Applied Information Technology And Computer Science Vol. 2 No. 2 (2021) 165-180

AITCS

Homepage: http://publisher.uthm.edu.my/periodicals/index.php/aitcs
e-ISSN :2773-5141

JAMBU_CHAT: An Online Chat Application for Android Smartphone

Tan Yit Jun, Sapiee Jamel*

Faculty of Computer Science and Information Technology,
Universiti Tun Hussein Onn Malaysia, Parit Raja, Johor, 86400, MALAYSIA

DOI: https://doi.org/10.30880/aitcs.2021.02.02.012
Received 14 June 2021; Accepted 09 September 2021; Available online 30 November 2021

Abstract: Nowadays, people often use chat applications to communicate with each other. Chat applications are prone to various attacks such as eavesdropping, Man-in-The-Middle attack, and impersonation attack. Nowadays, chat applications are using a combination of symmetric key encryption and a hashed based message authenticated code or an authenticated encryption like AES GCM for message confidentiality and authenticity. In this paper, we proposed AES JAMBU as the security mechanism for implementing a secure chat application. AES JAMBU is an authenticated encryption algorithm in the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) and is suitable for lightweight applications. JAMBU_CHAT is developed using Android Studio in the JAVA programming language and Firebase as the backend. It provides message confidentiality and authenticity, so it can detect any modification of the transferred message (ciphertext) and the metadata stored in the database. Therefore, an alert message will be shown to chat users instead of the modified message if the message was tempered or modified during transmission. The ECDH key exchange algorithm is also implemented in this chat application, so the key used for AES JAMBU encryption and decryption application is not hardcoded in the smartphone.

Keywords: Android, Chat application, Authenticated Encryption, ECDH

1. Introduction

Nowadays, people frequently using chat applications like WhatsApp, WeChat, Telegram and Line. Different attacks like eavesdropping, MiTM attack, and impersonation attack can happen on the chat application [1]. Each of them has a different protocol to ensure the messages' confidentiality and integrity that users send or receive. Most of the application uses the end-to-end encryption protocol to ensure the message can only be viewed by the sender and receiver, not even the application's company, for example, WhatsApp, LINE, and Signal. The different chat applications are using different methods to protect the confidentiality and authenticity of the messages. WhatsApp and Signal are using the AES 256 in Cipher Block Chaining (CBC) mode for message confidentiality and Secure Hash Algorithm (SHA) 256 HMAC for message integrity.

*Corresponding author: sapiee@uthm.edu.my

There is a chat application that used the authenticated encryption method to secure the message, which is Line corporation. It uses the authenticated cipher AES 256 in Galois/Counter Mode (GCM) to handle the data encryption and authentication. However, there are two flaws found in GCM mode. The first flaw is when the message is 2K blocks long, the n-bit tag can only ensure the authenticity of n-k bits messages. The second flaw is the GCM mode will lead to a successful forgery when the nonce of it is reused [2].

An authenticated encryption will not show the decrypted message if the input tag is different from the output tag during the decryption process [3]. Therefore, a good authenticated encryption method had to be implemented in the chat application. There is a competition - Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) had been conducted in the year 2012 to identify the authenticated ciphers that can be used in different fields and having a better performance than AES-GCM mode. AES JAMBU is the third-round candidate in CAESAR, which has the feature of lightweight and partial resistance against initialization vector (IV) reuse case [4]. While AES GCM will lose all the confidentiality and integrity of the message once the IV is reused [5].

The objective of this paper is to design and develop a chat application that ensures message confidentiality and authenticity using AES-JAMBU. We called this chat application as JAMBU_CHAT. A functional and security testing will be performed during the development of this chat application. In JAMBU_CHAT, message transfer between users is encrypted end-to-end using ECDH and AES JAMBU to ensure message confidentiality and authenticity [6]. JAMBU_CHAT allow the user to register an account by using their mobile telephone number. The message that saves inside the database is encrypted by using AES JAMBU with the shared key that exchanges between by using Elliptic Curve Diffie Hellman (ECDH). The importance of this paper is to ensure the confidentiality and authenticity of messages in the database by using a more secure authenticated encryption algorithm on chat applications instead of AES-GCM authenticated encryption.

2. Related Work

2.1 Existing Chat Application

The existing chat application that will be discussed in this part would be the WhatsApp, LINE, and Signal chat applications.

WhatsApp is a chat application owned by Facebook. Users can use it to send a text or voice messages, make voice or video calls, and share any documents that up to 100MB. To use WhatsApp, users are required a phone number to sign up for an account for WhatsApp. WhatsApp will send a verification code to the user during the sign-up phase to prove that the phone number is owned by the user [7]. All the communication in WhatsApp is protected by End-to-End Encryption (E2EE). This means only the user and the person who communicate with the user can read the message, not even WhatsApp staff. The End-to-End Encryption of WhatsApp is developed by Open Whisper System called Signal Protocol. Signal Protocol uses Elliptic curve Diffie-Hellman (ECDH) key agreement scheme with Curve25519, AES-256 in CBC mode, HMAC-SHA256, Double Ratchet algorithm to attain end-to-end encryption. The length of the message key is 80-byte, 32 bytes of it will use as AES-256 key, 32 bytes as HMAC-SHA256 key, and the last 16 bytes as an initialization vector.

LINE is a chat application that develops by LINE Corporation. It allows users to send texts, images, videos, voice messages, location and having a free voice and video calls between users. Users can sign up for their LINE account with an email or phone number. LINE requires the user to set their password during the sign-up phase. The password will be used to sign in to other devices or the next time login [8]. LINE protect their user's communication with End-to-End Encryption (E2EE). Also, the E2EE protocol used by LINE is called Letter Sealing. Letter Sealing protocol will locally encrypt the message before it is sent to LINE's messaging server so that even LINE also cannot decrypt the message. Letter

Sealing protocol is using Elliptic Curve Diffie-Hellman (ECDH) key agreement scheme with Curve25519, and AES-256 GCM authenticated encryption cipher.

Signal is a chat application developed by the Signal Technology Foundation and Signal Messenger LLC. Its able user to send a one-to-one message and also have a group chat. Users not only can send a text message but also images, videos, voice records and different types. It also able users to make voice calls and video calls. Signal keep their user's conversation secure by using the Signal Protocol, which provides end-to-end encryption. The Signal protocol combines the Double Ratchet Algorithm, prekeys and Extended Triple Diffie-Hellman handshake to generate the share key between users. Then, the message encryption algorithm used by Signal is AES-256 for confidentiality and HMAC-SHA256 to protect the message integrity. The elliptic curve that used in Elliptic Curve Diffie Hellman of Signal is Curve25519 [9].

2.2 Comparison of existing system

There are three existing chat application has been reviewed before design the chat application which are WhatsApp, LINE, and Signal. Then, the summary of each chat application has been compared with the proposed application in Table 1.

Table 1: Comparison between existing application with the proposed application.

	WhatsApp	LINE	Signal	Proposed Application
Message encryption algorithm	AES-256 in CBC	AES-256 in GCM	AES-256 in CBC	AES-JAMBU
Message hash function	SHA-256	N/A	SHA-256	N/A
Message authenticate	AES-256 in CBC with HMAC-SHA256	AES-256 in GCM	AES-256 in CBC with HMAC-SHA256	AES-128 JAMBU
End-to-End Encryption	Signal Protocol	Letter Sealing	Signal Protocol	ECDH with Secp256r1
Sign up & Login using	Phone number and one-time password (available to add an extra pin number)	Phone number or Email with password	Phone number and one-time password	Phone number and one-time password
Media able to send	Text, voice, images, file, video	Text, voice, images, file, video	Text, voice, images, file, video	Text only

Table 1 shown differences between the security mechanism used by the existing application and the proposed application for protecting the confidentiality and authenticity of the messages. WhatsApp uses AES 256 in CBC and HMAC-SHA256 to ensure the confidentiality and integrity of the message, which uses one more algorithm compared to LINE, which uses AES 256 in GCM mode only. The proposed application is using AES-128 JAMBU. The end-to-end encryption protocol used by WhatsApp and Signal is Signal Protocol. LINE uses its own protocol called Letter Sealing, and the proposed application uses ECDH with Secp256r1.

After that, WhatsApp users register or log in to their account with phone number and one-time password only, and it allows the user to add two-step verification which adds an extra PIN. This method prevents the user's account from brute force attack or dictionary attack since the one-time password is directly sent to the user's phone number. LINE user has to register or log in their account with a phone

number or email with a password. The proposed application is the same as Signal, which requires the user to use a phone number and one-time password to register and log in to the application. The media that are able to send by WhatsApp, LINE, and Signal protocol are text, voice, image, video and file. For proposed application, it is only able to send the text messages.

3. Methodology/Framework

This project will design and develop by using the object-oriented software development methodology. There are Object-oriented software development methodology has four main phases, which are requirement analysis, object-oriented analysis, object-oriented design and implementation and testing.

3.1 Object Oriented Requirement Analysis

In this phase, a use-case diagram has been illustrated to clarify the activity that can perform by the user. Then, the functional and non-functional requirements of this chat application will show in two tables. Table 2 show the functional requirement of the chat application.

Table 2: Functional Requirement Analysis

Module	Functionalities
User Register and Login	User use phone number to register and login the account for the chat application
User profile	Allow user to edit the profile image and display name.
User Lists	Show the list of the users and able the user to search for another users.
Send Message	Send message to the contact.
View Message	Show the history of messages with other users.
Delete Message	Delete the sent message.

Table 2 shown the module that will have in the chat application, which are user register and login, user profile, user list, send, view, and delete message module. The user is using the phone authentication method to register and log in to the chat application. Then, user can update their profile picture and username in the user profile module. After that, the user can select other users to start chatting in user lists. Last, the user is able to send, view, and delete the message that he has sent. Next, Table 3 will show the non-functional requirement.

Table 3: Non-functional requirement analysis

Requirement	Description
Operational	The system only available when there is internet connection
Performance	The message transmit time between user should not exceed 2 second
Security	The user may access the chat application by input the correct OTP (One-Time-Password) that send to the user's phone. The message is encrypted by using AES JAMBU authenticated encryption. The shared key between user will exchange by using Elliptic-curve Diffie–Hellman (ECDH).

Table 3 shown the non-functional requirement for the chat application in the aspect of operational, performance and security. The chat application is only available when there is an internet connection.

Then, the message transmits between users should not exceed 2 seconds. Last, the chat application is using OTP to ensure the user account safety and using AES JAMBU and ECDH to ensure message confidentiality and authenticity.

After that, Figure 1 will show the system architecture design of the proposed chat application that two users able to communicate securely with each other by using the chat application.

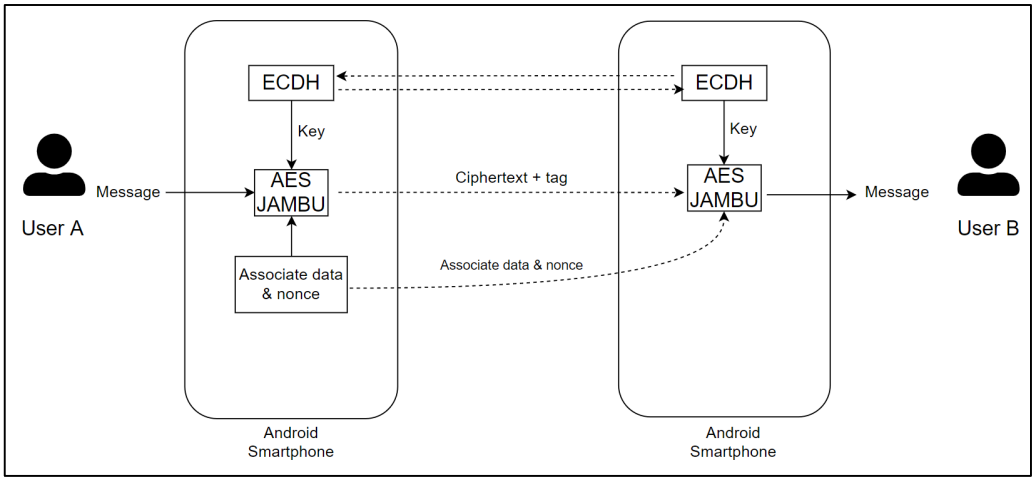

Figure 1: System architecture design of proposed chat application

Firstly, when the user registers to the chat application, it will generate an ECDH key pair. The secret key of the keypair will store inside the device of the user. The public key of the keypair will upload to the Firebase Realtime database along with the user's phone number to create an account in the database. Next, when a user wants to send a message to another user, the sender has to get the receiver's public key from the database and compute the ECDH shared key between them. The shared key is then used as the key for AES JAMBU to encrypt the message. The AES JAMBU will process the message, shared key, associated data, and nonce to output the ciphertext and tag. Then, the receiver will also compute the shared key to decrypt the message. Then, the same associated data and nonce will be used in the decryption process to produce one more tag to verify the authenticity of the message. If the tag during decryption is the same as the tag along with the ciphertext, then the message will display to the user. Else it will be discarded.

3.2 Object Oriented Analysis

There are three object-oriented analysis techniques will be used, which are object modelling, dynamic modelling and functional modelling. In dynamic modelling, a sequence diagram will be drawn to show the process of the chat application follow by time and the data that transmit during each process[10]. After that, an activity diagram will also be created to show the chat application's logic from user register until user close the chat application. The activity diagram for the chat application will show in Figure 2.

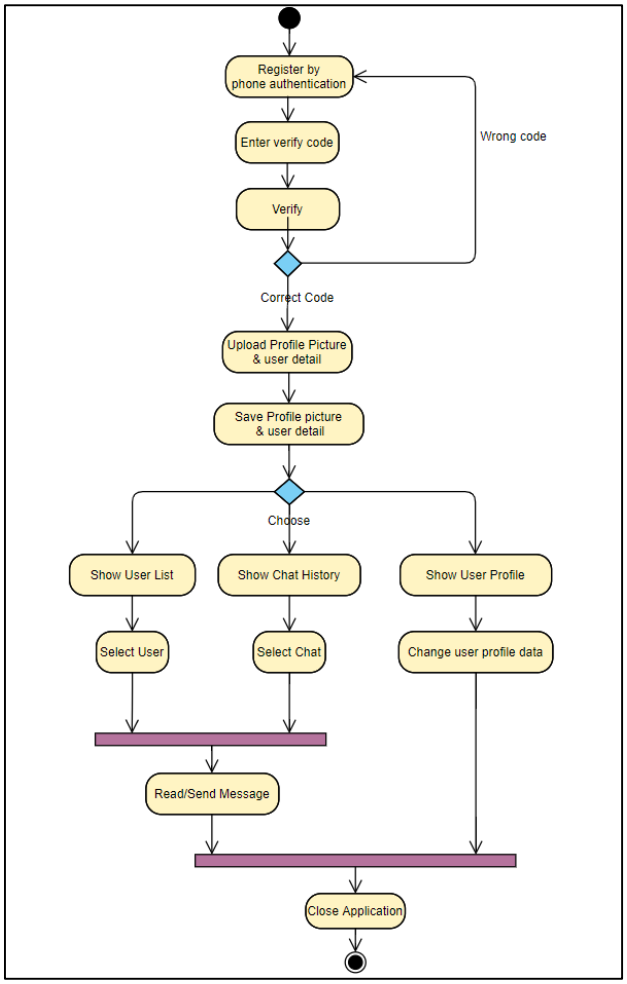

Figure 2: Activity diagram for proposed chat application

The activity diagram has shown the activity diagram for a new user in the chat application. Firstly, the user will go through the phone authentication process. If the user has input the wrong OTP, then it will be required to reenter the phone number for registration. After that, the next time login of the user will be automatically by using the Firebase Authentication service. Then, the user can set up the profile picture and user name. In the main menu, the user can select to show user list, show chat history with other users, or update profile. When the user selects other users in the user list or chat history, he can send a message to the selected user and also view the message history between him and the selected user. In the update profile page, the user can change his profile picture and also the user name. Last, the user can press on back button to close the application.

3.3 Object Oriented Design

In the object-oriented design phase, an ERD (Entity Relationship Diagram) will be created, and the user interface of the chat application will be designed. The ERD will be created by referring to the UML class diagram created in the analysis phases. The ERD will show in Figure 3.

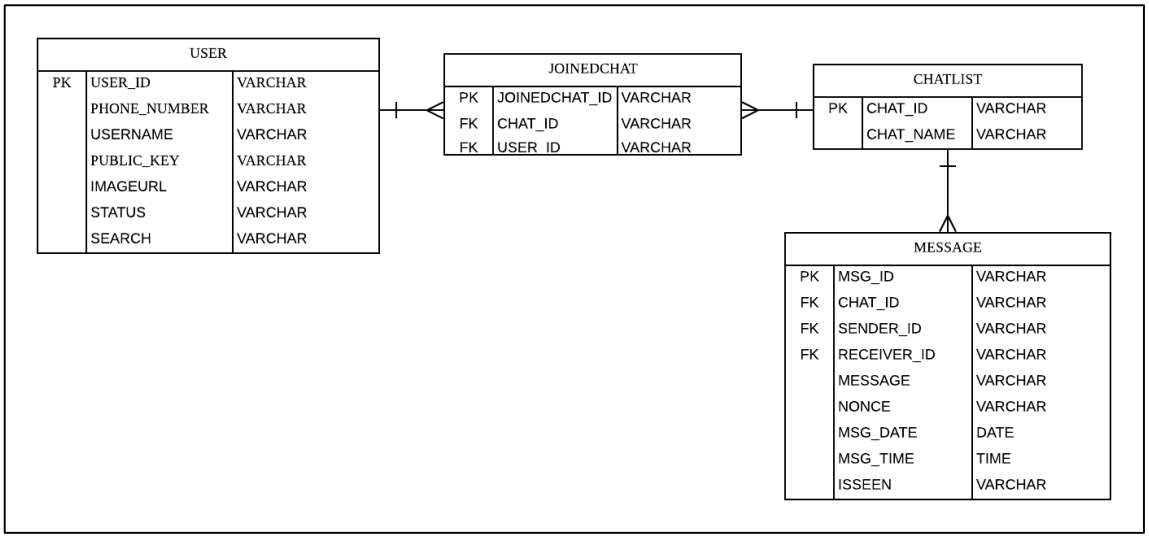

Figure 3: Entity Relationship Diagram for proposed chat application

The chat application is using the Firebase NoSQL database. However, an ERD has been designed to represent the NoSQL database. Each user will have the attribute of user id, phone number, user name, public key, Uniform Resource Locator of profile image in Firebase storage, online status and the user's name in lower case for searching function. Then, the message will have the attribute of message-id, sender id, receiver id, message in ciphertext, nonce, message date, message time, and whether the message has been seen. Then, when a user sends a message to another user, a chat list will be created, and the message between the two users will save the id of the chat list as the foreign key. After that, the two users will have the same chat list, which store in joined chat table.

3.4 Implementation and Testing

The original AES JAMBU is written in C [5]. JAMBU_CHAT is implement using Java programming languages. The coding process will work on Android Studio IDE. For object-oriented implementation, the object is turned into classes code by referring to the UML class diagram created in the analysis phase. The coding for the methods of each class will refer to the sequence diagram and activity diagram designed in the object-oriented analysis phase. Native Development Kit (NDK) and CMake need to be integrated with the Android Studio for Java Native Interface (JNI). Figure 4 shows the installation process of NDK and CMake in Android Studio.

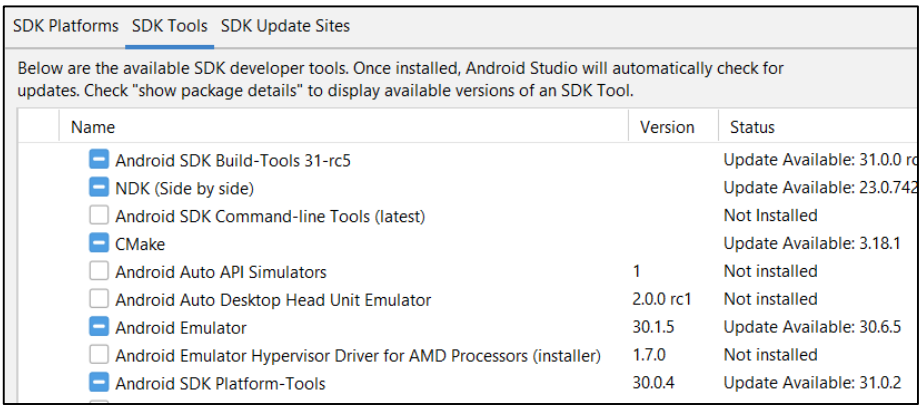

Figure 4: Installation of NDK and CMake

Besides that, JAMBU_CHAT will connect with the Firebase by register the chat application to a Firebase account. Then, the key-value store in the Firebase will design follow the ERD (entity

relationship diagram) that was designed in the object-oriented design phase. Figure 5 show the implementation of Firebase to project in build Gradle file.

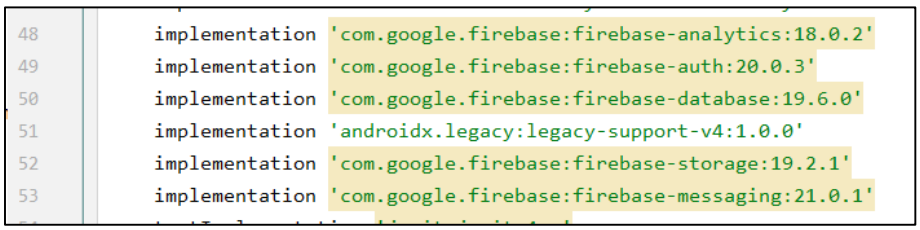

Figure 5: Implementation of Firebase to project

Moreover, to use the algorithm of ECDH will require to implement the Spongy Castle Java open-source library for cryptographic algorithm. Figure 6 show the implementation of Spongy Castle in project.

Figure 6: Implementation of Spongy Castle for ECDH

After this, JAMBU_CHAT will install on the Android smartphone, and a functionality test can be carried out using black-box testing. In black-box testing, different inputs to JAMBU_CHAT and the expected result of it will be included in a test plan. Next, the security checklist will be created to test the functionality of the security feature in the chat application. A user acceptance testing form will also create for a non-developer user to test all the functions in the chat application and record their evaluation towards the function and the security feature of the chat application.

3.5 Software Requirement

The software needed to develop an Android chat application is Android Studio version 4.1 to do the coding work for the chat application and design its interface. Next, the Dev-C++ Version 5.11 is needed to open and run the JAMBU AES algorithm since all the participant CAESAR is written in C programming language [5]. Moreover, Google Chrome is also needed to set up and configure the setting in Firebase. It also uses to view and manage the data that save in the Firebase.

3.6 Hardware Requirement

A 64-bit Microsoft Window 10 laptop with 12 GB RAM, 1 TB Hard Disk Drive, and 1920 x 1080 screen resolution is used as the development environment in this project. Two mobile phones with Android operating system Version 5.0 (and above) and have a minimum of 2 GB RAM and 16 GB internal storage is also required to test the function of the chat application. Two subscriber identification module cards (SIM cards) with the different number are also needed since the chat application is registered by using the phone number.

4. Results and Discussion

In this section, the result of ECDH and JAMBU_CHAT implementation will be shown. The process of the AES JAMBU to authenticate the message will also be shown and discussed.

4.1 ECDH implementation

Each user in JambuChat have a unique ECDH keypair, the private key will store in device and the public key will store in database. Figure 7 show the ECDH key pair generate after user register successfully.

```
321        reference = FirebaseDatabase.getInstance().getReference( path: "Users").child(userid);
322        keyPair = ecdhFunction.generateECKeys();
323        privateKeyHex = ecdhFunction.bytesToHex(keyPair.getPrivate().getEncoded(), keyPair.getPrivate().getEncoded().length);
324        publicKeyHex = ecdhFunction.bytesToHex(keyPair.getPublic().getEncoded(),keyPair.getPublic().getEncoded().length);
325
326        sharedPreferences = getSharedPreferences( name: "User_KeyPair", Context.MODE_PRIVATE);
327        SharedPreferences.Editor editor = sharedPreferences.edit();
328        editor.putString("PrivateKey", privateKeyHex);
329        editor.putString("PublicKey", publicKeyHex);
330        editor.apply();
```

Figure 7: Generate ECDH key pair after registration

Line 321 to 324 will show the ECDH key pair generate by using generateECKeys(). Then, in lines 326 to 330, the encoded private key and the public key will store in the application share preferences in private mode, which is not able to access by other applications. Line 332 to 340 is storing the user data in the database. After that, Figure 8 will show the ECDH key generate function.

```
31 @    public static KeyPair generateECKeys() {
32          try {
33
34              ECNamedCurveParameterSpec parameterSpec = ECNamedCurveTable.getParameterSpec( name: "secp256r1");
35              KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance( algorithm: "ECDH",  provider: "SC");
36
37              keyPairGenerator.initialize(parameterSpec);
38              KeyPair keyPair = keyPairGenerator.generateKeyPair();
39
40              return keyPair;
```

Figure 8: ECDH key generate function

In line 34 has shown the ECDH key parameter is get from the Secp256r1 curve. Then, line 35 show that the key pair is generated using the ECDH algorithm from Spongy Castle. Figure 9 is showing the result of the user public key that has been generated and store in the database.

Figure 9: User's public key store in database

4.2 JAMBU_CHAT Implementation

JAMBU_CHAT is a chat application that encrypts user's messages with AES JAMBU to ensure message confidentiality and authenticity. Figure 10 shows the code segment for the message encryption by using AES JAMBU before it is saved to the database.

```
211         //Encrypt message and save to database
212         reference = FirebaseDatabase.getInstance().getReference( path: "Users").child(userid);
213         reference.addListenerForSingleValueEvent(new ValueEventListener() {
214             @Override
215             public void onDataChange(@NonNull DataSnapshot snapshot) {
216                 User user = snapshot.getValue(User.class);
217                 receiverPublicKeyString = user.getPublicKey(); //get receiver's public key
218                 byte[] shareSecret_256 = ECDHFunction.getShareSecret(senderPrivateKeyString, receiverPublicKeyString); //Compute s
219                 shareKey = Arrays.copyOfRange(shareSecret_256,  from: 0,  to: 16); //truncate share secret to 128bit
220                 String encmsg = nativeLib.encryptMessage(message, shareKey,  ad: fuser.getUid()+current_date+current_time, nonce);
221
```

Figure 10: Encrypt message before send

In line 218, the code is computing the share secret between sender and receiver to use as the key in encryptMessage() in line 220. Line 220 has input the plaintext, share secret, associated data, and nonce for AES JAMBU encryption in JNI. Then, only the encrypted message saves to the database. Figure 11 shows the code segment of encryptMessage() in JNI. Then, Figure 12 show the ciphertext in hex string return by the encryptMessage().

```
857 //Function use in Message encrypt
858 extern "C"
859 JNIEXPORT jstring JNICALL
860 NativeLibrary.encryptMessage(JNIEnv *env, jobject NativeLibrary  thiz, jstring message,
861                                                                   jbyteArray key, jstring ad,
862                                                                   jbyteArray nonce) {
863     int result = 0;
864     int i = 0;
865     unsigned long long alen = 0;
866     unsigned long long mlen = 0;
867     unsigned long long clen = CRYPTO_ABYTES;
868     //Retrieve assiciated data
869     const char *assocdata = env->GetStringUTFChars(ad,  isCopy: 0);
870     unsigned char a[strlen(assocdata) + 1];
871     strcpy(reinterpret_cast<char *const>(a), assocdata);
872     //Retreve message string
873     const char *msg = env->GetStringUTFChars(message,  isCopy: 0);
```

Figure 11: encryptMessage() in JNI

```
886
887     //Encrypt using AES JAMBU
888     result |= crypto_aead_encrypt(c, &clen, m, mlen, a, alen, nsec, npub, k);
889
890     //Convert Ciphertext to Hex String
891     char hex[mlen+CRYPTO_ABYTES];
892     for(i = 0; i < mlen+CRYPTO_ABYTES; i++)
893         sprintf( dest: hex+2*i,  format: "%.2x", c[i]);
894     string enc_str(reinterpret_cast<const char *>(hex));
895
896     return env->NewStringUTF(enc_str.c_str());
897 }
```

Figure 12: Return ciphertext hex string

Line 869 to 871 in Figure 11 shows the process of converting “ad” in jstring or Java string to string in C that is able to use in AES JAMBU encryption. This process also same goes through on “message”, “key”, and “nonce”. Line 888 in Figure 12 has shown the message encrypt by crypto_aead_encrypt(), which is also the AES JAMBU source code method in CAESAR. Then, lines 891 to 894 in Figure 12 will convert the ciphertext from byte to string and return the string in line 896.

After encryption, the ciphertext hex string will store along with the message metadata like message date, time, and nonce to the database which shown in Figure 13.

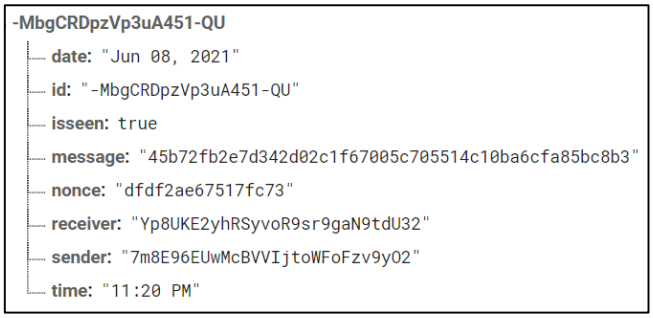

Figure 13: Encrypted message stored in database

The encrypted message is the ciphertext with a tag. The tag is the last 8 bytes of the ciphertext, "c10ba6cfa85bc8b3" is the tag for the message in Figure 13. The encrypted message will send along with the associated data like the date, time, sender id, and nonce to use in the decryption process. For message decryption, the data of the message will get from the database. Then the shared key will be computed, which same in the encryption process. Figure 14 shows the decryption of message when receiving an encrypted message.

```
byte[] ciphertxt_byte = decodeHexString(chat.getMessage()); //Change ciphertext Hex string to byte
byte[] nonce = decodeHexString(chat.getNonce());
chat.setMessage(nativeLib.decryptMessage(ciphertxt_byte, shareKey,  ad: chat.getSender()+chat.getDate()+chat.getTime(), nonce));
```

Figure 14: Decrypt message when receive

The ciphertxt_byte in Figure 14 is the ciphertext in a byte that decode from the hex string of encrypted message that store in the database. Then, the nonce in byte is also decoded from the hex string of nonce that is stored in the database. After that, the decryptMessage() will process the ciphertext, shared, key, associated data, and nonce to give an output that either the original message or an error message. Figure 15 has shown the decryptMessage() in JNI.

```
929     //Decrypt using AES JAMBU
930     result |= crypto_aead_decrypt(m, &mlen, nsec, c, clen, a, alen, npub, k);
931
932     string ascii = ""; //Declare variable for return the plaintext
933     //Set return plaintext as error message if tag verification failed
934     if(result!=1){
935         ascii ="Error: the message has been modified";
936     }else{
937         //Change each character in plaintext to hex string
938         char hex[mlen];
939         for(i = 0; i < mlen; i++)
940             sprintf( dest: hex+2*i,  format: "%.2x", m[i]);
941
942         string msg_hex(reinterpret_cast<const char *>(hex));
```

Figure 15: decryptMessage() in JNI

Lines 930 in Figure 15 is the decryption of message using crypto_aead_decrypt(), which is also the AES JAMBU decryption method in CAESAR. The last phase in the crypto_aead_decrypt() will check whether the decrypted message tag is same as the tag that come along with the ciphertext. If the verification is success the return value of this method would be 1. If failed, it will return -1. So, line 934 has set the rule that if the return value is other than 1, the message will be set to the error message in line 935 and return to the user. Else, the original message will return to users. The error message that will show to users will show in Figure 16.

Figure 16: Error message if tag verification failed.

Figure 16 show the error message that show to user if the tag verification in the last phase of decryption process has failed.

4.3 Application Interface

Figure 17 show the interface for message activity in the chat application.

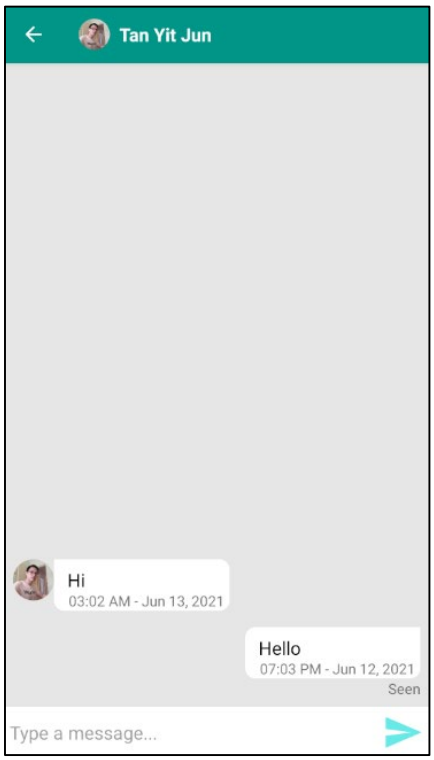

Figure 17: Interface of Message Activity

In Figure 17, the tab above the message activity will show the name of the receiver that the sender is chatting with. Then, the message that sends to the receiver will show on the right side, while the right side is the message from the receiver.

The messages that store in the database are in ciphertext form. The key that uses to encrypt the message is the ECDH shared secret between the sender and receiver. So, even the database admin would not be able to decrypt the messages between users. After that, any modification of the message during transmits or on the store will cause the message to change to an error message that alerts the receiver and sender. This is due to the AES JAMBU will verify the tag of the messages during the decryption process. Figure 18 has shown two users chatting in AES JAMBU, and Figure 19 show the two users' last message that has been encrypted and store in database.

Figure 18: Chat between two users

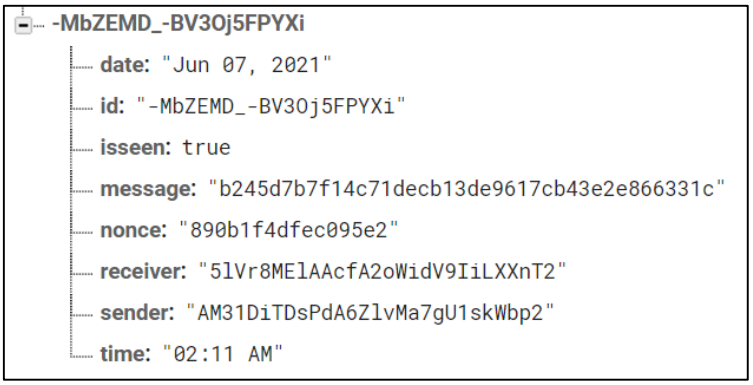

Figure 19: Last message data that store in database

In this above context, if the encrypted message has been modified even one bit, the message that shows on the user's chat page will become the error message that alerts the user that the message has been compromised. Figure 20 shown the modified message with one bit in the database. Figure 21 show the result that will show on the user's chat page.

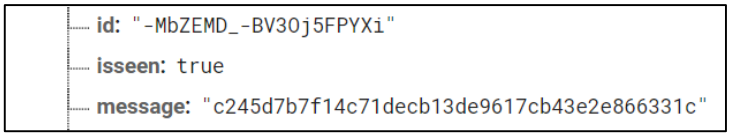

Figure 20: Modified last message that store in database

Figure 21: Error message after modified message

Figure 20 shown that the original data "b255d7…" has been a change to "c245d7…", the one-bit data change in the message data will cause the message not able to show to users. Instead, an error message will show to alert the users, which show in Figure 21.

4.4 Test Plan Result

After the application has been developed, functional testing has been performed to examine the functionality of the application. The testing is to identify any error that happen when using the chat application and also check whether the application has achieved the project objective and scope. Table 4, Table 5, and Table 6 has shown the summary result of the functional testing on the chat application.

Table 4: Test Plan for Phone Authentication Module

No.	Description	Expected Result	Actual Result
1	Register – Input phone number and add alphabet. Phone number = 01110688a84	Message appear: Please enter a valid phone number.	Pass
2	Register – Input phone number and add symbol Phone number = 1110688@84	Message appear: Please enter a valid phone number.	Pass
3	Register – Input a real phone number. Phone number = 01110688984	Message appear: Verify code has send, please check your message.	Pass
4	Verify code – Input a wrong verification code.	Message appear: Invalid verification code.	Pass
5	Verify code – Input a correct verification code.	Redirect to User Startup page.	Pass

Table 4 shown the testing result for the phone authentication module. The application is able to check whether the phone number that input by the user is valid or not. The error message is output if the user is input an invalid phone number. Then, the user is able to receive the verification code via SMS (short message service). The application is also able to check whether the verification input by the user is correct and redirect the user to the user-startup page.

Table 5: Test Plan for Message Activity Module

No.	Description	Expected Result	Actual Result
1	Click send button without input anything (In user 1 device)	Message appear: Please input your message. (In user 1 device)	Pass
2	Input “abcd” and click send button. (In user 1 device)	Message sent and show “abcd” which send by user 1 in chat history. (In user 1&2device)	Pass
3	Input “efgh” and click send button. (In user 2 device)	Message sent and show “efgh” below previous message and send by user 2 in chat history. (In user 1&2device)	Pass
4	Select “abcd” message and delete (In user 1 device)	Message deleted. (In user 1&2 device)	Pass

Table 5 shown the testing result for the message activity module in the chat application. The test case for showing the error message when the user did not input anything in the textbox has success. Then, the user can send, read, and delete the message without any problem.

Table 6: Test Plan for Profile Module

No.	Description	Expected Result	Actual Result
1	Upload a profile picture. (In user 1 device)	User 1 profile picture updated. (In user 1&2 device).	Pass
2	Update the user's name. (In user 1 device)	User 1 user's name updated. (In user 1&2 device)	Pass

Table 6 has shown the updated profile module in the chat application. The functionality for update the profile picture and user name has been tested and passed. After that, the chat application also goes through a security checklist for testing the functionality of the message encryption. Table 7 show the summary of security checklist result for the message encryption.

Table 7: Security check list for proposed system

No	Check List	Actual Result
1	All message in the database is show in ciphertext.	Pass
2	Modified a bit of the encrypted message in database. The modified message will show as "Error: This message has been modified" instead of the message".	Pass
3	Replace a message in database with other chat's message. The replace message will show as "Error: This message has been modified" instead of the message" in user device.	Pass

Table 7 shown the security checklist for the chat application. The message stored in the database is all in ciphertext form. Then, any modification of the encrypted message is able to detected in the application and show the error message to users.

5. Conclusion

AES JAMBU has been successfully implemented on Android smartphones as the authenticated encryption to ensure confidentiality and authenticity of chat messages instead of using a symmetric key encryption and a hashed based message authenticated code or the AES-GCM authenticated encryption.

The advantage of JAMBU_CHAT is that if any modifications are made to the message (ciphertext) during transmission, the error message will be shown to the recipient in the chat. This is the significant advantage of using authenticated encryption, where Tag is used to ensuring the integrity of the transferred message (ciphertext).

For future work, JAMBU_CHAT can be added with additional features such as group chat and various media attachments (image, video, voice record, and document file).

Acknowledgement

The authors would like to thank the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia for its support and encouragement throughout the process of conducting this study.

References

[1] M. Kuliya and H. Abubakar, "Secured Chatting System Using Cryptography," International Journal of Creat. Res. Thoughts, vol. 8, no. 9, pp. 23–26, 2020.

[2] N. Ferguson, “Authentication weaknesses in GCM,” Comments Submitt. to NIST Modes Oper. Process, pp. 1–19, 2005.

[3] P. Sarkar, “A Simple and Generic Construction of Authenticated Encryption with Associated Data,” ACM Trans. Inf. Syst. Secur., vol. 13, no. 4, pp. 1–16, Dec. 2010.

[4] T. Peyrin, S. M. Sim, L. Wang, and G. Zhang, “Cryptanalysis of JAMBU,” in International Workshop on Fast Software Encryption, 2015, pp. 264–281.

[5] H. Wu and T. Huang, “The JAMBU Lightweight Authentication Encryption Mode (v2.1),” Div. of Math Sciences., Nanyang Technological University, Singapore, 2016.

[6] C. Easttom, “An Overview of Key Exchange Protocols,” IOSR J. Math., vol. 13, no. 4, pp. 16–18, 2017.

[7] T. Sutikno, L. Handayani, D. Stiawan, M. A. Riyadi, and I. Much Ibnu Subroto, “WhatsApp, Viber and Telegram which is Best for Instant Messaging?,” Int. J. Electr. Comput. Eng., vol. 6, no. 3, p. 909, Jun. 2016.

[8] C. M. Sang and C. C. Yen, “Forensic analysis of LINE messenger on Android,” Journal. Computing., vol. 29, pp. 11–20, 2018.

[9] V. McCall, “What to know about Signal, the secure messaging app that keeps all of your conversations private,” Business Insider, 2021. [Online]. Available: https://www.businessinsider.com/what-is-signal. [Accessed: 25-Mar-2021].

[10] Muhammad Umair, “Object Oriented Analysis and Design,” Code Project, pp. 1–10, Dec 11, 2018.[Online]. Available: Code Project, https://www.codeproject.com/Articles/1137299/Object-Oriented-Analysis-and-Design. [Accessed Sept 15, 2020].