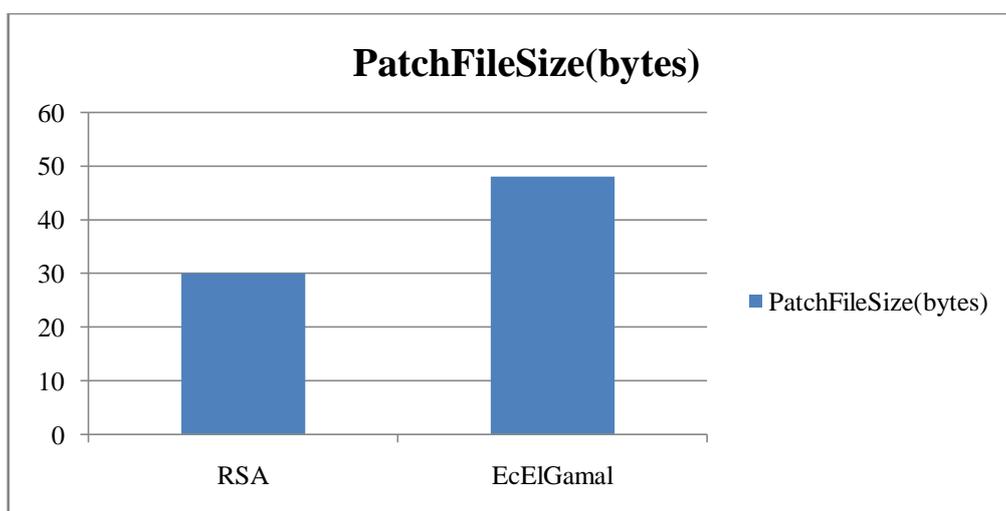
Graph 1: Total Time of enc and dec (ms)



Graph 2: Patch Process Time (ms)



Graph 3: EncFileSize (bytes)



Graph 4: PatchFileSize (bytes)

This approach has been proposed that the process time of patch file and total encryption and decryption time is less than from the previous approach because previous approach was based on multiplicative homomorphic property but the proposed approach is based on additive homomorphic property so that it is proved from the graphs even if the size of encryption file size and patch file size are large then it will take less encryption and decryption time and patch process time are less than the previous approach. Thus, this proposed approach is useful for achieving reduced process time of patch file on the large encrypted files.

V. SUGGESTIONS FOR IMPROVEMENT

The comprised support for the preferred of cloud providers is finite and primitive, but the framework can be explored to support new providers. In the present implementation, the organizations have to manage an application at the public clouds that will implement the updating method. The framework can be established in a way such that the providers can easily consolidate it with their podium. Furthermore working with encrypted data is calculation expensive and intensive in terms of storage and high performance data refining choices in the cloud can be applied for better performance. The proposed approach using newer homomorphic cryptosystems having additive homomorphism can be explored for performance benefits with the help of cryptographic techniques.

VI. CONCLUSION

In brief, the work gives a model of a framework that can be used by organizations to save and handle their data stored over untrusted public clouds. As part of the work the possibility of using homomorphic encryption scheme with additive homomorphism to update encrypted files, rather the transfer of full encrypted versions each time after an update, was examined. The developed model has delivered encouraging performance results as compared to other common solutions in the test environment. Hence the proposed approach could be considered for use in certain world scheme.

REFERENCES

- [1] Ponemon research study infographic:Who's minding your cloud? <http://www.ca.com/us/collateral/white-papers/na/ponemon-research-study-infographic-whosminding-your-cloud.aspx>, 2013.
- [2] R. Rivest, L. Adleman, and M. Dertouzos.On data banks and privacy homomorphisms.pages169–177. Academic Press, 1978.
- [3] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.
- [4] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.
- [5] <https://www.math.hmc.edu/~ursula/teaching/math189/finalpapers/elaine.pdf>, accessed on 14.03.2016
- [6] https://www.uam.es/personal_pdi/ciencias/fchamizo/assignaturas/cripto1011/ecc.pdf, accessed on 14.03.2016