Email Client Application with Rabbit Algorithm for Android Smart Phone

Muhammad Anwari Leksono, Rinaldi Munir
Teknik Informatika
Institut Teknologi Bandung
Bandung, Indonesia
m_anwari@students.itb.ac.id, rinaldi@informatika.org

***Abstract*—this paper explains the implementation of Rabbit algorithm in email application to secure email content on Android smart phone. This particular email client uses Rabbit algorithm to encrypt and decrypt confidential email's content. This application can be used to create, edit, send, retrieve, and read a simple email. Email client is built for Android smart phone by using Java. The experiments prove that by using Google Mail service this application can be used as simple email client with secured content feature.**

Rabbit algorithm; email; email client; Android smart phone;

I. INTRODUCTION

Email as one of many important communication facilities sometimes contains confidential content that needs to be secured. One of many methods to ensure contents secrecy is by using encryption. The proposed algorithm is Rabbit which is a symmetric algorithm and symmetric algorithm is faster than asymmetric ones [4]. Email service nowadays is commonly accessed by using smart phone. One of the most popular smart phone operating system is Android. In order to maintain secrecy of email sent from Android smart phone, encryption and decryption process must be planted in an email client for Android smart phone.

In this paper we present a client application for encrypting email in Android smart phone using Rabbit Algorithm. This application is designed to be able to encrypt email content, send emails, receive emails, read email, and decrypt email content.

There are many things to be considered in the making of this application such as: (1) integration of Rabbit algorithm with email client, (2) method to create encrypted content and send it by email, (3) method to retrieve email from email service provider, and (4) read an email completely including the encrypted content.

II. LITERATURES STUDY

A. Rabbit Algorithm

Rabbit algorithm was created by Fast Software Encryption in 2003 with Martin Boesgaard, Mette Vesterager, Jesper Christiansen, and Ove Scavenius. Rabbit uses 128 bit to encrypt or decrypt 128 bit of plaintext or cipher text. Rabbit has 17 variables: eight inner states, eight counters, and one carry. There are three schemes of Rabbit: (1) key setup scheme, (2) next state function scheme, and (3) extraction scheme. Key setup scheme is used for initiating value of variables. Next state function is used for updating variables value. Extraction scheme is used to create a 128 bit pseudorandom string. Cipher text is created by using XOR operation between 128 bit plaintext with 128 bit pseudorandom string [1].

B. Email

Email is method to exchange message by using internet connection. Email nowadays uses MIME as its format. Email transaction uses store-and-forward method [5]. This method enables people to send an email and received email later without having themselves online at the time an email arrives.

There are protocols defined for email transaction such as POP3 and IMAP are defined for retrieving email from internet and SMTP is defined for email delivery. These protocols can be modified with SSL with the aim of making them more secure. IMAP protocol allows email user to manipulate their email from any devices without worrying their email will be erased after being read. POP3 does the opposites. POP3 allows user to download their email from internet but this protocol will erased their email so the same email will not be downloaded next time [3].

C. Android Operating System

Android is developed by Google which is currently running in smart phones. Android was built on Linux platform and developed by using Java. Android has already been optimized in order to be used in mobile environment. Android is nowadays known as a popular smart phone operating system [7].

Android is an open source operating system so there are many developers who pay attention, explore, and fix Android security risk. In other words, Android is secured [2]. Developing third-party application in Android can be done by using Java, C, or C++. Using C or C++ does not improve

978-1-4673-4550-7/12/$31.00 ©2012 IEEE

application performance but it can be assured that using these languages can make source code more complicated [8].

III. Proposed Analysis and Solution Designs

There are some details to be thought and design for making this email client including development language, design of encryption and decryption process, email delivery and retrieval, and content making and reading.

A. Development Language

In order to develop an email client for Android, there are three languages that can be chosen. Java is seen to be the most appropriate language since Java has many free libraries that support and simplify the making of application. For this matter Java has JavaMail library that focuses in email.

B. Rabbit Design

Rabbit encrypts 128 bit data or plaintext with 128 bit key. If the plaintext is more than 128 bit length then plaintext must be divided into several parts. These parts can be written as B_1, B_2, .., B_{n-1}, and B_n. If the last part of plaintext, B_n, is less than 128 bit length then this part will be added with empty spaces until this part has 128 bit length.

The process of Rabbit algorithm can be explained shortly as follows: all variables are set to their initial value based on key setup scheme; all variables then are updated by using next state function scheme; extraction scheme creates 128 bits pseudorandom string, P_n, from variables values; this 128 bits pseudorandom string then is paired with B_n for XOR operation creating C_n as the cipher text. These steps imply that next state function scheme and extraction scheme will be called for $n + 4$ and n times respectively because next state function is also called in key setup scheme for four times.

Cipher text has 128 bits or 16 bytes of length. If cipher text is presented as set of characters then there will be 16 characters. The problem with this is if 8 bit of cipher text is translated into a character then there is no guarantee that corresponding character is readable, i.e. “00000000” means “null” so this will not be visible as character and “00001010” means “new line” and this also will not be visible. The proposed solution for this matter is by representing cipher text as a set of hexadecimal character. Each 8 bits will be translated into two hexadecimal characters. By doing this, “0000000” will be written as “00” and “00001010” will become “0A”. Since 8 bits or one byte will be translated into two hexadecimal characters, 128 bits cipher text will be translated into 32 hexadecimal characters text. Therefore, cipher text’s length will be two times longer than plaintext. Encryption flowchart can be seen at Figure 1.

Decryption is designed to be slightly different. Since every character in plaintexts is transformed into two hexadecimal characters in cipher texts, cipher texts must be read per two characters. If cipher text is more than 32 characters then cipher text must be divided into parts. Each part contains only 32 hexadecimal characters. Each part is then transformed into 128 bits cipher text. The rest of decrypting process is the same as encryption process. Decryption steps can be seen at Figure 2.

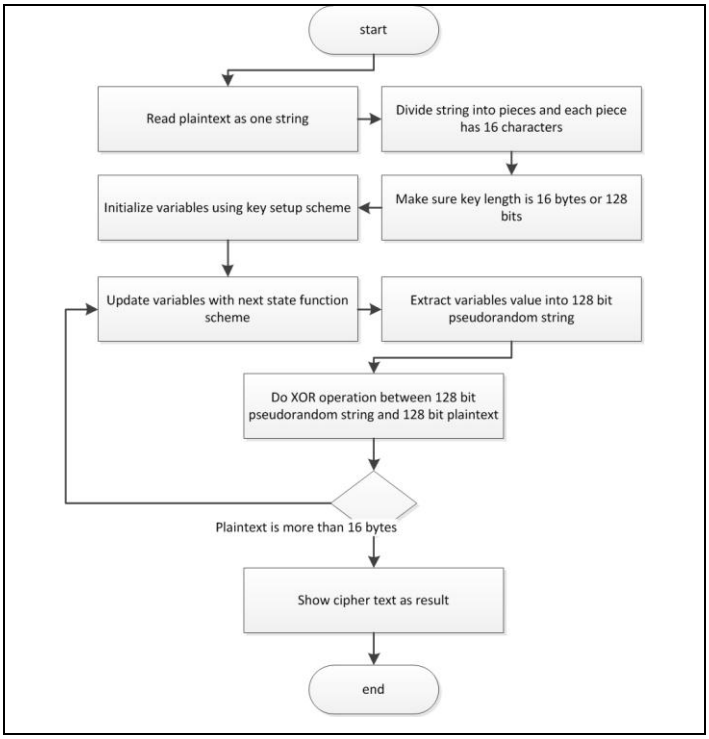


Figure 1. Encryption flowchart

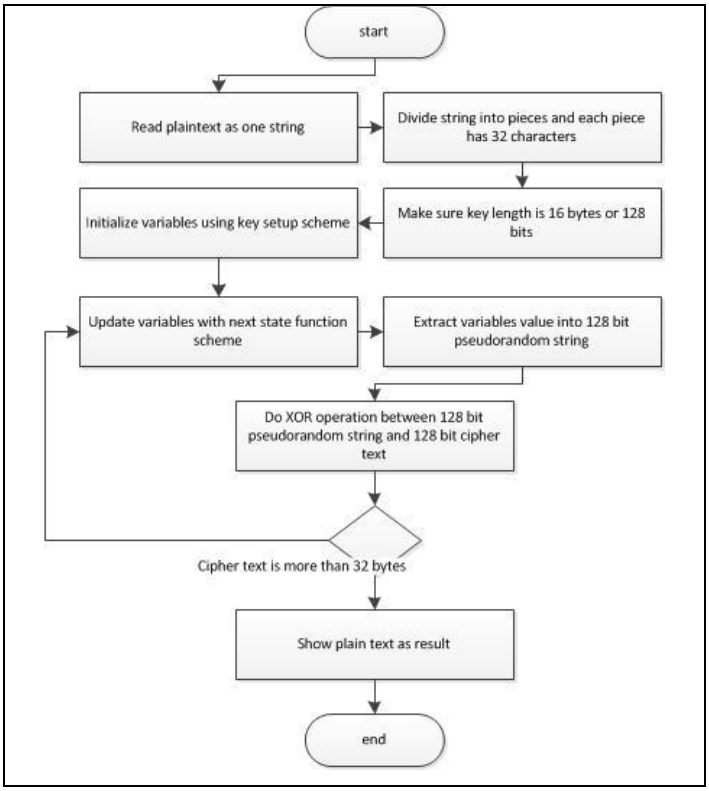


Figure 2. Decryption flowshart

C. Content Making and Email Delivery

Content making means creating cipher text that will be used as email content itself. In creating email content session, user may insert either plaintext or cipher text or both. In order to separate cipher text and plaintext, a pair of label is used. The labels are "<enc>" and "</enc>". In other words, all words between these two labels are called cipher text. Steps to create content and email can be seen at Figure 3.

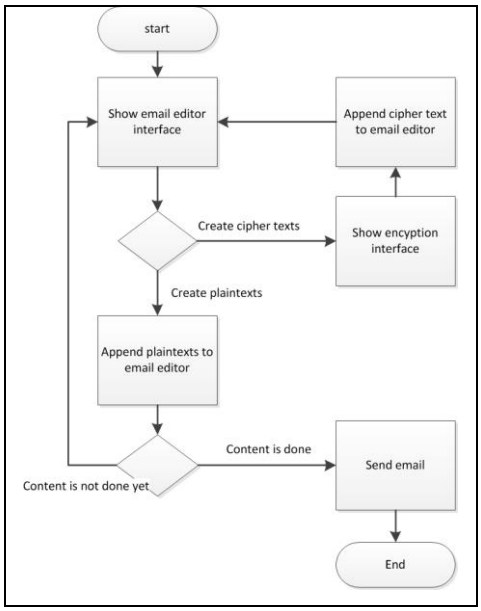


Figure 3. Content-making and email delivery flowchart

Any other desired content which is not encrypted shall be placed outside the labels. Plain content – not encrypted content – must be placed before "<enc>" or after "</enc>. Placement of plaintexts (plain contents) and cipher texts may vary, such as cipher text followed by plaintext or vice versa. Email is then sent by using SMTP. JavaMail provides SMTP with TLS and SSL support. Email content is prepared to be sent as text/plain content. This is a simple type of email content which only contains text.

D. Email Retrieval and Content Reading

Retrieving email from internet can be done either with IMAP or POP3. The chosen protocol is IMAP. IMAP is chosen because it allows emails retrieval from internet without deleting those emails in internet like what POP3 does. This implies that even emails are already read by using application once, emails can still be accessed.

Email content is composed by cipher texts and plaintexts. In order to read email completely, cipher texts and plaintexts must be divided into parts where each part contains either cipher texts or plaintexts. These parts also need to be arranged in proper order so when parts are joined, content will not lose its semantic meaning.

Each part contains TYPE which defines if the subtext is cipher text or plaintext and CONTENT which defines the subtext itself. These parts can be seen in table below. If TYPE equals to zero then CONTENT must be plain text and if TYPE equals to one then CONTENT must be cipher text. After decrypting, all parts are joined into one content. These steps can be seen at Figure 4.

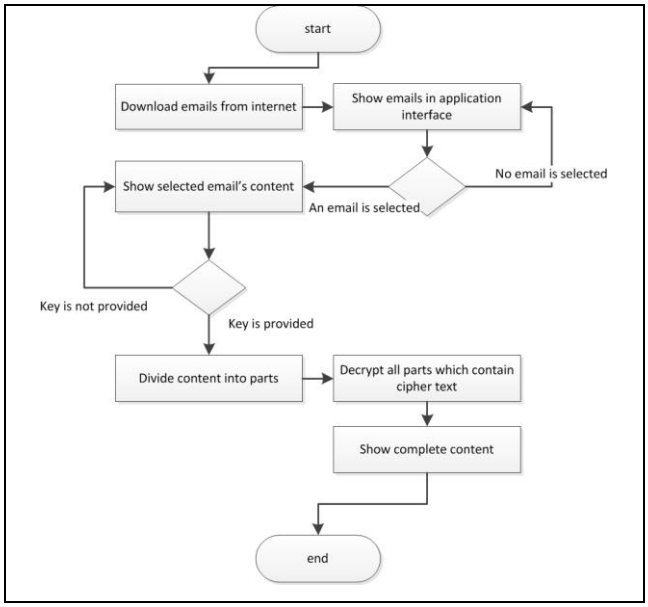


Figure 4. Email retrieval and content reading flowchart

IV. Implementation and Experiments

Implementation is done at personal computer. Application is developed with Eclipse 3.6 with Android Plugin, Android SDK, JDK 1.6, and JRE 16. Application is tested in Sony Xperia Ray with Android 4.0.4 operating system.

Experiments are done in order to know if encryption-decryption, email exchange, and content creation do their function well. Encryption-decryption function is tested by using some texts to be encrypted and the cipher text is decrypted. If test data is the same as plaintext from decryption then encryption-decryption function does its function well.

TABLE I. Table Test Data Encryption Result

No	Ciphertext
1	34cf9bd6e25efd9a80b96dc0b44e1609
2	02cad6faae5a8fabd4ff21c9cb451b14 e3737b3c4cf07428cbeed24fcc4cd5a0
3	4d98c48eb8489dcdcbe529c0b44e1609
4	34cf9bd6e2528a819db16899b4074509 cb6b501513ba2a7386e2cd51c403d5b2 31b9022cf81e8c187d8a8ba790bfd0bf 4dbcf1c5cae2fe23914b44a8779f9504 1417338e78a3a9dbba5d3d3fb6d98740 68c71aadbc1d22e461b13bc0f1944865 4eae5ff6a8329fbab99379672d7ec9ae 29a59eb4fa88159f49e59b172161da42

Content creation function is tested along with encryption test. Test data which are used to be encrypted and inserted into email contain are "Hello World", "~`!@#$%^&*()_+-

={}[]:'<>?,./|'", "1234567890", "Hello, today is September 11th 2012. Today is a big day. One of my friends says, 'Hey, this new day should be awesome, right?'". These four test data are defined representing words, punctuations, numbers, and their combination. These four test data is then encrypted by using "helloworld" as the key. The cipher text from words, punctuations, numbers, and combinations respectively can be shown at TABLE I. Application interfaces for all test data with "helloworld' as the key are shown by Figure 5.

(a) (b) (c) (d)

Figure 5. Application interfaces to encrypt all test data.

These four cipher texts are then used as part of email main content. Cipher texts are inserted into main content by placing them between two labels, "<enc>" and "</enc>".To check that encryption is done properly, the cipher texts must be decrypted. Decryption result can be seen at TABLE II. Since decryption results are as readable as the original plain texts then it is safe to assume that encryption-decryption function works properly.

TABLE II. TABLE TEST DATA DECRYPTION RESULT

No	Decryption Result
1	Hello World
2	~`!@#$%^&*()_+-={}[]:'<>?,./|'
3	1234567890
4	Hello, today is September 11th 2012. Today is a big day. One of my friends says, 'Hey, this new day should be awesome, right?'

Email content is created by combining plaintexts and cipher texts into one writing. There are many arrangements that can be formed by combining cipher texts with plaintexts. Examples of content can be seen at Figure 6. and Figure 7. below.

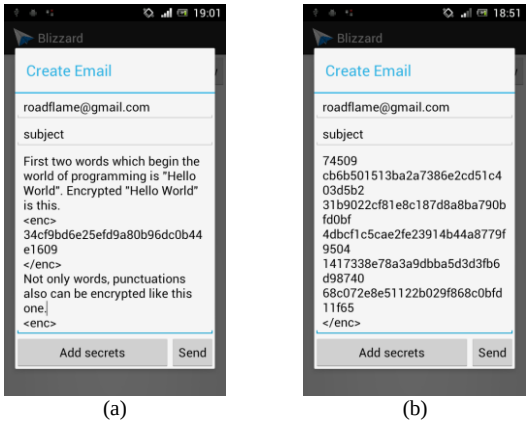
```
First two words which begin the world of
programming is "Hello World". Encrypted "Hello
World" is this.
<enc>
34cf9bd6e25efd9a80b96dc0b44e1609
</enc>
Not only words, punctuations also can be
encrypted like this one.
<enc>
02cad6faae5a8fabd4ff21c9cb451b14
</enc>
And, that is it. Farewell for now.
```

Figure 6. First example content

```
<enc>
4d98c48eb8489dcdcbe529c0b44e1609
<enc>
<enc>
cb6b501513ba2a7386e2cd51c403d5b2
31b9022cf81e8c187d8a8ba790bfd0bf
4dbcf1c5cae2fe23914b44a8779f9504
1417338e78a3a9dbba5d3d3fb6d98740
68c71aadbc1d22e461b13bc0f1944865
4eae5ff6a8329fbab99379672d7ec9ae
29a59eb4fa88159f49e59b172161da42
</enc>
First cipher text comes from numbers
encryption.
Second cipher text comes from paragraph
encryption.
```

Figure 7. Second example content

Application interfaces for creating these two examples content can be seen at Figure 8.



(a) (b)

Figure 8. Application interfaces to send (a) first content and (b) second content.

There are many ways to place plaintext and cipher text other than two above. These two example paragraphs above are used for two email content. Emails are then sent to two different addresses which are roadflame@yahoo.com and

roadflame@gmail.com from address flameroad64@gmail.com. To check if emails are sent properly, these two addresses are accessed by using web browser. The result of delivery experiment is that these two emails arrive at desired addresses and application succeeds to send emails properly. These emails with these certain contents are then forwarded to the sender itself at flameroad64@gmail.com. After forwarding emails, application by using sender account can read these example emails.

Emails must be readable completely including cipher texts inserted in email's content. Complete email content can be served after decrypting cipher texts, reading plaintexts, and joining then together in the right order. These complete readable contents can be seen at Figure 9. and Figure 10. Application interfaces for each reading can be seen at Figure 11. (a) and Figure 11. (b).

```
First two words which begin the world
programming is "Hello World". Encrypted "Hello
World" is this.
--
Hello World
--
Not only words, punctuations are also can be
encrypted like this one.
--
~`!@#$%^&*()_+-={}[]:'<>?,./|'
--
And, that is it. Farewell for now.
```

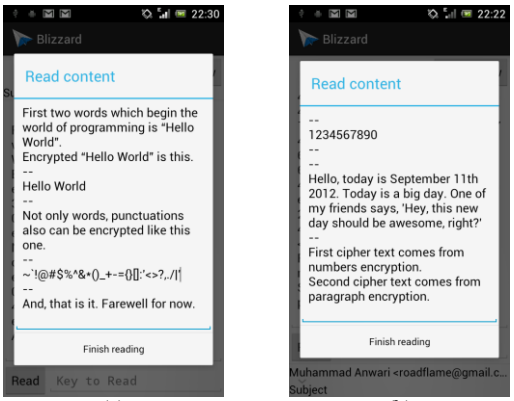
Figure 9. First example content reading

```
--
1234567890
--
--
Hello, today is September 11th 2012. Today is a
big day. One of my friends says, 'Hey, this new
day should be awesome, right?'
--
First cipher text comes from numbers
encryption.
Second cipher text comes from paragraph
encryption.
```

Figure 10. Second example content reading

Content can be read completely including the cipher texts. Decrypted cipher texts are decrypted into readable plaintexts. Since plaintexts from decryption are as good as original plaintexts, reading-decrypting function for application is proven to work properly.

Email content security has to be proven. This is done by eaves dropping the network related to email delivery. Eaves dropping is necessary in order to know whether the email content is still secured even in network or not. Expected result is that cipher texts which are inserted in email content are still unreadable but plaintexts may be readable.



(a) (b)

Figure 11. Application interfaces in reading (a) first example content, and (b) second example content

To intercept email transmission in local area network, tool which is used is WireShark. Application is run by using emulator from PC. WireShark captures all data frames sent from PC related to application's email transmission. JavaMail provides a feature which allows user to debug email transmission related to the email protocols. This feature is called mail-debug.

With this feature, before application sends an email to the nearest network, mail-debug captures this email and reads the content. Content is then shown by PC. Mail-debug captures all messages that come and go from application. The expected result is that email content will be captured and able to be read plainly.

V. Experiment Results Analysis

Cipher text is always larger than plaintext because every character in plaintext is transformed into two hexadecimal characters in cipher text. Decryption results are not exactly the same as original plaintexts. The difference is located at the end of plaintexts from decryption. Results of decryption may have spaces at the end of the text. These spaces are added to make the length of plaintexts is dividable by 16.

Labels are added to every cipher texts from one Rabbit session. Rabbit session starts with one key setup for one plaintext in any length. This means that if content contains two pairs of label then user must have used two Rabbit sessions.

Emails are sent by using SMTP protocol. The only email service which is working for this application is Google Mail. Google Mail does not allow email transmission by using unsecure protocol. That is why when using JavaMail, SMTP must be followed by enabling TLS protocol or SSL.

IMAP protocol is used to retrieve emails. By using IMAP protocol, emails can be downloaded and accessed more than just once. Content type for emails is text/plain. This explains the reason why application is only able to send text messages.

In order to read content completely, content itself must be divided into parts based on whether it is plaintext or cipher

text. Content breaking for example one and two can be seen at TABLE III. and TABLE IV. respectively.

TABLE III. TABLE CONTENT PARTS FOR FIRST CONTENT

No	Type	Content
1	0	First two words which begin the world programming is "Hello World". Encrypted "Hello World" is this
2	1	34cf9bd6e25efd9a80b96dc0b44e1609
3	0	Not only words, punctuations are also can be encrypted like this one.
4	1	02cad6faae5a8fabd4ff21c9cb451b14
5	0	And, that is it. Farewell for now.

TABLE IV. TABLE CONTENT PARTS FOR SECOND CONTENT

No	Type	Content
1	1	4d98c48eb8489dcdcbe529c0b44e1609
2	1	cb6b501513ba2a7386e2cd51c403d5b2 31b9022cf81e8c187d8a8ba790bfd0bf 4dbcf1c5cae2fe23914b44a8779f9504 1417338e78a3a9dbba5d3d3fb6d98740 68c71aadbc1d22e461b13bc0f1944865 4eae5ff6a8329fbab99379672d7ec9ae 29a59eb4fa88159f49e59b172161da42
3	0	First cipher text comes from numbers encryption. Second cipher text comes from paragraph encryption.

Decrypting is only done to content which has type one. After decrypting is finished for all type one contents, all parts are joined into one content. Joining process is done by following the order of the parts. This order needs to be followed to maintain the content semantic meaning.

Tapping email transmission using WireShark is a failure. This is caused by JavaMail itself. JavaMail requires SMTP to be secured with TLS. By using TLS all data frames are encrypted before they go out to the nearest network. This fact means that data frames can be intercepted but they can't be read at all.

Result from using mail-debug feature is a log from application's activity. This log records all messages coming from network to application and going from application from network. Email's content is shown at this log clearly. Log shows that cipher texts are not changed.

VI. CONCLUSIONS

There are conclusions from implementation Rabbit algorithm in email client for Android smart phone.

1. Rabbit algorithm is implemented by using Java. Encryption-decryption feature is combined with email content editor.
2. Cipher texts and plaintexts must be able to be separated. This separation is done by using labels.
3. Emails are sent by using SMTP service with TLS support from JavaMail and received by using IMAP.
4. Content reading is done by dividing content into parts. Parts which contain cipher texts will be decrypted.

There are suggestions to make this email application better.

1. Another email content type should be supported.
2. Supported email services can be widened so other than Google Mail service are useable.
3. Encryption-decryption can be used for images, sounds, and other multimedia data types.

REFERENCES

[1] Boesgaard. Martin, Pedersen. Thomas, Vesterager. Mette, "The Rabbit Stream Cipher – Design and Security Analysis".

[2] Bryan. Randy, "Top 4 Advantages of Android over the iPhone". 2010 [Online]. Available: http://randybryan.com/?p=671. [Accessed 7 November 2011].

[3] Crispin. M, "Internet Message Access Protocol - Version 4rev1".

[4] KetuWare, "Symmectic vs. Asymmectic Encryption".

[5] Partige. Craig, "The Technical Development of Internet Email".

[6] Postel. J.B, "Simple Main Transfer Protocol".

[7] Titlow. J.P, "Android, the Fastest Growing Smartphone OS in Europe, Zooms Past iPhone". 2011. [Online]. Available: http://www.readwriteweb.com/archives/android_european_marketshare_beats_iphone.php. [Accessed 3 November 2011].

[8] Turner. David, "Introducing Android 1.5 NDK". 2009. [Online]. Available: http://android-developers.blogspot.com/2009/06/introducing-android-15-ndk-release-1.html. [Available 6 December 2011].