Chat Application Using Homomorphic Encryption

Sagarik Behera [*], *Ashwin Kanth , Avik Adithya Suresh, CVR Ashwin, and Jhansi Rani Prathuri*

Dept of Computer Science and Engineering , CMR Institute of Technology , Bengaluru , India

Abstract. Chat applications nowadays have evolved into one of the most significant and widely used applications on smart devices. They are capable of sending and receiving messages, documents, and images with zero cost to any part of the world. All the messages should be protected. All the chat applications today are used to send messages quickly and securely. The actuality of the situation is that the messages sent are not as secure as they claim to be. So, to bridge this gap, in this paper Homomorphic Encryption is used to secure the messages further while also not reducing the speed of the transaction. This paper aims to implement a Chat application using Homomorphic Encryption which adds a layer of security over end-to-end encryption.

1. Introduction

The fast development of smart devices has led to the development of chat applications as well. In recent years, chat applications have made a major change in social media because of their features like exchanging text, images, documents, and even voice messages which make them very appealing to the audience. However, the main Unique Selling Point (USP) for such chat applications should be security, privacy, and speed. The Electronic Frontier Foundation (EFF) conducted a test and found that the majority of popular chat applications failed to meet the needed standards.

The threats or risks that a chat application can face are identity theft, firewall tunneling, data security leaks, and spam. To overcome such challenges developers usually try to avoid exposing private information and encryption techniques to strengthen the channel of exchange. Different encryption methods which can be implemented are the Rivest-Shamir-Adleman algorithm (RSA), Elliptical curve cryptography (ECC), End-to-End, Diffie Hellmann Algorithm (DE), etc.

[*] sagarika.b@cmrit.ac.in

It was found that End-to-End encryption can be used to prevent security breaches or identity thefts which are the main problem with chat applications. This makes sure that no third party can interfere and get access to the information being exchanged. There have been cases where even this has failed sometimes so our main aim would be to strengthen this using Homomorphic encryption. Homomorphic encryption allows us to perform an operation on the Ciphertext without actually having to decode it. In this way privacy aspect of the message are not compromised and are providing an extra layer of security for this message.

The main objective of this paper is to implement Homomorphic Encryption over an End-to-End encrypted Chat application to strengthen its security.

2. Literature Survey

The main problems which tackled with this application are the possible security breaches and identity thefts that are threatening the privacy of the users. To tackle this situation, Homomorphic encryption methods applied which can solve this issue. Literature survey helped to know more about the problems for chat applications, Homomorphic encryption, and different encryption algorithms which are being used in the industry as of now.

The authors of paper [1] had done a detailed analysis of Homomorphic encryption method. In the usual encryption method, when a user stores data in the third-party location it has to be encrypted first. But when some operations need to be performed on the encrypted data, the user has to share the secret key with the third party. This leads to privacy and secrecy issues. To overcome this problem, now researchers are focusing more on homomorphic encryption method. The detailed study of Partial homomorphic encryption (PHE), Somewhat homomorphic encryption (SWHE), Fully homomorphic encryption (FHE) given in paper [1]. Any encryption scheme's performance evaluated through some criteria such as speed, security and algorithm's simplicity. In case of FHE it is lagging in speed when it is implemented in software. Now researchers are focusing on the hardware implantation using Field-Programmable Array (FPGA) and Graphical Processing Unit (GPU).

Design and implementation of Homomorphic Encryption Processing Unit (HEPU) coprocessor given in paper [2]. It is an FPGA based accelerator. NTRU-based LTV Homomorphic encryption implemented in FPGA. The authors of paper [2] focused on implementing and accelerating Chinese Remainder Theorem (CRT) and inverse Chinese Remainder Theorem (iCRT) using FPGA.

A secure chat application with end-to-end encryption designed for android platform proposed in the paper [3]. The authors used ECDH algorithm to generate key pair and AES algorithm to encrypt the text messages. For image and voice messages RC4 algorithm used for encryption. Due to the convenience of communication using chat apps increases, the security also becomes higher meaning a proper cryptography scheme is needed to protect the messages. Keeping this in mind the authors have implemented a chat application on Android which follows the End-to-End encryption method for high security. They also give the experimental result of their chat applications performance such as the accuracy of the received message, average encryption, and decryption time.

In today's scenario WhatsApp instant messenger service become more popular among all the groups of people across the world. Since it uses End-to-End encryption

(E2EE) to protect user's data and privacy, it become a concern for some Governments to track the illegal activities such as terrorist activities, child pornography etc. To tackle such type of activities Government wants "backdoor" of user's data. But the users of WhatsApp don't want the "backdoor" of their messages. The authors of paper [4] presented the advantages of E2EE in the messaging app and stated why Government shouldn't interfere in the privacy of user.

The main goal of paper [5] is to use RSA homomorphic encryption, which is somewhat homomorphic in nature, to create a key pair from multiple keys instead of a single key pair. Data and network security during data transmission are among the parameters investigated, with encryption playing a crucial role. The use of the Internet for communication has skyrocketed, necessitating a slew of adjustments to maintain secure transmission.The biggest disadvantage is that encrypting with many keys takes longer. The amount of noise is more than predicted. Future work will focus on reducing the time it takes to encrypt using multiple keys and reducing noise increase.

RSA algorithm used for Homomorphic encryption in paper [6] and [7]. It is a partial homomorphic encryption method.

In this paper [8], the author shed light on the importance of messaging apps and the number of people using it for various purposes. A lot of sensitive information is shared as well which can lead to a lot of problems if it falls in the wrong hands. This paper proposes an Instant messaging application where fully homomorphic encryption is used and monitors the memory usage and performance and compares it with ECC. The authors clearly convey the improvements in performance and security when FHE is used.

In this proposed method MTProto 2.0 mobile protocol is used for end-to-end encryption of plaintext. The detailed analysis of this protocol given in paper [9] and explanation of this protocol given in section 3.

3. Proposed Method

3.1 System Architecture

A typical Chat application will follow a simple architecture in which the sender sends a message and this acts as the plain text for the End-to-End Encryption Algorithm. The algorithm gives out a ciphertext after encrypting it using a suitable key, which is then sent to the receiver who decodes it using his key. There are few cases where this model has failed; the proposed method is trying to strengthen this by adding one more layer of encryption.

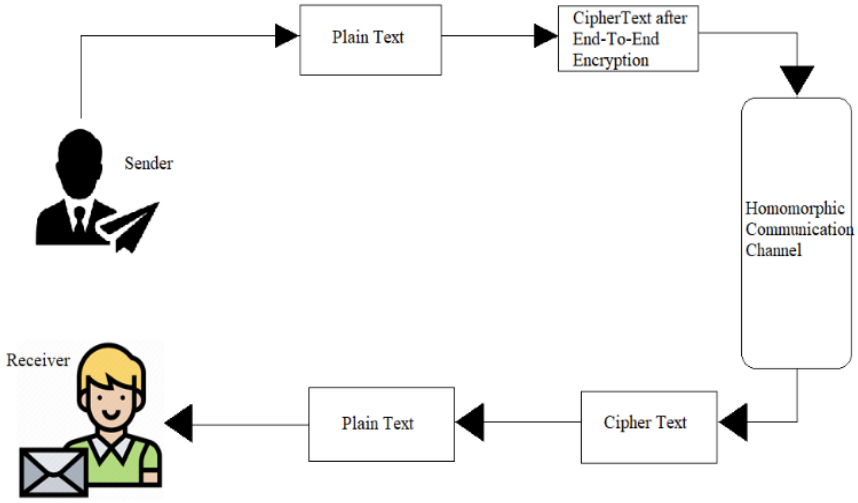

Figure 1. System Architecture

In this chat application as shown in Figure 1, after the cipher text is generated it will be send through a Homomorphic Communication channel which performs certain operations on the ciphertext to make it more complex and harder to decrypt if there is a breach of information. This channel acts as the additional layer of protection for the chat application which will strengthen the security of the messages.

3.2 Homomorphic Encryption

Homomorphic Encryption is a lattice-based encryption system that allows the user to perform calculations on the ciphertext as if he/she would do them on plaintexts with the decrypted results matching the plaintext operations. Homomorphic supports addition and multiplication on the ciphertext, whereas other operations can be derived or approximated from the above two operations. Other operations are hard to implement directly as the scheme hides the message in noise.
Homomorphic encryption can be classified into three different types:
1) Partially Homomorphic: This scheme supports the use of either addition or multiplication but not both.

2) Somewhat Homomorphic Encryption: This scheme supports the use of both addition and multiplication on the ciphertext.

3) Fully Homomorphic Encryption: This scheme supports arbitrary operations on ciphertext with no limitations on the number of operations.

Figure 2 gives a visual representation of the flow of Homomorphic Encryption. Homomorphic Encryption provides a safe channel for the exchange of messages. It has been proved to support the inferences of a trained model for different neural network models.

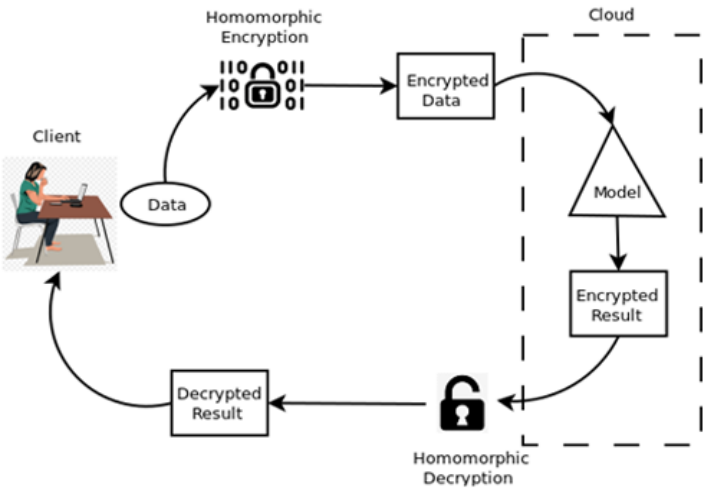

Figure 2. Homomorphic Flow Diagram

3.3 RSA Algorithm

Rivest–Shamir–Adleman Algorithm or the RSA Algorithm is a public key encryption technique that is widely used for various encryption purposes but primarily for data transmission. It is a very old encryption technology that was invented in 1977.

It is an asymmetric key exchange method that uses two keys namely the public key and a private key. The client shares his public key to a server it wants to establish a connection. The server uses this public key to encrypt the requested data from the client. The data sent by the server is then decrypted by the client using his private key. Because of this even if a third party gets hold of the public key or the ciphertext he will not be able to decrypt the message and steal the information.

This algorithm ensures that the keys generated are secure always. The steps involved in the key generation are:

Steps:

1. Two large prime numbers x, and y are selected.
2. Calculate N = x * y
3. Calculate the totient function; phi(N) = (x-1)(y-1)
4. Select an integer i, such that i, is co-prime to $\phi(N)$ and $1 < i < \phi(N)$. The pair of numbers (N,i) makes up the public key.
5. Calculate j, such that $i*j=1 \bmod \phi(N)$

3.4 MTProto 2.0 Mobile Protocol

MTProto 2.0 is a cryptographic protocol which is used in popular Telegram messaging application. The protocol is intended for applications on mobile devices to connect a server API. The sequence diagram of this protocol for secure messaging is shown in Figure 3. There is a sender S and Receiver R. To perform E2EE messaging between S and R, a session key needs to be established using Diffie Hellman Key Exchange through the server. The steps involved in this operation are as follows:

1. The sender S initiates the operation. It sends a request to the server to get the Diffie Hellman (DH) parameter.
2. The server sends the parameter (g, p) to the sender S. Where p is a prime number and g is the primitive root of p and $g < p$.
3. Sender S selects a secret key s, where $s \in Z_p$ and it calculates the public key $g_s=g^s \bmod p$ and generates session identifier id.
4. After that it sends a request message to receiver R through the server to start the encrypted chat. This request message contains (id, R, g_s). After verifying the sender's identity, the server forwards the message to receiver R. It contains (id, S, R, g_s).
5. Once the receiver accepts the chat request, the server shares the DH parameter (g, p) with the receiver.
6. Now the receiver selects a secret key r, where $r \in Z_p$ and it calculates the public key $g_r=g^r \bmod p$.

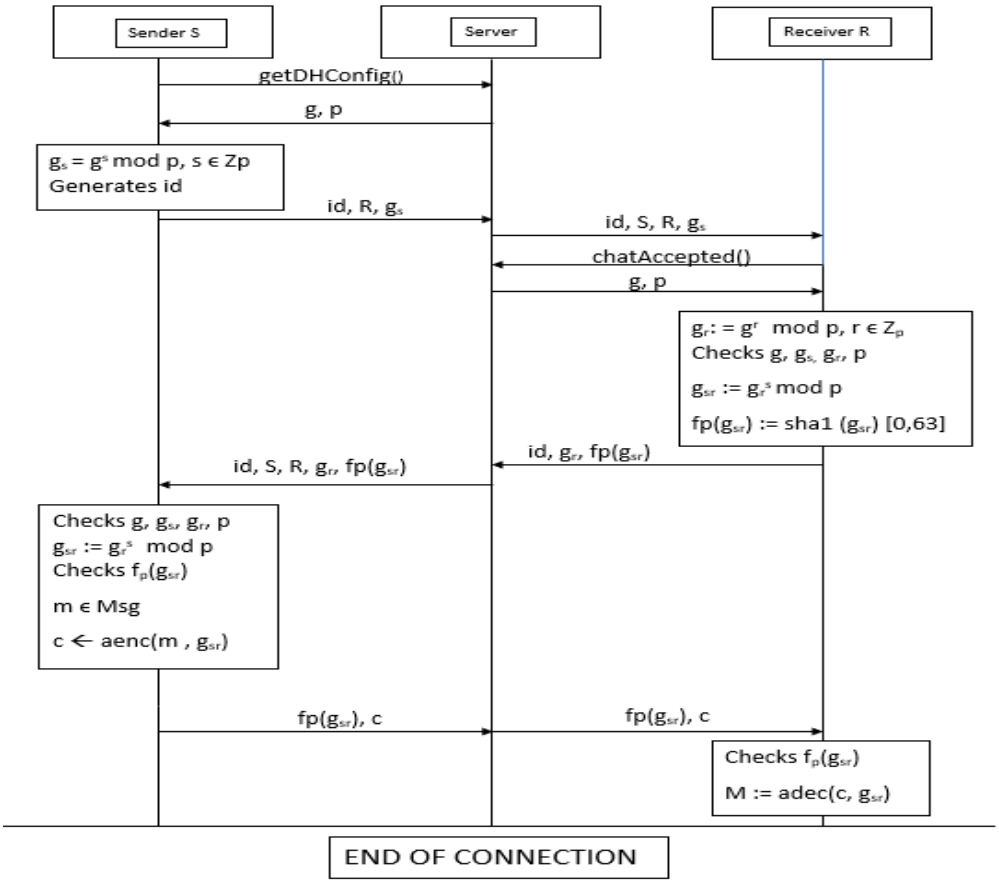

Figure 3. Sequence Diagram of MTProto 2.0 for secure messaging

7. Receiver R calculates the shared secret key $g_{s,r}$ using S's public key. Where $g_{s,r} = (g_s)r \bmod p$. Then it shares the fingerprint of the shared secret key $fp(g_{s,r})$ which is 64 bits of the SHA 1 of the key, id and public key g_r with the sender.
8. After receiving the message sender S verifies it by calculating $g_{s,r}$ using it's secret key. $g_{s,r} = (g_r)s \bmod p$ and checks the fingerprint $fp(g_{s,r})$.
9. After verifying the fingerprint of the secret key, sender S encrypts the message m using $g_{s,r}$ and generates the ciphertext c. It sends the fingerprint of the secretkey with ciphertext to the receiver.
10. Receiver verifies the fingerprint of the secret key and decrypts the cipher text c using $g_{s,r}$ to get back the message m.

4. Results and Discussion

In this paper, an application presented that acts as a secure channel for information exchange. The channel was developed using homomorphic encryption which is able to provide an extra layer of security for the End-to-End encrypted chat application. Since homomorphic encryption requires a lot of hardware the application has a time delay which is observed while researching the encryption technique.

Table 1. Time taken for different key size with different text length

RSA Key Size	Text Length	Time Taken in ms
1024	5	6
1024	17	5
1024	23	6
1024	45	5
2048	5	11
2048	17	9
2048	50	9
2048	105	13
2048	142	15
4096	132	20
4096	169	18
4096	230	21
4096	292	20
5120	396	35
5120	476	47
5120	531	31
5120	555	30

The chat application has an average delay of 0ms in sending messages without homomorphic encryption. The average time delay when the message passes through the homomorphic channel is 25 ms. The time delay varies based on the length of the text. The time delay is more due to the key generation which takes quite some time. The key being used in this is of the size 212. Since generating such a big key takes time the time delay is in 25 ms for every message.

As shown in Table1 the time taken does not change drastically with the text length, this is due to the key which is being used. Once a key is generated that key will be

used for that connection so it does not affect the time taken. With key being big it makes the connection even safer.

The main aim of this app was to see how well the homomorphic encryption can be handled with the present-day hardware improvements as it was found to be very slow when it was first implemented. After implementing it was found the delay to be around 25 ms which is pretty good for the standard of a chat application. Further works can be done to optimize this. Figure 4 shows the graphical representation of time taken for different key size with different text length.

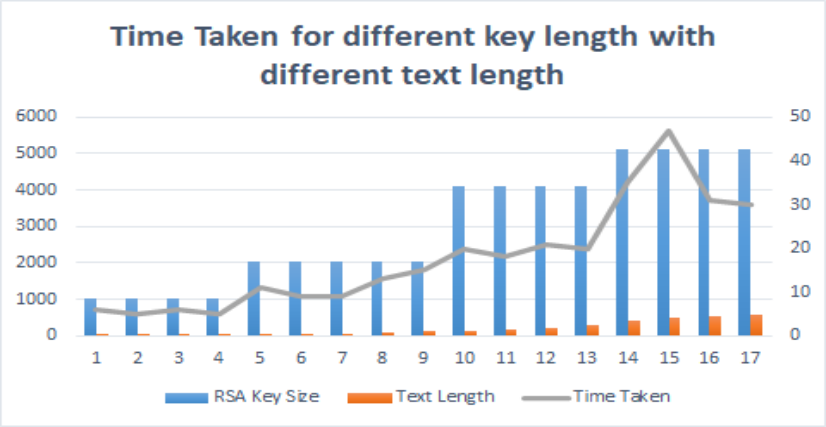

Figure 4. Time taken for different key size with different text length

5. Conclusion and Future Work

The exchange of messages through a Homomorphic channel ensures that the message is reached to the receiver without any third-party getting access to it and there is no chance of identity theft. Since the channel is implemented over the End-to-End encryption layer it makes it even more complex for the person trying to steal the information to decode it. The best feature about this encryption is that the ciphertext need not be decoded before any operation is being performed over it. It works very well with the End-to-End encryption method as the message can be decoded only at the receiver's end once it is converted to a ciphertext. The ciphertext here acts as the input to the Homomorphic channel.

Every feature comes with a drawback of its own drawbacks. Homomorphic encryption comes with drawbacks like more computational speed which leads to a small delay in the message being received or sent by the user. After implementation we found the average delay for the message to be sent as 25 ms. Homomorphic encryption demands high hardware requirements due to the complex operations it has to perform. If the system does not have the minimum specifications required then the application can become slower than usual.

Keeping in mind all these limitations this encryption method can be pursued to make it more optimized by different means. The application as of now has an average delay of 25 ms, if this small delay can be ignored, it will become a very secure channel for communication of information and images. It can be implanted in small organizations

where secure transfer of information is required and where the small delay in the information can be neglected.

References

[1] Acar, Abbas, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. "A survey on homomorphic encryption schemes: Theory and implementation." *ACM Computing Surveys (Csur)* **51**, no. 4 (2018): 1-35.

[2] Cousins, David Bruce, Kurt Rohloff, and Daniel Sumorok. "Designing an FPGA-accelerated homomorphic encryption co-processor." *IEEE Transactions on Emerging Topics in Computing* **5**, no. 2 (2016): 193-206.

[3] Ali, Ammar, and Ali Sagheer. "Design of secure chatting application with end to end encryption for android platform." *Iraqi Journal for Computers and Informatics* **43**, no. 1 (2017): 22-27.

[4] Endeley, Robert E. "End-to-end encryption in messaging services and national security—case of WhatsApp messenger." *Journal of Information Security* **9**, no. 01 (2018): 95.

[5] Makkaoui, Khalid El, Abdellah Ezzati, and Abderrahim Beni-Hssane. "Cloud-RSA: An enhanced homomorphic encryption scheme." *In Europe and MENA Cooperation Advances in Information and Communication Technologies*, pp. 471-480. Springer, Cham, 2017.

[6] Chandravathi D, and P V Lakshmi, "Privacy preserving using extended Euclidean algorithm applied to RSA-homomorphic encryption technique." **VOLUME-8** ISSUE-10, AUGUST 2019, REGULAR ISSUE 8, no. 10 (2019): 3175-3179.

[7] Abid, Rabia, Celestine Iwendi, Abdul Rehman Javed, Muhammad Rizwan, Zunera Jalil, Joseph Henry Anajemba, and Cresantus Biamba. "An optimised homomorphic CRT-RSA algorithm for secure and efficient communication." *Personal and Ubiquitous Computing* (2021): 1-14.

[8] Natanael, Dimas, and Dewi Suryani. "Text Encryption in Android Chat Applications using Elliptical Curve Cryptography (ECC)." *Procedia Computer Science* **135** (2018): 283-291.

[9] Miculan, Marino, and Nicola Vitacolonna. "Automated Symbolic Verification of Telegram's MTProto 2.0." *arXiv preprint arXiv*:2012.03141 (2020).