Incorporation of Cryptography Mechanism in a Smartphone File Sharing System Design

Nitesh Rijal

Department of Computer Science, Jawaharlal Nehru Technological University
Kakinada, Andra Pradesh, India
rijal.it@gmail.com

***Abstract*— Modern day technology has become very advanced. Internet has become accessible to almost every corner of the world. Electronic Mail, commonly known as "email", has become the primary mode of communication all over the world. Email serves as an integral part of our everyday life. Be it work, or to be in touch with distant family members, it has become the most cheapest means of communication because of its nearly zero cost in terms of money.**

The aim of this project is to develop a smartphone app that is able to encrypt any file to be used attachments using an email app and also decrypt it at the receiver's end, so that middle man cannot view them even if they tap the mail in the middle.

***Keywords*— Encryption, Decryption, Cryptography, Smartphone, SMTP, E-mail, Spoofing, Phising**

I. Introduction

Smartphones have become very easily accessible to a common man. Gone are the days of Blackberry and business class Nokia phones which were used by business tycoons. Be it the revolution brought by Apple in terms of its classy smartphones or Google's Android mobile operating system, it has lowered the bar and changed the concept that smartphones are only for business people.

But every thing in this world has two aspects. With innovation, there comes threat. The ability to communicate on the go, may also have a price to pay. We can call them Hackers, Email Spammers or Internet Frauders, they are always a threat to our communication channel. There is no assurance that the email you send to your colleague or family member is reached to them without being read or altered by a third party in between the communication line.

A. Need of Encryption of Email contents and attachments

Email Spoofing is one of the most common techniques used by Internet Frauders or Phishers. Email Spoofing basically means impersonating as someone else to send a message to you, and modifying the email headers in such a way that it may appear to be from a legitimate source to you. Email spoofers generally change the "replyto" field in the email header, so that when you reply back to their email, instead of going back to your source email it goes to the spoofer's address in the "replyto" field. There are many attempts made to reduce email spoofing, but none of them provide us 100% positive results.

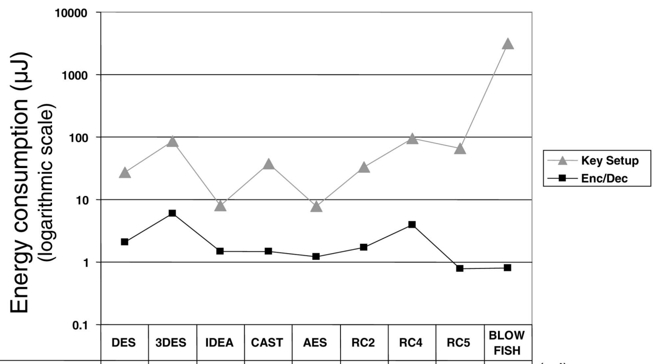
Encrypting a message and its content is the least one can do before sending a confidential message to someone. Not only email content, but the attachments that we send are more valuable to us. Encryping the message is less important when you send the most valuable asset in the attachments. There are many encryption techniques that can be used in email applications in desktop or web. But, when we use the same in a smartphone, we have to consider its processing speed as well as battery backup. Our aim is to achieve a winwin condition between both decent encryption and decryption standard and efficient processing and battery status.

B. Literature Review

There has been various attempts in the past to make users able to send Encrypted messages using email. There has not been any improvement in the method and techniques used to do so. Email clients attempt to encrypt the email contents, but fail to encrypt the attachments. That can pose risk for users of the system.

There has been lot of efforts using different algorithms for encrypting data. The most common and readily available algorithm now-a-days is Advanced Encryption Standard (AES). It is also the most secure encryption standard known to withstand any kind of brute-force attacks as impossible.

AES uses 128, 196 and 256 key lengths for encryption and decryption of data. AES is the perfect combination of Security and Power Efficiency. In mobile devices, we have limited resources like RAM, Processing speed and Battery backup, thus using AES is the best choice.



	DES	3DES	IDEA	CAST	AES	RC2	RC4	RC5	BLOW FISH	
Key Setup	27.53	87.04	7.96	37.63	7.87	32.94	95.97	66.54	3166.3	(µJ)
Enc/Dec	2.08	6.04	1.47	1.47	1.21	1.73	3.93	0.79	0.81	(µJ/byte)

Fig. 1 Energy consumption data for various symmetric ciphers.

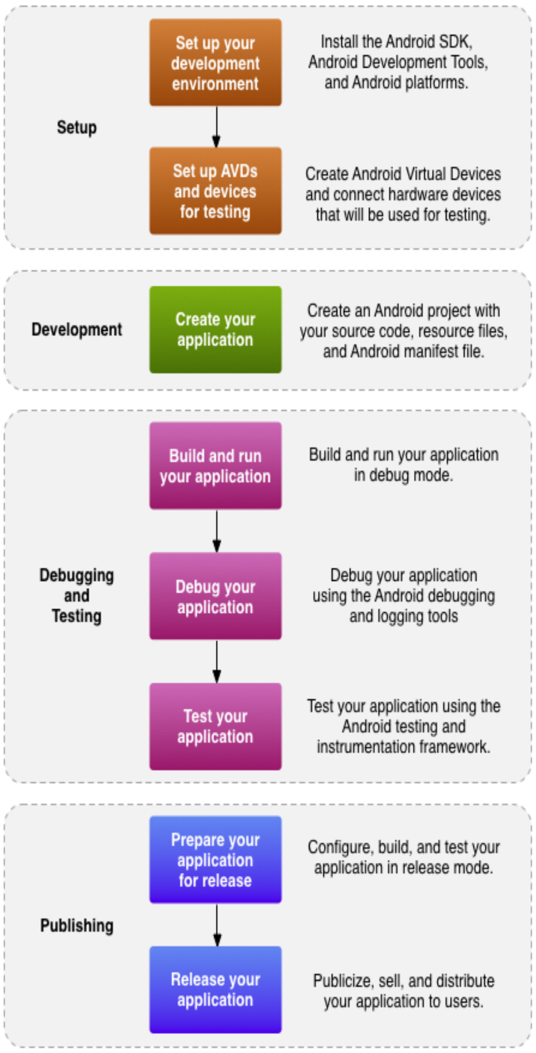
II. Background

The project was developed in native Android. This helped me to use the existing Android UI APIs for the interface and fields validation. Besides the use of existing APIs, I had to use third-party libraries (in my case, my internship provider's) to accomplish major tasks like Encryption/Decryption and HTTP Digest Authentication.

The product is targetted for minimum Android API version 8 (i.e. Android v2.2) and above. It has been tested in Android Virtual Device for the following versions:

- Froyo (Android v2.2)
- Gingerbread (Android v2.3.3)
- Icecream Sandwich (Android v4.0.3 & v4.1.2)
- Jellybean (Android v4.2)

The development process for an Android application[7] can be depicted by the figure below:



Setup
Set up your development environment
Install the Android SDK, Android Development Tools, and Android platforms.
Set up AVDs and devices for testing
Create Android Virtual Devices and connect hardware devices that will be used for testing.
Development
Create your application
Create an Android project with your source code, resource files, and Android manifest file.
Debugging and Testing
Build and run your application
Build and run your application in debug mode.
Debug your application
Debug your application using the Android debugging and logging tools
Test your application
Test your application using the Android testing and instrumentation framework.
Publishing
Prepare your application for release
Configure, build, and test your application in release mode.
Release your application
Publicize, sell, and distribute your application to users.

Fig. 2 The Development Process of Android applications[7]

A. *Android Software Development Kit*

The Android SDK[7] provides the API libraries and developer tools necessary to build, test, and debug apps for Android. The Android software development kit (SDK) includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials. Currently supported development platforms include computers running Linux (any modern desktop Linux distribution), Mac OS X 10.5.8 or later, Windows XP or later.

The officially supported integrated development environment (IDE) is Eclipse using the Android Development Tools (ADT) Plugin, though IntelliJ IDEA IDE (all editions) fully supports Android development out of the box, and NetBeans IDE also supports Android development via a plugin. Additionally, developers may use any text editor to edit Java and XML files, then use command line tools (Java Development Kit and Apache Ant are required) to create, build and debug Android applications as well as control attached Android devices (e.g., triggering a reboot, installing software package(s) remotely).

Enhancements to Android's SDK go hand in hand with the overall Android platform development. The SDK also supports older versions of the Android platform in case developers wish to target their applications at older devices. Development tools are downloadable components, so after one has downloaded the latest version and platform, older platforms and tools can also be downloaded for compatibility testing.

Android applications are packaged in .apk format and stored under /data/app folder on the Android OS (the folder is accessible only to the root user for security reasons). APK package contains .dex files (compiled byte code files called Dalvik executables), resource files, etc.

B. *Loment Cryptography Library*

The Loment Cryptography Library is a Java Library (JAR) that comprises of all the cryptography methods used for encryption and decryption works of Loment apps. The library is itself a patented library which has its base upon the open source Bouncy Castle[9] library used in Java.

The Loment Cryptography Library has been provided to me by Loment Inc. with just its implementation methods exposed. The code for the library has been level 3 obfuscated hence only its implementation can be seen, not the code. There are 2 main classes that I have used to implement the encryption and decryption methods in the app.

1) EncoderDecoder Class: This class gives the methods ***Encode*** and ***Decode***. The Encode method is responsible for generating the encoding string from the password provided by the user, that will be passed to the next method. The Decode method does the opposite, it takes input from decryption method.

The implementation is:

String EncoderDecoder.encode(byte[] arg0)
byte[] EncoderDecoder.decode(String arg0)

2) Crypter Class: The Crypter class does the actual encryption and decryption. It provides 2 methods namely ***encryptNew*** and ***decryptNew***. These methods are responsible for encrypting and decrypting stream data when provided with the input from the encode and decode methods.

The implementation is:

Crypter.encryptNew(Key arg0, byte[] arg1)
Crypter.decryptNew(Key arg0, byte[] arg1, boolean arg2)

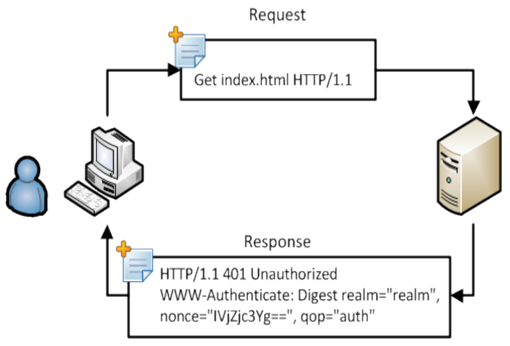
C. Loment Sthithi APIs

The Loment Sthithi APIs are the gateway to their API server calls. The APIs gives the developers the access to lot of functions that are needed in any Loment application. Loment Inc. uses a common Loment ID for all of their apps. Hence, when a user is registered for any one of Loment apps, he can use other products using the same Loment ID. The request url for all Loment API calls is ***https://api-sthithi.loment.net/*** which is password protected and uses HTTP Digest for its access. It is accessible to only the developers or any other client with proper ***USERNAME*** and ***KEY***.

The major API group include:

- Register User
- Authenticate User
- Subscription Manager
- Payment Manager
- User Information
- Device Information

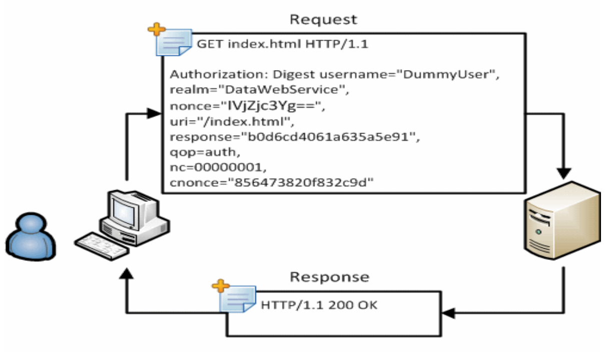
1) DigestAuthHandler Class: The HTTP Digest Authentication system works on two passes. In the first pass the client sends the request to the server with username and password encorporated in it. The server responds to the client with a challenge along with response code 401 (unauthorized). The challenge contains a realm, nonce, cnonce and qop. In the second pass, the client now encorporates all the fields as presented by server and re-sends the request.



```
Get index.html HTTP/1.1
```

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest realm="realm",
nonce="IVjZjc3Yg==", qop="auth"
```

Fig.3 Pass 1 of HTTP Digest Authentication



```
GET index.html HTTP/1.1

Authorization: Digest username="DummyUser",
realm="DataWebService",
nonce="IVjZjc3Yg==",
uri="/index.html",
response="b0d6cd4061a635a5e91",
qop=auth,
nc=00000001,
cnonce="856473820f832c9d"
```

```
HTTP/1.1 200 OK
```

Fig. 4 Pass 2 of HTTP Digest Authentication

2) DataConnection Class: The DataConnection class is reponsible for handling user requests. The user can request by using 2 methods namely GET and POST. The GET method requests data from a specified resource while the POST method submits data to be processed to a specified resource.

The DataConnection class exposes 2 main method calls namely sendGetRequest and sendPostRequest.

The implementation is:

String sendGetRequest(String request_url, String data)
String sendPostRequest(String request_url, byte[] data)

The response of DataConnection class is a JSON String which contains the responseCode, responseMessage and other fields that has been requested during the request.

3) Core API methods used: Though there are a number of methods that can be accessed though the Sthithi APIs, it is not necessary that every product uses the methods. I had used the following methods during my project implementation.

- *User_add*

  This method allows us to register a new user to Loment Inc. Once registered, users can use this account to use any of Loment products.

  URI: /user/register/
  TYPE: Post Data
  PARAMETERS: none

- *User_authenticate*

This method allows us to authenticate the login credentials. It handles the process of hashing the password in the server side and does the authentication.

URI: /user/:username/authenticate/
TYPE: Post Data
PARAMETERS: <:username>

- *User_account*

This method allows us to get or update the user information. It also allows us to change out Loment password.

URI: /user/:username/account
TYPE: Get Data
PARAMETERS: <:username>

III. IMPLEMENTATION

A. Android Activity Lifecycle[6]

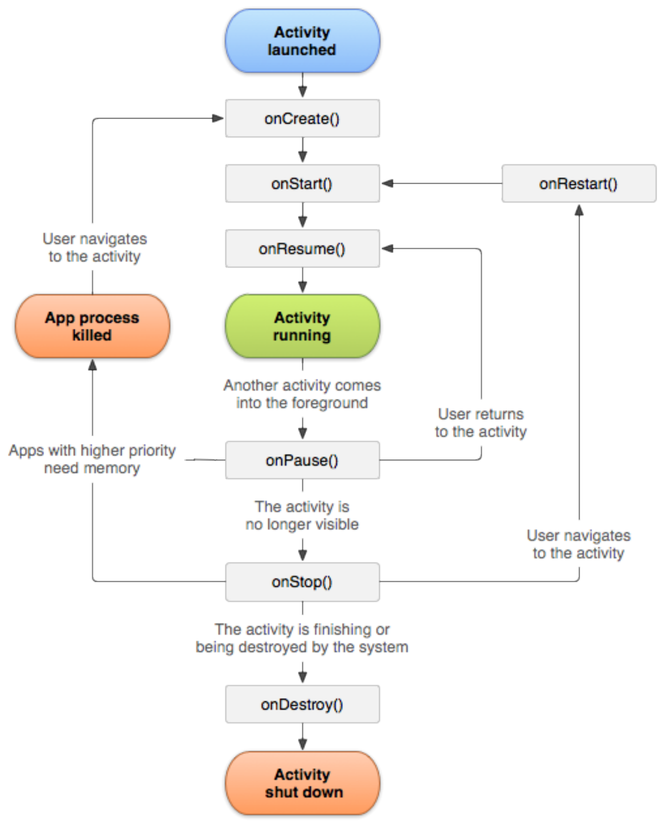
The Android application contains a series of Activities. Activities in the system are managed as an activity stack. When a new activity is started, it is placed on the top of the stack and becomes the running activity -- the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits.

An activity has essentially four states:

- If an activity in the foreground of the screen (at the top of the stack), it is active or running.
- If an activity has lost focus but is still visible (that is, a new non-full-sized or transparent activity has focus on top of your activity), it is paused. A paused activity is completely alive (it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations.
- If an activity is completely obscured by another activity, it is stopped. It still retains all state and member information, however, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere.
- If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.

There are three key loops within an activity:

- The **entire lifetime** of an activity happens between the first call to ***onCreate(Bundle)*** through to a single final call to ***onDestroy()***. An activity will do all setup of "global" state in ***onCreate()***, and release all remaining resources in ***onDestroy()***.


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

- The **visible lifetime** of an activity happens between a call to ***onStart()*** until a corresponding call to ***onStop()***. During this time the user can see the activity on-screen, though it may not be in the foreground and interacting with the user. Between these two methods you can maintain resources that are needed to show the activity to the user.
- The **foreground lifetime** of an activity happens between a call to ***onResume()*** until a corresponding call to ***onPause()***. During this time the activity is in front of all other activities and interacting with the user. An activity can frequently go between the resumed and paused states.

Fig. 5 Android activity lifecycle

The implementation is:

```
public class Activity extends ApplicationContext {

    protected void onCreate(Bundle savedInstanceState);

    protected void onStart();

    protected void onRestart();

    protected void onResume();

    protected void onPause();

    protected void onStop();

    protected void onDestroy();

}
```

TABLE I
DESCRIPTION OF ACTIVITY STATES

Method	Description	Next Call
onCreate()	called when the activity is first created.	onStart()
onRestart()	called after your activity has been stopped, prior to it being started again.	onStart()
onStart()	called when the activity is becoming visible to the user.	onResume()/ onStop()
onResume()	called when the activity will start interacting with the user.	onPause()
onPause()	called when the system is about to start resuming a previous activity.	onResume()/ onStop()
onStop()	called when the activity is no longer visible to the user, because another activity has been resumed and is covering this one.	onRestart()/ onDestroy()
OnDestroy()	the final call before the activity is destroyed.	nothing

B. HTTP Digest Authentication

The Digest mechanism[5] allows a client to authenticate itself by presenting credentials consisting of an MD5 digest, transmitted in a request message. It is based on the principle that the client and server are in possession of a shared secret, a password string. The advantage of this method is that the client password is only used in calculating the digest, so it remains safe from network exposure.

The Digest mechanism is a challenge/response protocol in which the client presents its credentials in response to a challenge from the server, which consists of an opaque data string called a "nonce". This nonce serves as additional input to the MD5 function, and allows the server to influence the digest value in a way not controlled by the client.

In order to authenticate the client, the server simply compares the digest value received from the client with the value it has computed internally. If the values match, the client must be in possession of the same nonce and password as the server, so the client is authenticated. The same technique is used in the mutual authentication scenario, where the server authenticates itself to the client by presenting a digest as credentials in response to a challenge from the client. In this case also the challenge consists of a client-produced nonce to be used as input to the digest function, allowing the client to influence the digest value in a way not controlled by the server.

1) *Header Entry Elements:*

- "***Challenge***" Element
- "***ClientAuth***" Element
- "***NextChallenge***" Element
- "***InitChallenge***" Element

The pictorial representation of working mechanism of HTTP Digest Authentication system is shown in Fig. 3 and Fig. 4.

C. Cryptography Library

Loment Crytography Library is a patented library[10] which is patented under **US 20120250594 A1**.

The cryptography library uses a series of steps to encrypt and decrypt any string. It used AES-256 along with Salt and IV with 5 block padding size for its purposes. The actual mechanism of generating the Salt, IV and the Message Key from the user provided password is a secret because it is a commercial product.

1) Steps in Encryption: Each round of the encryption process requires a series of steps to alter the state array. These steps involve four types of operations called:

i. SubBytes

This operation is a simple substitution that converts every byte into a different value. AES defines a table of 256 values for the substitution. We work through the 16 bytes of the state array, use each byte as an index into the 256-byte substitution table, and replace the byte with the value from the substitution table. Because all possible 256 byte values are present in the table, we end up with a totally new result in the state array, which can be restored to its original contents using an inverse substitution table. The contents of the substitution table are not arbitrary; the entries are computed using a mathematical formula but most implementations will simply have the substitution table stored in memory as part of the design.

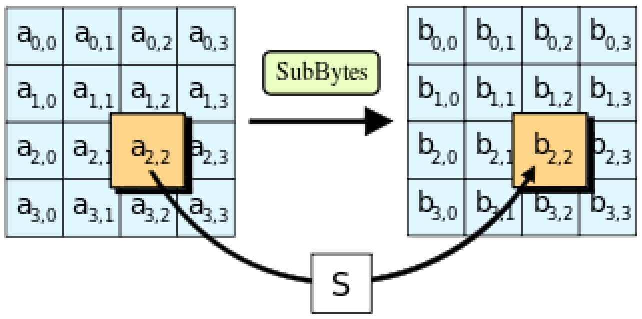

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
S

Fig. 6 AES SubBytes Operation

The implementation is:

```
public byte[ ][ ] subBytes(byte[][] state)
{

  for (int i=0;i<4;i++)

  {

    for (int j=0;j<4;j++)

    {

      int row = getFirstFourBits(state[i][j]);

      int column = getSecondFourBit(state[i][j]);

      state[i][j] = sBoxSubstitution(row,column);

    }

  }
return state;

}
```

ii. ShiftRows

The ShiftRows step is performed on the rows of the state matrix. It cyclically shifts the bytes in each row by a certain offset. The first row remains unchanged. Each byte of the second row is shifted one position to the left. Similarly, the third and fourth rows are shifted by two positions and three positions respectively.

The importance of this step is to make columns not linear independent If so, AES becomes four independent block ciphers.

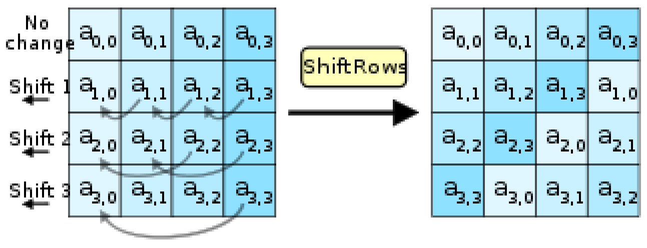


Fig.7 AES ShiftRows Operation

The implementation is:

```
shiftRows(byte state[][])

{

   for(int i=0;i<4;i++)

   {

         cyclicLeftShift(i);

    }

}
```

iii. MixColumns

This operation is the most difficult, both to explain and perform. Each column of the state array is processed separately to produce a new column. The new column replaces the old one. The processing involves a matrix multiplication.

It takes each column of the state array and replaces it with a new column computed by the matrix multiplication.

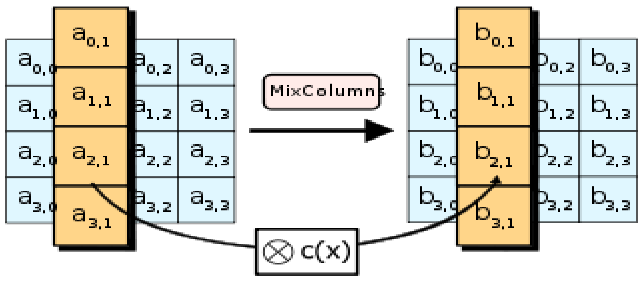


Fig. 8 AES MixColumns Operation

The implementation is:

```
public byte[ ][ ] mixColumns(byte[ ][ ] state)

{

  for (int c=0;c<4;c++)

  {

      state [c]=matrixMultiplication(state[c], polynomial);

  }
```

```
return state;

}
```

iv. XorRoundKey

After the MixColumns operation, the XorRoundKey is very simple indeed and hardly needs its own name. This operation simply takes the existing state array, XORs the value of the appropriate round key, and replaces the state array with the result. It is done once before the rounds start and then once per round, using each of the round keys in turn.

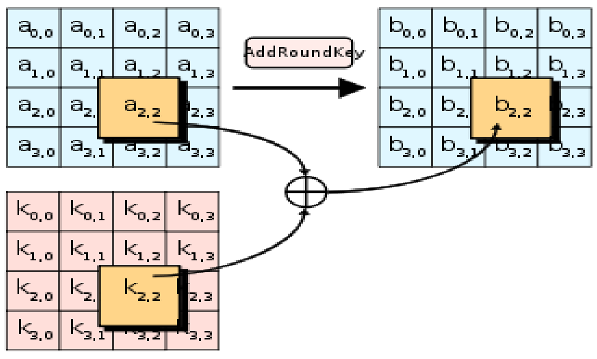


Fig. 10 AES AddRoundKeys Operation

The implementation is:

```
public byte[ ][ ] addRoundKey(byte[ ][ ] state,byte[ ][ ]

roundkey)
{
  for (int i=0;i<4;i++)
  {
    for (int j=0;j<4;j++)

    {
      state [i][j]=doExclusiveOR(state[i][j],
 roundkey[i][j]);
    }

}
return state;

}
```

2) Steps in Decryption: Decryption involves reversing all the steps taken in encryption using inverse functions:

- ***InvSubBytes***
- ***InvShiftRows***
- ***InvMixColumns***

XorRoundKey doesn't need an inverse function because XORing twice takes you back to the original value. InvSubBytes works the same way as SubBytes but uses a different table that returns the original value. InvShiftRows involves rotating left instead of right and InvMixColumns uses a different constant matrix to multiply the columns.

3) Challenges in File Encryption and Decryption: File encryption and decryption is very much different than encryption and decryption of string. We need to consider the file padding (null bytes) to be added into the block while encryption and needs to be removed while decryption.

Files can be converted to File Input Stream for reading and writing. While encryption, we read the file block by block. We used the block size of 2736 bytes for our purpose. While writing the encrypted file, in the last pass, we add padding to complete the block size.

Now, when the encrypted file is shared, we cannot determine, how much padding was added in the file. Moreover, removing all the trailing padding also creates problems because there already may be padding in the original file itself. Hence, the last block coulnot be written reliably.

As an alternative solution, we used the file size as a measure to read and write. We would pass the actual file size in each file's header while encryption and write only that number of bytes while decryption. This way we overcame the padding problem.

IV. Conclusion

The need of file encryption is into its greatest level because sharing data between 2 parties always needs privacy and security. Using email as the medium of communication on one hand gives us the power to communicate instantly on the go while on the other hand increases the risk of our message being sniffed and tampered while its transmission though different gateways.

Developing an android app that could encrypt and share a file while it could also decrypt and open the file for viewing using any email client was the main motive of this app. Besides this, it also works a stand-alone app for general file encrypting decrypting process or for hiding a file by encrypting its content using a custom extension.

Acknowledgment

I express my deep gratitude and regards to Dr. M. Venkata Chalapathi (CEO Loment Inc) for giving me the opportunity to complete my internship and his guidance, help, encouragement throughout the course of this thesis work at Loment Inc, Hyderabad.

I would like to express my thanks with gratitude to Dr. J.V.R Murthy, Internal Guide and Professor, Department of Computer Science & Engineering for his encouragement and valuable guidance in bringing shape to this dissertation.

I am grateful to Smt. L. Sumalatha, Head of the Department, Department of Computer Science & Engineering, University College of Engineering, JNTU Kakinada for her encouragement and motivation.

I am thankful to all the Professors and Faculty Members in the department for their teachings and academic support and thanks to Technical Staff and Non-teaching staff in the department for their support.

Last, but not the least, I would like to express my heartfelt thanks to Mr. Bharath Kishore Aluri, Mr. Madhur Vyas and all other senior members at Loment Inc. for their support and help during the completion of the project.

References

[1] Lee, H., Shin, DH., and Jung, HC., *Implementations of Block Cipher SEED on Smartphone Operating Systems*, 2The Fifth International Conference on Emerging Security Information, Systems and Technologies, 2011.

[2] L. Shurui, L. Jie, Z. Ru, and W. Cong, *"A Modified AES Algorithm for the Platform of Smartphone"*, 2010 International Conference on Computational Aspects of Social Networks, 2010, pp. 749-752.

[3] A. Visoiu and S. Trif, *"Open Source Security Components for Mobile Applications"* Open Source Science Journal, vol. 2, no. 2, 2010. pp. 155-166.

[4] National Institute of Standards and Technology, *Specification for the ADVANCED ENCRYPTION STANDARD*, 2001, [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[5] The Whitemesa specifications for SOAP Extensions: Basic and Digest Authtication. [Online] Available: www.whitemesa.com/soapauth.html

[6] Android Activity Lifecycle Reference. [Online]. Available: http://developer.android.com/reference/android/app/Activity.html

[7] Android Development Workflow and Software Development. [Online]. Available: http://developer.android.com/tools/workflow/index.html

[8] Miranda, P., Siekkinen, M., *TLS and Energy Consumption On a Mobile Device: A Measurement Study*, IEEE, 2011.

[9] The Legion of Bouncy Castle. [Online] Available: http://www.bouncycastle.org/java.html

[10] Loment Encryption Library Patent Claim. [Online] Available: http://www.google.com/patents/US20120250594?dq=inassignee:loment+inassignee:inc