Peer to Peer Secure Communication in Mobile Environment: A Novel Approach

Ashutosh K. Singh, Shubham Maheshwari ,S. Verma and Rahul Dekar
Indian Institute of Information Technology,Allahabad,India

ABSTRACT

Peer-to-Peer (P2P) communicating is a networking and distributed communication paradigm which allows symmetric sharing of messages between two entities, ideally without any interception from any third entity. Privacy in peer-to-peer communication is, however, can be compromised via a large number of approaches.It is thereforea need of the hour to provide security against eavesdropping and electronic surveillance while exchanging confidential and/or personal information. SMS or Short Message Service is the most widely used mode of communication for information exchange around the globe. In this paper, an approach to android application is presented which will be helpful for users of android based mobile phones to share textual information via SMS without the fear of it being intercepted by any middle-man. A cryptography technique is implemented which avoids the leakage of clear messages that is being transferred from one end to the other, by encrypting it using an already exchanged private key, thereby making the middle-man unable to decipher the intercepted information. The private key exchange is done using the Diffie-Hellman key exchange algorithm, and AES-128 is used as the encryption/decryption algorithm. The messages sent and received are stored in a database accessible by the application, for future reference to the messages. The implemented application was successfully tested on Android Virtual Machine as well as on mobile phone running android version 2.3. The message received was a random jumble of characters and the original message was available only through the application.

General Terms

SMS communication, android, secure mobile communication

Keywords

android, cryptography, mobile communication, security, sms communication

1. INTRODUCTION

Secure communication of information is the prime concern in the military and top business class persons. Started by IBM and Bell Labs, the era of cryptography is at its peak in the modern world. Seeing the increasing demand for ways of secure communication, cryptographic applications have found their way into a variety of domains, viz. ATM, credit cards, EDGE, GSM, SIM, computer networks, even native desktop applications. In this paper, an approach is proposed to turn simple and widely used SMS channel of communication into a secure way of exchanging information.

Cryptography has been the core of security framework in Mobile Ad hoc Network (MANET)[1]. The key management for peer-to-peer secure communicationin MANET has been a hot topic of research, owing to the limitations on energy and computational capabilities of the devices. The computational capabilities of mobile devices have, however, now been increased by a much substantial number, and is expected to increase in the upcoming years. This facilitates the implementation of much complex architectures for MANET.

Short Message Service, or commonly known as SMS, isthe most popular and widely used mode of exchanging information via mobile devices. Unfortunately, the SMS technology does not provide a built-in support for any security feature[2]. Addition of a layer on the SMS technology, which ensures the security of the exchanged information, would allow SMS messages to be used in applications where confidentiality and authenticity of the messages must be insured.

Open Handset Alliance (OHA) hosting members like Google, Motorola, HTC etc. released an open source platform Android for mobile devices[3]. It runs on a linux shell with a customized JVM that sits on top of the underlying kernel. Continued efforts are made to make the new operating system more and more energy efficient, so as to provide the mobile devices with better battery life. Android, a mobile operating system for both netbooks as well as for other embedded devices, allows developers to create applications in Java language, giving access to device's hardware via specific hardware drivers. It is gaining popularity among the consumers rapidly, owing to the availability of a wide range of applications and expertise. Because Android is open source, there is a lot more freedom around who can use it on their device and who can develop apps for it, as well as more freedom to adapt and customise it. This makes it the perfect candidate to be worked on for creating the SMS encrypting app to solve our purpose.

Android allows us to gain access to the incoming and outgoing SMS's. Thus, we can encode the textual information required to be sent in the background before actually sending it, and similarly, decode the text received in the background before displaying it to the end-user. This also makes the application user friendly, thereby enabling the user with no technical knowledge use it effortlessly.

For the encryption/decryption to work, a symmetric key needs to be exchanged between the communicating peers. This can also be done in the background, on a single click of a dedicated button. While the user types the required text to be sent, the app generates a symmetric key for the pair of communication peers using the Diffie-Hellman key exchange algorithm. Once the key has been successfully exchanged, the corresponding two end-users can use it any number of times to communicate information.

Storage of the messages communicated can also be done. Android provides a full support of the SQLite database, which is used for storing the received messages along with their encryption keys. Thus, user can easily access the received messages anytime in future. An option to delete the saved messages is also provided in case the user wants to delete some or all of them. The messages are stored in the .db files maintained by the application.

Since the app stores the messages and the key to decrypt them, a certain method must be used to prevent unauthorized access to the

mobile app. An authorization screen is designed for the same.

Figure 1. System Architecture of Android[8]

2. ANDROID

Google's Andy Rubin describes Android as:

"The first truly open and comprehensive platform for mobile devices, all of the software to run a mobile phone but without the proprietary obstacles that have hindered mobile innovation."[4]

Android[5] software platform incorporates a Linux kernel that is responsible for providing the base functionalities like memory management, processing, networking, storage, etc. to the system, a middleware layer that supports the hosted applications, and finally some core applications. The middleware layer has the native Android libraries (written in C/C++), core java libraries (written in Java) and Dalvik Virtual Machine, an optimized version of JVM, which is responsible for the execution of the binaries of higher layer applications. Android applications are written in Java and consist of separated modules, so-called components. Android middleware known as Binder, which is a reduced implementation of Open Binder, provides an Inter Component Communication (ICC) mechanism which enables the components to communicate with each other and also to the components of other applications. [6]

2.1 System Architecture of Android

●Applications: Android phones generally come with a set of preinstalled applications for the basic functioning of the mobile phone. These apps include an e-mail client, an SMS management app, WebKit based browser, music player and a picture gallery, basic camera and video recording application, home screen, a calculator and an alarm clock. All these apps are a part of Android Open Source Project (AOSP).

● Application Framework: The Android SDK provides the developers with the APIs to gain access to all the hardware components of the mobile device, such as camera, Wi-Fi, power management, accelerometers, network connections, etc. [7]. This gives developers the freedom to use the mobile device's hardware in any way.

●Libraries: Android's middleware consists of c/c++ libraries, which are used by different components/application modules of the android. These Libraries are accessible to developers via the use of corresponding API calls[8].

●Linux Kernel:Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack[8].

Android also provide a mechanism to run apps invisibly, in the background. This facilitates the developer to come up with an event-driven architecture for the application, wherein the app would be running in the background, waiting for the event to happen. It is a perfect scenario for screening incoming calls and SMSs before being notified on the screen.

As mentioned before, Android also provides facility for efficient data storage and retrieval via SQLite Database. SQLite is a lightweight relational database available for each application. Since each application's database is sandboxed – availability of the database's data is limited to the corresponding application only – developers can securely use it for data storage. There is, however, a mechanism for managed sharing of the databases using Content Providers.

2.2. Android application fundamentals

Applications for android platform are coded in Java Programming language, and all the code is packed in a '.apk' file which is the file extension for installable file of the android applications. Upon installation, each application lives in its own sandbox and work in isolation from other processes.

2.3. Application Components

They are the essential building blocks of an Android application. They are of four types[11]:

1. Activity: An activity represents a single screen with a user interface. It is analogous to the canvas in J2ME where we can place any draw able item such as button, Text field, etc and many more user interaction components.

2. Service: A service is a component that runs in the background to perform long-running operations or to perform work for remote processes.

3. Content provider: A content provider manages a shared set of application data.

4. Broadcast receiver: A broadcast receiver is a component that responds to system-wide broadcast announcements.

There is some permission required for every application if it is using some hardware intensive resource and to be explicitly added by developer in the manifest.xml file.

3. AES (ADVANCED ENCRYPTION STANDARD)

This standard specifies the Rijndael algorithm, a symmetric block cipher that can process data blocks of 128 bits, using cipher keys of length 128, 192 and 256 bits.

The Advanced Encryption Standard (AES) specifies a

FIPS-approved cryptographic algorithm that is available for use in protection of electronic data, mobile or static. The AES algorithm is a symmetric block cipher algorithm that encrypts (encipher) and decrypts (decipher) information. Encryption converts data to an unintelligible form called cipher text; decrypting the cipher text converts the data back into its original form, called plain-text[9].

AES implementation is provided in JAVA. Java Cryptography Extension (JCE) was integrated with the SDK and the JRE, beginning with the Java 2 SDK, Standard Edition (J2SE) v1.4.0. Thus, AES can be used as any other cipher in Java programs.

4. DIFFIE-HELLMAN KEY EXCHANGE ALGORITHM

DiffieHellman[10] is a protocol, explained in Algorithm 1, which allows two users to share a secret key, over an unsecure communication channel, without any prior secrets.

The prerequisites requires for the algorithm is to have a Prime number (p), and a Generator (g), with the condition that 'p' > 'g' and 'g' is a Primitive Root of 'p'.

Algorithm 1.Diffie Hellman key exchange algorithm

1. Alice and Bob agree on 'p' and 'g'.
2. Alice chooses a random natural number 'a' and sends '(ga mod p)' to Bob.
3. Bob chooses a random natural number 'b' and sends '(gb mod p)' to Alice
4. Alice calculates key as '((gb)a mod p)'
5. Bob calculates key as '((ga)b mod p)'

5. APPROACH

The workflow for the implementation of the application is as follows:

1. To begin with, the authorization of user by the application is essential. The user would also have an option to change the login password of the installed application.

2. If Authorized he/she may send or receive encrypted messages.

3. On the Sender's end if a key is already present for encrypting the message for a session (session is defined as the continued duration between the starting and closing of application) for a particular receiver then user can directly send the encrypted message. Go to step 5.

4. Else sender has to first do the key exchange with the receiver. Then Go to Step 3.

5. On the Receiver's end, when key not present and key exchange takes place. It firstly receives a Key Exchange message having p, g, Y_a of the Deffie Hellman Key Exchange and index of p and g, it then calculates it's Y_b then sends Y_b to the sender from which key exchange request was initialized. At initiator's end the message received is Get_Yb message. Else the message will be encrypted message and receiver decrypt the message and displays it in Toast.

6. Further receiver can become sender and vice versa accordingly.

7. Any user can access the database where different senders have sent him the messages and he can also get access to the particular message and do delete operation on that message.

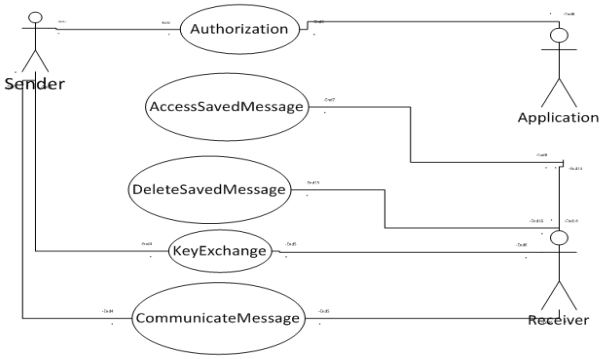

Figure 2. Use Case diagram for the application

5.1 Authorization

It happens by saving the password to a permanent file at the installation time. Every time on application start-up, username and passwords are matched accordingly and only then there is an access to the application components. At every change password option selected old passwords and new passwords are provided by the user and the permanent file is updated.

1. Select a random number 'r' which indicates the index of array (varying from 0-9) of keys_p/keys_g to be selected
2. Select a random number a of 20 digits
3. Calculate Ya as
   Ya = g.modPow(Xa,p)
   where:
   *modPow: a function in BigInteger class of Java
   *g: value of keys_g at location 'r'
   *p: value of keys_p at location 'r'
4. Create key exchange message by appending Key_Exchange flag, r, and Ya, and send it to receiver.
5. Wait for receiving Yb from receiver
6. when Yb is received, calculate key as
   key = Yb.madPow(Xa,p)

1. Receive the key exchange message. Extract r and Ya from it
2. Calculate Xb, i.e. random number of 20 digits
3. Calculate Yb as
   Yb = g.modPow(Xb,p)
   where:
   *modPow: a function in BigInteger class of Java
   *g: value of keys_g at location 'r'
   *p: value of keys_p at location 'r'
4. Create a Get_Yb message by appending Yb to it and send it to sender
5. When received calculate key as
   key = Ya.modPow(Xb,p)

Algorithm 3.Key handshake algorithm at receiver

5.2 Key hand shake

Whenever a session (as defined above) begins for a sender to send an encrypted message a key is required to encrypt it and the same key would be required by the receiver to decrypt the message for further processing with the message. This is provided by key hand shake algorithm. We have already maintained keys_p&keys_g String arrays of size 10 where keys_p contains prime numbers of 128 bit and keys_g contains the primitive root of corresponding keys_p in our application. At the sender end Algorithm 2 is implemented.
At receiver's end the Algorithm 3 follows.

5.3 Encryption/decryption of messages

For encryption/decryption, the in-built java libraries are used with a 128-bit key.

Algorithm 4 for encryption and Algorithm 5 for decryption is followed.

Algorithm 4.Encryption of message to be sent

1. Generate a key
   a. If key is more than 128-bit long, truncate it to 128-bit by discarding the Least Significant Bits
   b. Else if key is less than 128-bit long, do padding at Least Significant Bit
   c. Store it to x
2. SecretKeySpec key = new SecretKeySpec(x, "AES")
3. Cipher cip = Cipher.getInstance("AES")
4. Cip.init(Cipher.ENCRYPT_MODE,key)
5. Byte[] encrypted = cip.doFinal(message.getBytes())
6. message = Base64 encoding of 'encrypted'

1. Generate a key
   a. If key is more than 128-bit long, truncate it to 128-bit by discarding the Least Significant Bits
   b. Else if key is less than 128-bit long, do padding at Least Significant Bit
   c. Store it to x
2. SecretKeySpec key = new SecretKeySpec(x, "AES")
3. Cipher cip = Cipher.getInstance("AES")
4. cip.init(Cipher.DECRYPT_MODE,key)
5. byte[] enc = base64 decoding of the received message
6. message = cip.doFinal(enc)

Algorithm 5.Decryption of message received

5.4 Handling Failures

A key requirement in such an application is support an unplanned disconnection of one of the peers from a session[12], as the communication may fail between the two atomic processes - key handshake and send encrypted message.

●When it comes up, sender may send the encrypted message to receiver assuming it may have the key. But as according to our assumption, key must be there with an application for a particular session. So, receiver may not decrypt the encrypted message. Hence, it discards the message and sends a KEY_NOT_FOUND_EXCEPTION_DELETE_KEY message to sender. And sender then do key hand shake and then again send the encrypted message.

●In case the application which comes up the second time sends a request for key exchange and if the key is already present at the receiver's end, the receiver's key is deleted and new key is exchanged.

5.5 Maintaining Database

Database is maintained for the application. Android provides the full support of the SQLite database. A database whenever 1st

sender sends a message. It is stored in application as .db file. Emulator/device instances store SQLite3 databases in the folder /data/data/<*package_name*>/databases.We need database in our application to access the encrypted messages. In our case Database contains two types of the table.

Type 1. Table TABLE_SENDER whose fields are (phone TEXT PRIMARY KEY)

Type 2. Table TABLE_NAME whose fields are (id INTEGER PRIMARY KEY (auto-incremented), message TEXT , key TEXT)

Where TABLE_SENDER will be storing the table name "TABLE_SENDER" and TABLE_NAME will be storing ("TABLE_NO" + phone). Thus, there will be as many table of type 2 as there are number of senders.

Using database will also help us to delete messages very easily.

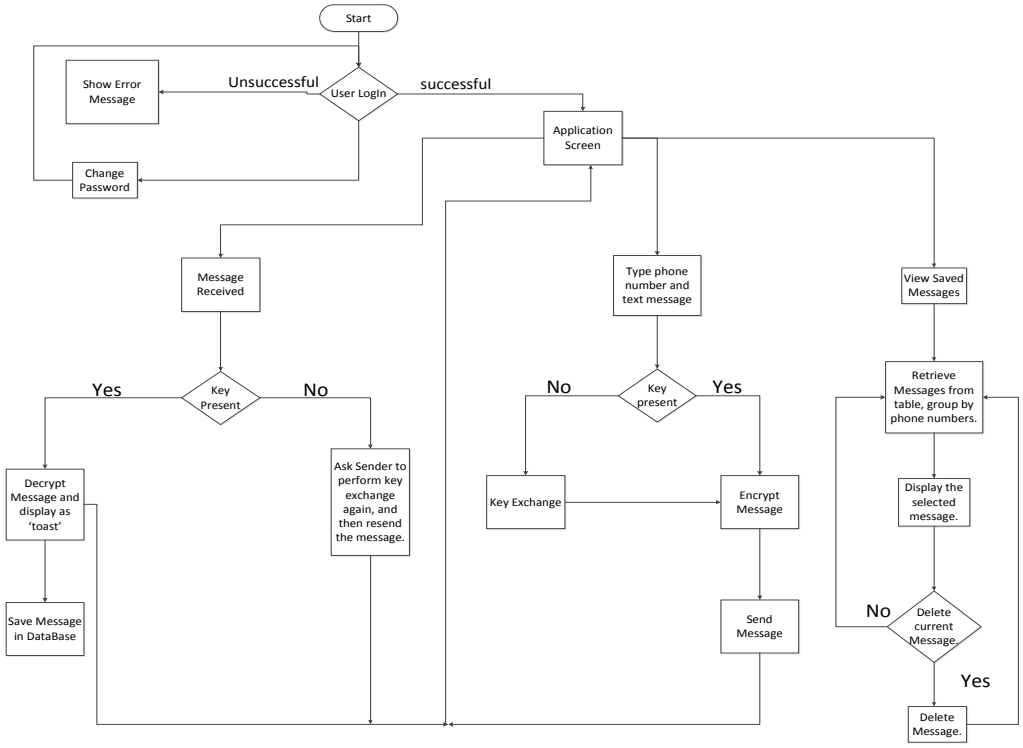

Figure 3. Flow Chart diagram for the application

6. CONCLUSION

The login screen is successfully made. Key Hand Shake is being implemented. And as a result keys can be generated for a particular session for a particular sender. Hence forth, the encryption of message through the 128 bit AES encryption and decryption of a message is done using in built Java Libraries (JCE). Some exception handling also being done by the application in case when one node goes down and when it comes up it does not has the key for the recent session and other node may send encrypted message without having knowledge that the other node has key or not, then a KEY_NOT_FOUND_EXCEPTION_DELETE_KEY message is being sent in that case which force the other end to delete it's key and resend the message in a new session. In case the application which comes up sends a request for key exchange the key if present at receiver's end is deleted and new key is exchanged.

A database is maintained for storing the messages in accordance with the sender phone numbers. By this user can delete or access any message.

The actual message received is some random set of alphabets, illegible to any normal person. Only after decrypting the message received, using the corresponding exchanged key, would the original message be revealed. Thus our original goal of having an application to secure information exchange via use of the most widely method of communication, SMS, is achieved.

Some previous works have been done in providing methods for secure communication, but our application utilizes the most commonly used mode of communication, SMS, and converting it into a secure mode of information exchange, it opens up a wide variety of other applications SMS could be used in.

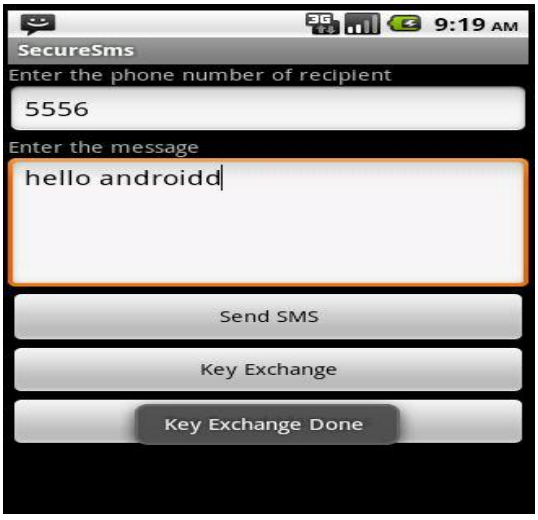

Figure 4. Snapshot of the application's main screen on successfully doing a key exchange

Since the message is converted into a random set of characters before entering into the network channel, same method can also be used for communicating between peers sitting behind a Network Address Translator (NAT)[13].

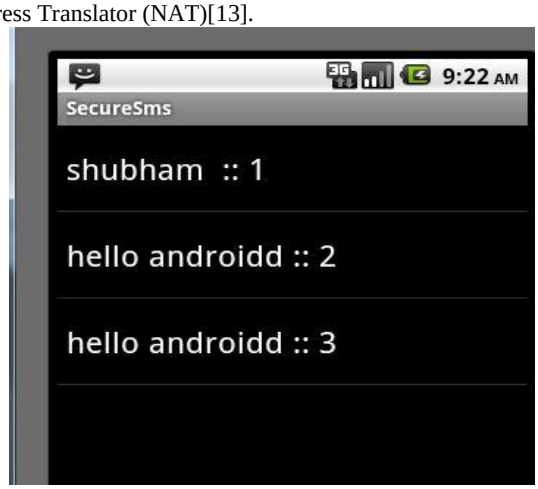

Figure 5. Snapshot of the application's stored messages management screen showing the stored messages (with their corresponding serial number)

7. FURTHER WORK

The application can be further extended to do secure communication by means of voice calls also. That is if third party may tap the phone call then he/she may not understand the communication between the sender and receiver. Also, VOIP (Voice over Internet Protocol) can be utilized so that the messages are not being sent as SMS, instead they are being sent through the socket where IP of the phone will be required.

The application approach can be further extended to work with non-symmetric key algorithms, thus eliminating the need of exchanging a symmetric key before the start of communication.

8.REFERENCES

[1] Patnaik, G.K.; Gore, M.M.Tree-Like Peer-to-Peer Symmetric Key Management in Mobile Ad Hoc Network in Networks and Communication, NETCOM. December 2009.

[2] De Santis, A.; Castiglione, A.; Cattaneo, G.; Cembalo, M.; Petagna, F.; Petrillo, U.F.An Extensible Framework for Efficient Secure SMS in Complex, Intelligent and Software Intensive Systems (CISIS), February 2010.

[3] Paul, K.; Kundu, T.K.Android on Mobile Devices: An Energy Perspective in Computer and Information Technology (CIT), IEEE 10th International Conference, June, 2010.

[4] http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html

[5] Android. http://www.android.com.

[6] Palm Source, Inc. Open Binder. Version 1. http://www.angryredplanet.com/~hackbod/openbinder/docs/html/index.html, 2005.

[7] Reto Meier. Professional android application development. Indiana: Wiley Publishing, Inc. 2009, pp. 6.

[8] developer.android.com/guide/basics/what-is-android.html

[9] Federal Information Processing Standards Publication 197. November 26, 2001.

[10] New Directions in Cryptography. W. Diffie and M. E. Hellman, IEEE Transactions on Information Theory, vol. IT-22, Nov. 1976, pp: 644–654

[11] J.F. DiMarzio. Android, A Programmer's Guide. MCGraw-Hill Osborne

[12] Abdul-Mehdi, Z.T.; Mahmod, R.Security Management Model for Mobile Databases Transaction Management in Information and Communication Technologies: From Theory to Applications, April, 2008.

[13] Sunghyun Yoon; Soon Seok Lee; Sang-Ha Kim. Seamless and secure P2P communication scheme for mobile Internet devices behind NAT in Consumer Electronics (ICCE), 2012 IEEE International Conference, January, 2012.