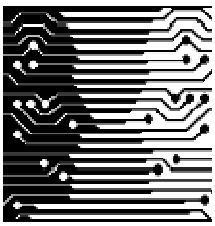
Security of Mobile Banking: Secure SMS Banking

By Ming Ki Chong

Supervised by: Alapan Arnab, Andrew Hutchison

Date of Submission: 06 November 2006



Data Network Architectures Group
Department of Computer Science
University of Cape Town
Private Bag, Rondebosch
7701, South Africa

Abstract

Mobile banking is a new scheme for bank customers to perform banking transactions. There are many different types of mobile banking. This paper describes the security shortfalls of the current mobile banking solution and it analyses the possible attacks (both passive and active) on the current SMS banking solution. A new secure SMS messaging protocol is designed to offer better security for SMS banking. This report also describes the prototype implementation of the designed protocol using Java™ technology. Testing for both the designed protocol and the prototype implementation is explained in this report. A validation of the designed protocol is provided in this report to show the design solution does enhance security for SMS banking. Finally, suggestions are made regarding the possibility for future work to improve security for the designed Secure SMS protocol.

Acknowledgement

Andrew Hutchison and *Alapan Arnab* for their guidance and supervision, organizing equipments for the project and provide the opportunity for our team to represent University of Cape Town to participate in the security forum organized by T-System.

Kelvin Chikomo for his valuable inputs and suggestions.

DNA research group for sponsoring the equipments that are required for this project.

Ndapandula Nakashole for her support throughout the year.

WarCraftIII® and *DotA* that provide much entertainment.

University of Cape Town, Computer Science Honours class of 2006, it is a great pleasure to be part of the class.

Table of Contents

List of Figures

List of Tables

1. Introduction and Motivation

In the past decade, the number of online banking users has increased rapidly. This has led many developers to investigate on more convenient methods for customers to perform remote transactions. Mobile banking is a new convenient scheme for customers to perform transactions. Mobile banking is predicted to increase as the number of mobile phone users increases.

Mobile banking is a recently new research area. At present, many banks are promoting their mobile banking services heavily. As mobile banking becomes popular, the concern for security of mobile banking is raised. Thorough research is needed to examine the security of using mobile phones to transmit banking details. The security of the mobile phone can provide is limited, this is due to mobile phones has limited functionalities. These limitations have prevented developers to improve the current mobile banking solutions.

This project investigates on the security issues with mobile banking using Short Message Service (SMS) as the communication channel. The investigation is aimed to uncover the security shortfalls of mobile banking using the GSM network and the SMS protocol. Both passive and active attacks are considered. The exposure of the security shortfalls has called for a new secure SMS protocol to be designed to enhance security of SMS banking.

This report provides the research on the security problems with current SMS banking. It provides a protocol solution to ensure SMS banking is secure. A prototype was built to simulate the use of the designed protocol. Testing of the protocol was performed to illustrate the designed solution do provide the security that are required for mobile banking. Experiments were carried out to prove the designed protocol does enhance the security of the standard SMS protocol and it proves the secure SMS protocol has advantages over other mobile banking solutions.

1.1. Success Criteria

The following are the set of success criteria for this project. If all of these criteria are met, it would indicate the project is successful.

- The designed protocol must follow the principle of security services (confidentiality, integrity, authentication, non-repudiation and availability).
- The designed solution can prevent both passive and active attacks.
- The mobile banking application is deployable onto a mobile phone.
- The mobile banking solution must transfer banking details via SMS.

1.2. Report Outline

Section 2: Theory / Background

Discuss the current security implementation of GSM network, SMS protocol and Mobile Banking in South Africa.

Section 3: Problem Definitions and Goals
Identifies the problems with the security of current SMS Banking solution and the goals of this project to build a secure SMS Banking solution.

Section 4: Design
Outlines the design of the protocol solution for secure SMS messaging and the software engineering of the implementation.

Section 5: Implementation / Integration
Describes how the system was implemented, the tools that are used to implement the solution prototype.

Section 6: Final Testing
Describes the tests that were performed to ensure the designed protocol is secure and the implementation is correct.

Section 7: Validations and Results
Describe the designed solution is valid to be a secure SMS messaging protocol and provide a table of security comparison between the current mobile banking solutions and the designed solution.

Section 8: Conclusion
Draws conclusion of the designed secure SMS messaging protocol and the security weakness in the designed solution.

Section 9: Future Work / Maintenance
Suggest possible avenues of future work to enhance security for the designed protocol.

2. Theory / Background

2.1. Introduction of Background

Many countries with mobile communication services use Global System for Mobile Communication (GSM) architecture to network their mobile connections. GSM network was initially designed to be used for voice communication. As the usage of mobile phone increases, people begin to use their mobile phones for additional means of data transmissions.

The most popular type of data transmission is Short Message Service (SMS). This service allows the user to send plaintext messages to the destination mobile phones. The advantages of using SMS are that it is relatively inexpensive and it is reliable for sending non-sensitive messages. SMS messages are sent asynchronously. When a message is submitted for sending, the service provider will keep the sending message in its message buffer until the message is delivered to the destination mobile phone.

Since year 2002, banks in South Africa are heavily promoting mobile banking. Mobile banking is a convenient method to let those who struggle to access to bank services to have an easier banking approach. The intention of mobile banking is to reach out to the unbanked population and to provide them a low cost banking scheme. Banks offer mobile banking services which allow their customers to transmit their banking details via SMS messages. SMS banking service is convenient. The user can perform mobile banking during anytime and at anywhere. The transmission speed of SMS messages is generally fast and SMS message delivery is reliable. The user can expect their transactions to be transmitted and performs in real time.

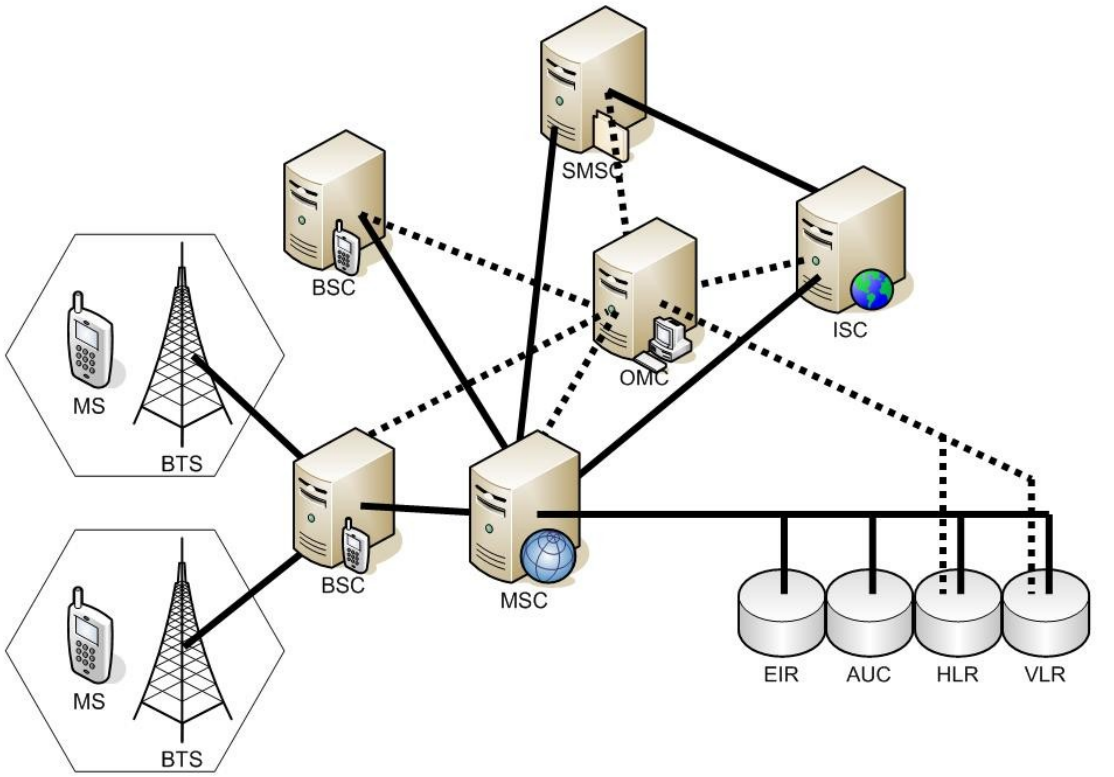
The problems with SMS banking are that the SMS message is not entirely secured. There are many flaws in the GSM architecture which lead to security shortfalls for SMS banking.

In this chapter, we provide an overview of the GSM architecture, an investigation of the current situation of GSM security, SMS security and some researched solutions for the discovered security problems of SMS protocol. At the end of this chapter we will look at some mobile banking services that are currently offering by the banks in South Africa.

2.2. GSM Architecture

In Figure 1, we illustrate an overview of the GSM architecture described by Eberspächer [2]. The solid lines show how the communication signals transfer between the essential components and the dotted lines shows the internal connections use for maintenance.

In a typical scenario, the *Mobile Station* (MS) – the cellular handset of the user – initiates to send signals out. The signals are transmitted from the MS and received at the *Base Transceiver Station* (BTS).



Key:

MS – Mobile Station
BTS – Base Transceiver Station
BSC – Base Station Controller
MSC – Mobile Switch Centre
OMC – Operation and Management Centre
SMSC – Short Message Service Centre
ISC – International Switching Centre
EIR – Equipment Identity Register
AUC – Authentication Centre
HLR – Home Location Registry
VLR – Visitor Location Registry

Figure 1. GSM Architecture

The BTS act as a transmitter and receiver of the radio signals from mobile phones. The BTS translates the radio signals into digital format and then it transfers the digital signals to the *Base Station Controller* (BSC).

The BSC controls multiple BTSs within a small geographical area. The BSC forwards the received signals to *Mobile Switching Centre* (MSC) and the MSC interrogates its databases (*Home* and *Visitor Location Register* (HLR and VLR)) for the location information about the destination mobile handset.

If the signal originates or terminates at the fixed telephone line network then the signal will be routed from the MSC to the *Gateway MSC* (GMSC). If the received signal is an SMS message then the message would be stored in the *Short Message Service Centre* (SMSC) and the message will wait to be delivered. Even after the SMS is delivered, the message content still maintains in the SMSC persistence database. If the signal needs to

be redirected internationally then the signal will be routed via the *International Switching Centre* (ISC) to another country.

The maintenance is controlled by the *Operation and Management Centre* (OMC). The *Equipment Identity Register* and *Authentication Register* (EIR and AUC in respective order) databases are used for equipment verifications and user authentication.

2.3. GSM Security

The current GSM security implementations are described in the following subsections.

2.3.1. Subscriber Identity Confidentiality

In GSM network, the subscriber's identity is uniquely identified by the *International Mobile Subscriber Identity* (IMSI). This identity is store within the *Subscriber Identity Module* (SIM) card which is given by the service provider. The mobile phone can only operate with a SIM card with a valid IMSI when it is inserted into a handset with a valid IMEI[1]. The IMSI is needed during subscriber verification. It is important to keep the subscriber's IMSI secure to prevent the attackers localizing and tracking user's physical location [2]. Therefore the IMSI cannot be transmitted over the air in plaintext.

Instead a *Temporary Mobile Subscriber Identity* (TMSI) is used for identification of the subscriber. The TMSI is only valid during each temporary session when the subscriber verification is needed. The TMSI is limited to local area validity. It must be used in conjunction with the *Location Area Identification*[2] (LAI).

To keep the IMSI and the TMSI secure, they are stored with the service provider inside the VLR database. During verification, the operator uses the IMSI of the subscriber to interrogate the VLR database for the subscriber's TMSI. The operator compares the retrieved TMSI with the mobile station's TMSI for authentication.

To send the TMSI over the air, the content must first be encrypted before it is send. After the end of each successful authentication, the server will send the next TMSI to the subscriber's mobile phone for next authentication.

2.3.2. Authentication of Subscriber's Identity

A *Subscriber Authentication Key* (Ki) is assigned in addition to the IMSI for the authentication of the subscriber's identity [2]. This secret key is used for all security functions and it is pre-computed and installed onto the SIM card by the operator. The service provider has this key pre-stored in the *Authentication Centre* (AUC).

The subscriber's identity is authenticated using A3 algorithm. A3 algorithm requires the mobile phone and the operator to have the same Ki. A random number is generated by the operator to calculate the *Signature Response* (SRES). The operator sends the same

[1] IMEI stands for International Mobile station Equipment Identity. It is use for the service provider to keep records of equipments that are forbidden to operate on their network.
[2] LAI is the identification of the area where the mobile station is sending its signal to.

random number to the mobile station, and then the mobile station will calculate its SRES and sends it back to the operator. The operator authenticates the subscriber by comparing both SRES values. If the SRES is identical, it is assumed the mobile phone is authentic. Figure 2 illustrates the basic flow diagram of the subscriber authentication.

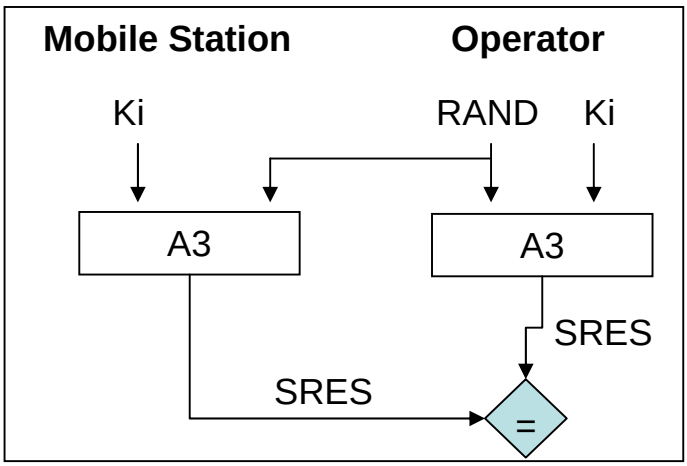


Figure 2. Principle of Subscriber Authentication

2.3.3. Cipher Key Generation

To ensure secure signalling, the data that gets sent across the air must be encrypted. In GSM network, the algorithm used for encryption is A5 algorithm. The *cipher key* (Kc) that is use for encryption is generated at each communicating side using the A8 algorithm. A8 generates symmetric keys that are used by the mobile station and the mobile operator for encryption.

To generate new cipher key, the network operator selects a *random number* (RAND). The RAND is sent over the air to the mobile station. A8 algorithm requires the mobile station and the operator to have the same RAND and the pre-negotiated Ki values. The algorithm takes both values to generate a cipher key (Kc). Figure 3 illustrates the procedure of key generation using A8 algorithm.

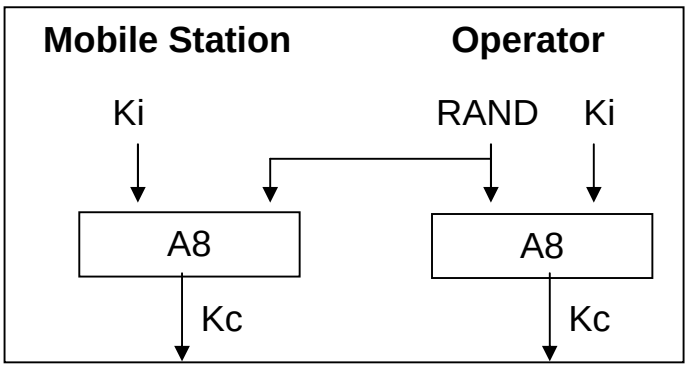


Figure 3. Generation of the Cipher Key

2.3.4. Data confidentiality

Any intruder can intercept signals that are transmitting over the air. The transmitting data cannot be in plaintext. Therefore the transmitting data requires encryption. The mobile station and the operator can use the generated cipher key to encrypt data. The A5 algorithm is used for encrypting data that gets transmitted over the GSM network.

A5 algorithm is a symmetric ciphering[3] algorithm. It uses the cipher key (Kc) and the data as the algorithm input parameters. The algorithm produces the cipher text as output. The ciphered text gets converted into radio signals by the mobile station and sends across the air to the Base Transceiver Station. Figure 4 shows the flow diagram of how the data is encrypted using A5 encryption.

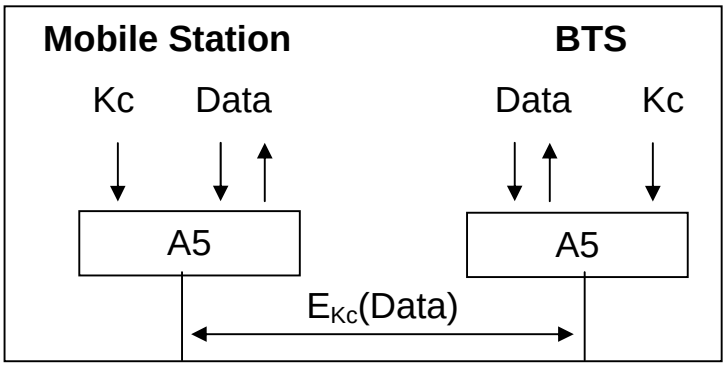


Figure 4. Data Encryption

2.4. Security Shortfalls of GSM Architecture

2.4.1. A5 Ciphering Security Shortfalls

A5 algorithm maintains data confidentiality for the transmission between the mobile station and the base transceiver station. There exist two versions of A5 algorithm; these are namely A5/1 and A5/2. A5/1 is the stronger version of the algorithm. Due to export restrictions on encryption technology, only western European nations and a few other specialized markets are allowed to have this GSM encryption technology [4]. A5/2 is used in many central and eastern European nations, it is deemed as the weaker encryption algorithm [3, 4]. A5/2 is used in area where A5/1 is prohibited due to the restrictions.

The stronger version, A5/1 algorithm, was reversed engineered by Biryukov, Shamir and Wagner [3, 5]. Their method was never published, but it was claimed that the algorithm can be cracked with a moderately high performance computer in real time. Similarly the A5/2 algorithm was also defeated. Wagner and Goldberg took less than a day to crack the algorithm and the method only requires five clock cycles which they have exposed the A5/2 algorithm is vulnerable for sensitive data transmission [3].

[3] Both sender and receiver use the same key to perform encryption / decryption

A newer version of A5 algorithm, call A5/3, was introduced after the announcement of the discovery of flaws in the standard A5 algorithm [6]. A5/3 is specifically designed to protect signals, so sensitive information is protected over the radio channel.

2.4.2. A3 and A8 Authentication Security

The A3 and A8 algorithms are intended to be used for authentications. Wagner and Goldberg had discovered flaws within these algorithms and they proved that it is possible to obtain the K_i [6, 7].

In a realistic scenario, the attacker can clone the subscriber's SIM card by using a SIM card reader. Then the attacker can analyse the security data stored within the SIM card. Once the K_i is obtained, the attacker can write the K_i and the IMSI into another SIM card [6] and the attacker can use the new SIM card to perform masquerading attacks.

2.5. Security Problems with SMS

The initial idea for SMS usage was intended for the subscribers to send non-sensitive messages across the open GSM network. Mutual authentications, text encryptions, end-to-end security, non-repudiation and many security concerns for SMS were omitted during the design of GSM architecture [5]. In this section we uncover some of the security problems of using SMS.

2.5.1. Cell Phone Crashing

De Haas had performed a research of using SMS to crash some Nokia mobile phones that are running with older firmware [3]. The attacker sends a malformed SMS message to the receiver's phone. The malformed message causes a buffer overflow error on the mobile phone operating system and the error causes the phone to stop functioning [9]. After the mobile phone is crashed, the phone might restart unexpectedly, lock up and stop handling SMS messages.

2.5.2. Forging Originator's Address

SMS spoofing attack is when the attacker sends out SMS messages that appear to be from a legit sender. It is possible to alter the originator's address field in the SMS header to another alpha-numerical string. It hides the original sender's address [3] and the sender can send out hoax messages and performs masquerading attacks.

2.5.3. SMS Encryption

The default data format for SMS message is in plaintext. The only encryption involved during transmission is the encryption between the base transceiver station and the mobile station. End-to-end encryption is currently not available. The encryption algorithm used is A5 which is proven to be vulnerable. Therefore a more secure algorithm is needed. The new secure algorithm does not necessarily have to replace the original A5 algorithm.

2.5.4. Denial of Service Attack

There is security vulnerability at the SMS Centre (SMSC). When an SMS message is received at the SMSC, the message gets queued up at the storage buffer. The attacker can

exploit this vulnerability by flooding the buffer queue with multiple meaningless messages to a target mobile number. This type of flooding can causes the SMSC to reject incoming messages for the victim because the storage space is limited in the buffer queue [10].

Even though SMS protocol uses a different signalling channel compare to the normal voice call channel. The data of the SMS is sent through the control channel to the user. The control channel is used for calls initiation. Therefore if the control channel is flooded with SMS messages, then there would be no chance for the call initiation signal to get through and it causes a denial of service attack.

This type of attack is not necessarily applicable. It will not succeed if the service provider has some kind of monitoring system to examine the message buffer queue. Once the monitoring system detects abnormal activities, it can alert the control system to stop receiving message from the attacker.

2.6. Secure SMS Model

2.6.1. SMS Handshaking Protocol

Handshaking
M1: C -> S: $E_{PK_{pub}}$[Username || **Slt** || **SQ** || $\mathbf{R_c}$]
M2: S -> C: E_{SK}[$\mathbf{R_C}$ || $\mathbf{R_S}$ || **SQ**] where $\mathbf{SQ_n} > (\mathbf{SQ_{n-1}} + 1)$
M3: C -> S: E_{SK}[$\mathbf{R_C}$ || $\mathbf{R_S}$ || **SQ**] where $\mathbf{SQ_n} > (\mathbf{SQ_{n-1}} + 1)$

Encrypted Data Transmission
M4: C -> S: E_{SK}[$\mathbf{DT_C}$ || $\mathbf{SQ_n}$] where $\mathbf{SQ_n} > \mathbf{SQ_{n-1}} + 1$
M5: S -> C: E_{SK}[$\mathbf{DT_S}$ || $\mathbf{SQ_n}$] where $\mathbf{SQ_n} > \mathbf{SQ_{n-1}} + 1$
.
.
.
Mn

Keys:
$\mathbf{PK_{pub}}$ - the public key of the server
SK - the session key
Slt - the salt number
$\mathbf{R_C}$ & $\mathbf{R_S}$ - the random number generated by Client and Server respectively
SQ - the sequence number
$\mathbf{DT_C}$ & $\mathbf{DT_S}$ - the data send from the Client and the Server in respective order

Figure 5. Message Handshaking Protocol. Source from [5]

Ratshinanga et al. [8] have suggested a message handshaking protocol using public key and session key encryptions for the communication between the client and the server. Figure 5 illustrates the SMS handshaking protocol. The client initiates the connection by sending its username and a salt number[4] to the server in an encrypted message using the

[4] Salt is an initialization vector of a block cipher. Often specifically salt is an initialization vector used to obscure a pass phrase. http://en.wikipedia.org/wiki/Salt_(cryptography)

server's public key. When the server receives the client message, the message is decrypted using the server's private key. The server retrieves the salt number and the username from the received message, and it also retrieves the relative user PIN from its database. The server can calculate a session key for the client by using a secure key generator with pre-negotiated values as the input parameters. Once the session key is calculated, the server replies an encrypted message using the generated session key. The client can calculate the session key in parallel using the same parameters. Once the client receives a reply from the server, it decrypts the message using its generated session key. A secure connection is established.

The session key is generated individually by hashing the username, salt number and the shared PIN. Attackers cannot generate a session key without knowing the PIN of the user. The salt value provides a more secure session key generation as the salt number is 128 bits long [8], it increases difficulty for the attacker to crack the session key. To prevent replay attacks, a sequence number (SQ) is used. This SQ number is incremented by one, each time when the message reaches its destination. The client or the server can check if the SQ is incremented by one from the previous SQ. If not, the received information can be discarded as it indicates it is out of sequence.

2.6.2. SMS Encryption

Hassinen [11] had built a Java™ application to encrypt SMS messages using mathematical encryption scheme call Quasigroup. The main idea of Quasigroup (Q) is, in a finite set of elements, it is possible to apply a function operator (denoted by *, all operators are invertible and have unique solutions) upon elements *x* from the set Q to produce another element which belongs to the same group. To return the original element *x*, we can apply the inverse operator (denoted by \ right inverse or / left inverse) with the inverse element to retrieve the original element.

In equation form the Quasigroup is defined as, $x * (x \setminus y) = y = x \setminus (x * y)$

Quasigroup encryption is a symmetric key encryption algorithm. To encrypt the sending message, the application applies an encryption support element (leader) *l* upon the first element to produce y_1, such as $y_1 = l * x_1$. The next chain of encrypting elements use the previous encrypted element as the encryption support element to calculate for the next encrypted element, such as $y_{i+1} = y_i * x_{i+1}$ (i = 1… n - 1). To decrypt the message, use the same approach, $x_1 = l \setminus y_1$ and $x_{i+1} = y_i \setminus y_{i+1}$ (i = 1…n - 1). This type of encryption scheme produces a chain of ciphering. It encrypts the current element with the previous encrypted result.

The application built by Hassinen [9] was developed using Java J2ME[5]environment. The application requires the user to insert the password (password assume to be pre-negotiated between the sender and the receiver) for encryption. The application takes this password uses it as the *leader* element for encryption and encrypts the message then

[5] J2ME or Java 2 Micro Edition. API designed for portal devices.

sends it to the destination handset. The receiver handset uses the same password to decrypt the received message.

2.7. Current Mobile Banking Services

2.7.1. Mobile Payment

Mallat [12] defined two types of mobile payment, micro payment and macro payment. Micro payment is the transactions that involves with small value of money, such as the user purchasing mobile contents (ring tone, logos, games and etc.), shops, kiosks and fast food. The use of micro payment is to purchase cheap items and it targets mainly for convenient. It requires minimum security. Macro payment is to pay for large value transactions such as online purchase, restaurants and etc. Macro payment requires secure communication controls and must follows security standards to ensure the users' banking details are secure.

All mobile banking services fall under the macro payment category. To build a safe and secure mobile banking application, it must conform to the principles of security service and theses are confidentiality, authentication, integrity, non-repudiation and availability [18].

2.7.2. WIZZIT

WIZZIT offers the first South African mobile banking services. Their target is aimed to service the population that are un-banked [11, 12]. Their mobile banking service is simple to use. The user only has to dial a string of USSD numbers[6] to connect to the banking server to perform transactions.

2.7.3. SMS Banking in South Africa

Many banks in South Africa use the Wireless Internet Gateway (WIG) technology to transfer SMS banking transactions. The procedure of registration for SMS banking in South Africa is in the following outlines. The client must first register for a bank account with SMS banking service activated. The user must obtain a 32K SIM card. The user can then contact their GSM service provider to activate mobile banking on the user's SIM card and the banking menu will be downloaded onto the SIM card.

To perform banking transactions, the user operates the banking application from their mobile phone. A display menu shows all the different types of transaction options available for the registered user. The user selects his/her wanting transaction and enters the details into the mobile phone. The mobile phone will transmit an SMS to the banking server. Any banking SMS messages from the server will not be stored on the phone, the

[6] USSD is a method of transmitting information or instructions over a GSM network. USSD has some similarities with SMS since both use the GSM network's signaling path. Unlike SMS, USSD is not a store and forward service and is session-oriented such that when a user accesses a USSD service, a session is established and the radio connection stays open until the user, application, or time out releases it. Source from http://www.indie.dk/buzz.htm
Example of a USSD string:*<number>* <service_number>#

reason of this is to increase security. MTN mobile banking instructions have examples and guideline on how to perform mobile banking transactions [15].

Currently South African banks, such as Standard Bank and ABSA use the WIG mobile banking approach. First National Bank (FNB) uses the USSD approach which is similar to what WIZZIT is offering. FNB requires the user to first send an USSD string with the user's PIN to the banking server. Then the server returns a message to notify the user that the server is ready to accept banking SMS message. This approach is not secure because every user's detail is transmitted in plaintext. The mobile network operator has full access into the banking details sent by the user.

2.8. Summary

In this section we provide an analysis of the current security approaches that are mentioned in this chapter.

Table 1. Analysis of the current security approaches for SMS banking

Current SMS Banking Approaches	**Security Shortfalls**
Plaintext SMS messages get sent over open radio interface which are encrypted using A5 algorithm.	A5 encryption algorithm is not entirely secure. Research has shown that this method has flaws and it is vulnerable to attacks.
SMS messages waiting to be delivered are store in the service provider's SMSC storage buffer in plaintext. Even after the message is delivered, the service provider keeps records of all the messages.	If the message content is not encrypted then any personnel who have access to the service provider's SMS data can view the sensitive details.
USSD banking.	The verification depends only on the sender's number, such that if the SIM card is lost or the SIM card is duplicated, the attacker can use the victim's account to perform transaction. The USSD message that gets sent to the bank server is only encrypted between the mobile station and the base transceiver station. The message is in plaintext within the mobile operator's network.
PIN authentication	First National Bank uses USSD to allow their customers to send their authentication PIN. The service provider can read the PIN because it is sent in plaintext.
Some SMS mobile banking solutions use WIG with the SIM menu as the application.	Having the application loaded onto the SIM card makes the mobile banking application SIM card dependent. If the SIM card is lost, the security of the mobile banking is vulnerable.

3. Problem Definitions and Goals

3.1. Security Problem

Current mobile banking services offered by banks are not secure enough. At the time of writing this report, First National Bank (FNB) in South Africa is offering mobile banking service using SMS protocol [1]. What this service offered is, for the users to perform banking transactions; the users send their banking details to the bank server in text message via the GSM network. The GSM architecture has built-in security mechanisms (such as A5 encryptions) to prevent external attackers to read the content of any intersected messages. But at the SMSC, the SMS message is store in plaintext format. The current SMS banking design has neglected the fact that some employees working for the cellular service provider can have access to the transmitted message at the service stations. Therefore using plaintext SMS message to send security details is not sufficiently secure.

3.2. Economic Problem

The second problem of SMS banking is the cost of each mobile transaction is expensive. The cellular network provider charges on the basis of the number of text message sent out from the user's mobile phone. Thus as the number of text message requires for performing a mobile banking transaction increases, the cost of each mobile transaction increases in the same ratio. If the mobile transaction requires sending *N* text messages, the cost of each mobile transaction communication will cost *N* times the cost of a single text message. Therefore the cost of the communication depends largely on the number of message requires to be sent from the mobile phone.

3.3. Goals

The goals of this project are to investigate and uncover the security shortfalls and the possible attacks on the current mobile banking solutions. The second aim of this project is to design and implement a secure messaging protocol using SMS via GSM network and integrate this protocol with mobile banking system. This new approach must overcome the shortfalls existing in current approaches and thus improve the security of SMS banking and minimising the economic effects of such changes.

3.4. Hypotheses

A research approach is adopted to investigate on the security of the current mobile banking solutions. An experimental approach is used to prove or disprove the following hypotheses:

- A secure SMS messaging protocol can be designed and implemented that offers better security.
- The designed protocol can be integrated with mobile banking to form secure SMS banking.

- Secure SMS banking has security advantages over other types of mobile banking protocols such as USSD, GPRS and etc.

4. Design

4.1. Design Requirements

The designed protocol must conform to the principles of security service; these are namely data confidentiality, data integrity, authentication (access control), non-repudiation and availability service [18]. In addition, we believe that the following requirements form core of any secure mobile banking system, and we detail our motivations for including such requirements.

4.1.1. High-Speed Protocol

The designed protocol needs to be executed in real time. It must be able to finish performing the banking transactions in an acceptable time period. The time to execute the protocol should not differ from normal user experience.

4.1.2. Cost Effective

Text messages cost money. Therefore by minimising the number of messages used for a transaction, the cost for the protocol communication will reduce in proportion. The number of messages sent across the GSM network must be as minimal as possible.

4.1.3. Adaptability

The protocol must not be limit to the use of SMS only. It must be adaptable to use other means of communication media that are similar to SMS to transfer the banking transaction to the destination server.

4.1.4. Secure Message

The confidentiality of the message content transferred from the mobile phone to the bank server must be preserved. Any unauthorised individual who managed to obtain the message must not be able to read the secured contents within the SMS message. Only the parties with the correct security details can acquire the message content. If the transmitting message was altered, the receiver should be able to notice the message content is changed.

4.2. System Overview

The structure of the secure SMS banking system is broken down into a three tiers system. The *mobile application* is responsible for generating the secure SMS message and sends the message via SMS across the GSM network to the destination server. The *bank server* has listeners that constantly listen for incoming messages and the server application decodes the received messages into a program interpretable format. The bank server follows the designed secure SMS protocol to verify the security of the received message. The *backend database* contains all the banking details and security details of the users. Figure 6 illustrates a graphical representation of the three tier system.

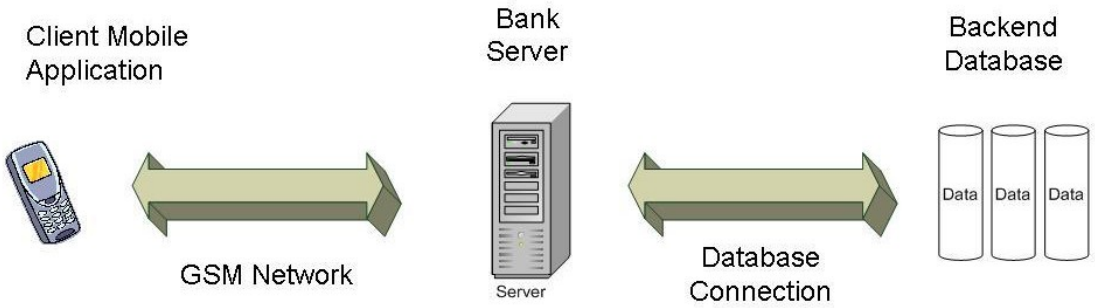


Figure 6. Three Tiers System Overview

4.3. Protocol Layers

The communication process between the bank server and the backend database is independent from the communication between the mobile phone and the bank server. In this project, we are mainly concern with the security of the communication between the mobile application and the bank server. The following figure provides the stack of the protocol layers that the application must follow.

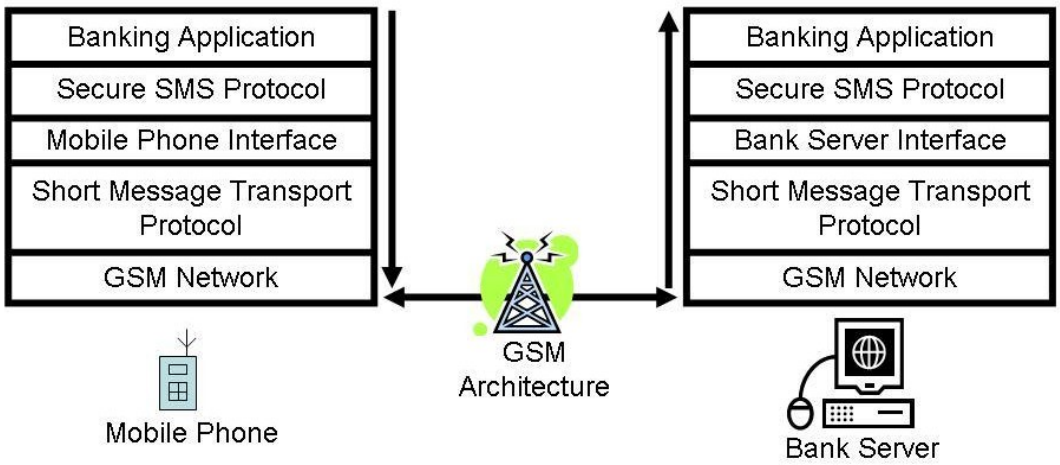


Figure 7. Protocol Layers Stack to Send/Receive Secure SMS

In this system, the mobile phone application is always the transaction originator who initiates the communication. The descriptions of the different layers of protocol illustrated in Figure 7 are as follow:

- The *Banking Application* layer provides the interface between the user and the application. It gets the banking details from the user and translates it into a machine interpretable format.
- The *Secure SMS Protocol* layer is responsible for translating the mobile banking application information into a secure message format and the protocol transforms this message into a standard SMS message.
- The *Mobile Phone Interface* is the mobile phone platform specifications that are defined by the mobile phone manufacturer. Different mobile phone

manufacturers provide different types of programming methods to transmit SMS messages.

- The *Bank Server Interface* is the application programming interface of the bank server. Different programming tools and libraries have it own programming interface to receive or send SMS messages.
- The Short Message Transport Protocol (SM-TP) is the standard SMS transmission protocol[7] used by the GSM network.
- The GSM network has it own security protocol for transmitting data securely across the open air interface.

When the bank server receives a message, the message gets reversed engineer by the server application. The server application decodes the contents of the message. The server application also performs the suitable transaction according to the security contents of the message. After a transaction is performed, the bank server replies a respond message back to the message originator to inform the status of the transaction.

4.4. Design Assumptions

Some assumptions are made before the solution was designed. The following are the assumptions made:

- The users trust the bank server.
- This design protocol provides a secure communication between the mobile phone and the bank. Therefore it is assume that the bank server has it secure communication channels between the server application and the backend database.
- The bank has very high security with their backend database. Therefore it is assume the backend database cannot be accessed externally.
- The users need an account before they can perform mobile banking. Therefore the users must be previously registered at the bank for a mobile banking account.
- During registration, the user is required to self-select a Personal Identification Number (PIN) for authentication – this is the user selected password – only the user and the bank server should know this password.
- It is assume that the bank has a secure method to distribute the mobile banking application to the user's mobile phone. The user has the mobile banking application installed on his/her mobile phone.
- It is assume that the user's mobile phone is compatible with the mobile application. The mobile phone must be able to run the application or else the user cannot perform mobile banking.
- During registration or on user request, the user can receive the list of one-time password securely from the bank. The one-time passwords list can be delivered to the user by the user physically fetching it from the bank or the

[7] For the full specification of the SMS protocol for GSM network, the reader can find the latest version of the specification from http://www.3gpp.org/ftp/Specs/html-info/0340.htm

bank has a secure delivery channel to the user for delivering the passwords list.

- If the passwords list is lost, the user must request a new list of passwords from the bank directly to reset the old passwords and the bank can generate a new list of passwords.

4.5. Protocol Sequences

In the GSM network, SMS messages are sent asynchronously to the receiver. The sender does not know if the receiver got the message until a delivery report is replied from the server. This protocol uses SMS message to send secured banking content, therefore these messages are also sent asynchronously. In Figure 8, we illustrate the full protocol. In Figure 9, we illustrate the sequence steps of the Secure SMS protocol between the user and the bank server.

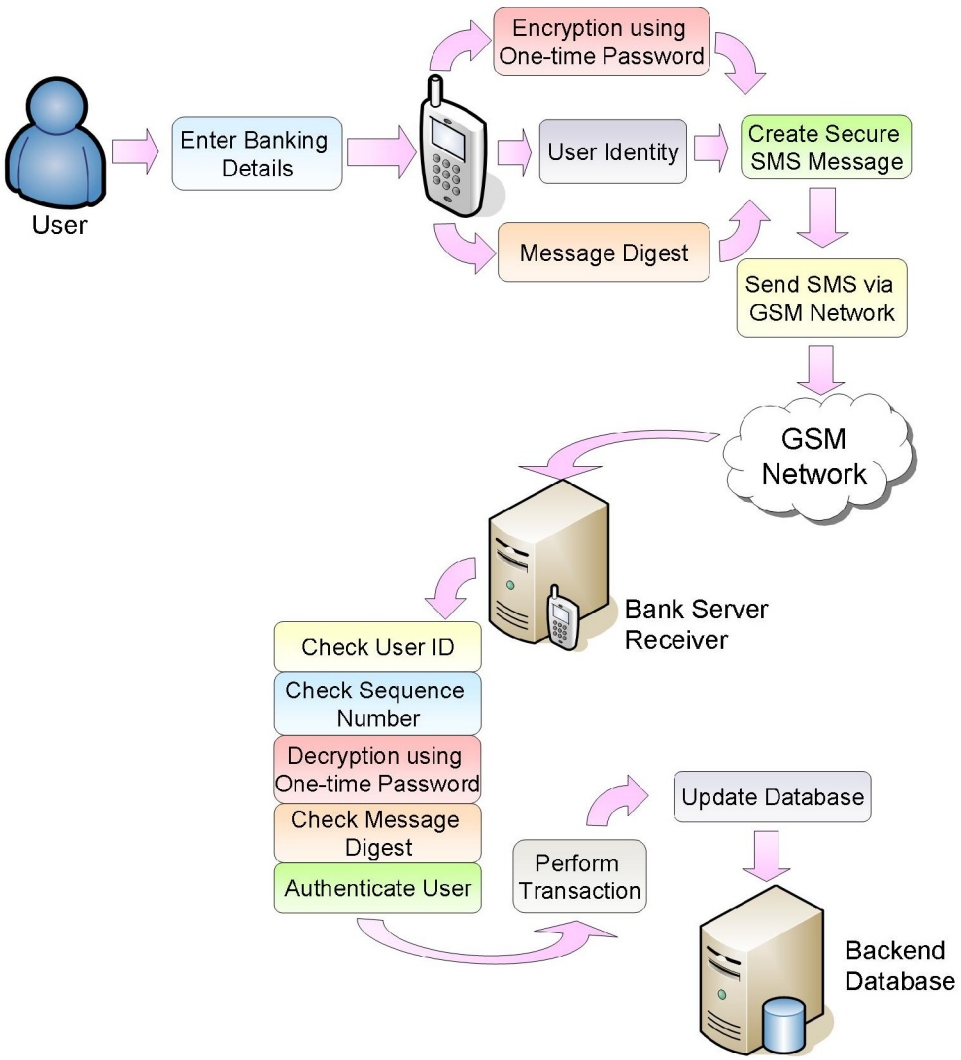


Figure 8. Protocol Overview

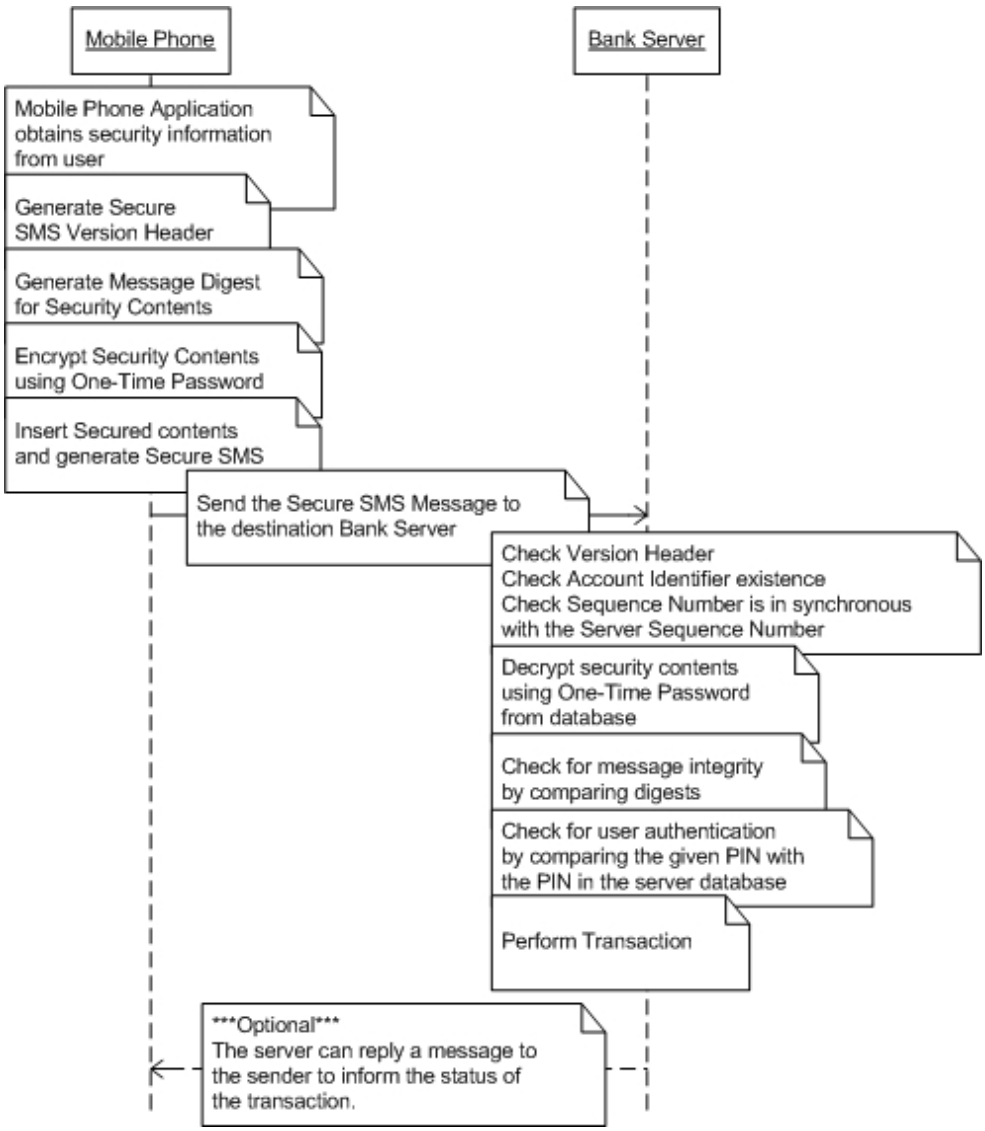


Figure 9. The Sequence Diagram of the Secure SMS Protocol between Mobile Phone and Bank Server

Since this protocol requires the messages to be sent asynchronously, we can think of the protocol to be divided into two parts. The first part is the message generation. The mobile phone generates the message and sends it to the server. And the second part is the message security checks. The server reads in the message and decodes the contents and performs security checks. The following subsections describe each part of the protocol.

4.5.1. Generating and Sending Secure SMS Message

The mobile phone captures all the required security information from the user. The mobile application captures the account identification, the user password, the sequence number and the one-time password from user. This information is used to generate the secure SMS message to be sent to the server. The mobile application has a preset version bytes pattern, this pattern is inserted into the message.

The digest is a single number which can ensure message integrity for the receiver side. The contents that require maintaining integrity is flexible, it is dependent on the needs of the developer. In the next section "4.6.Message Structure", we will describe the fields that are used for calculating the message digest. The requirement of maintaining message integrity is that at least some of the contents that are used for calculating the message digest need to be encrypted. This ensures message integrity because if the message is intersected, the attacker cannot use the encrypted contents to generate another digest. The integrity validation will not pass if any part of the original message is altered.

The fields of content that need to be encrypted are dependent on the needs of the developer. The developers can choose the fields that they required to encrypt. The protocol requires that the message to have some details not be encrypted. This is for the bank to identify the account holder's identity. The algorithm uses for encryption must be a symmetric encryption algorithm. The key used for encryption is the one-time password which is given to the user by the server. The one-time passwords are only known by the server and the user.

After the application completes processing the security contents, the contents are placed in the SMS message according to the structure described in the "4.6.Message Structure" section. It is optional for the developer to include bytes encoding such as Base64 encoding to ease SMS message transmissions.

4.5.2. Receiving and Decoding Secure SMS Message

When the server receives the message from the cellular network, it decodes the message according to the structure described in the "4.6.Message Structure" section.

The server first checks for the version bytes pattern. If the pattern is correct, it is assume that the message is suitable for the secure SMS protocol. During the next step, the server reads in the account identifier from the message and checks if the account identifier exists in the server database. Follow by this, the server retrieves the current sequence number for the given account identifier. The server checks if the sequence number read from the message matches the sequence number stored in the server database.

If the above security checks all passed, the server proceeds to retrieve the one-time password from the database. The password is indexed by the account identifier and the sequence number. Thereafter the server uses the retrieved password as the decryption key to decode the encrypted contents. If the decryption is successful, then the used one-time password is discarded and the server's sequence counter gets increment by the value of 1.

After the decryption, the server reads the secure contents that are required to calculate for the message digest. The message digest is calculated using the same algorithm as the algorithm used by the mobile application. The server compares the two digests for message integrity. If the message is proven to have not been altered, then the server retrieves the PIN – the account holder's personal password – from the message and compares it against the stored account holder's PIN from the server database.

If all of the above security checks pass, the server performs the requested transaction.

4.6. Message Structure

The secured SMS message is divided into multiple fields to accommodate for the various security details required for the protocol. To ease the understanding of the message structure, in Figure 10, we showed the overview of the structure for a secured SMS message. The numbers above the header fields are the minimum number of bytes required for each field for the protocol. The number of bytes for each field can increase depending on the security algorithms used and the implementation requirements. In the "5.Implementation/Integration" section, we will explain the reason why some of the fields are chosen to be that size.

For demonstration purpose, three types of transaction are being simulated in this project. These types of transaction are: *check balance*, *transfer money* and *purchase airtime*. The types of transaction can change depending on the types of services provided by the bank.

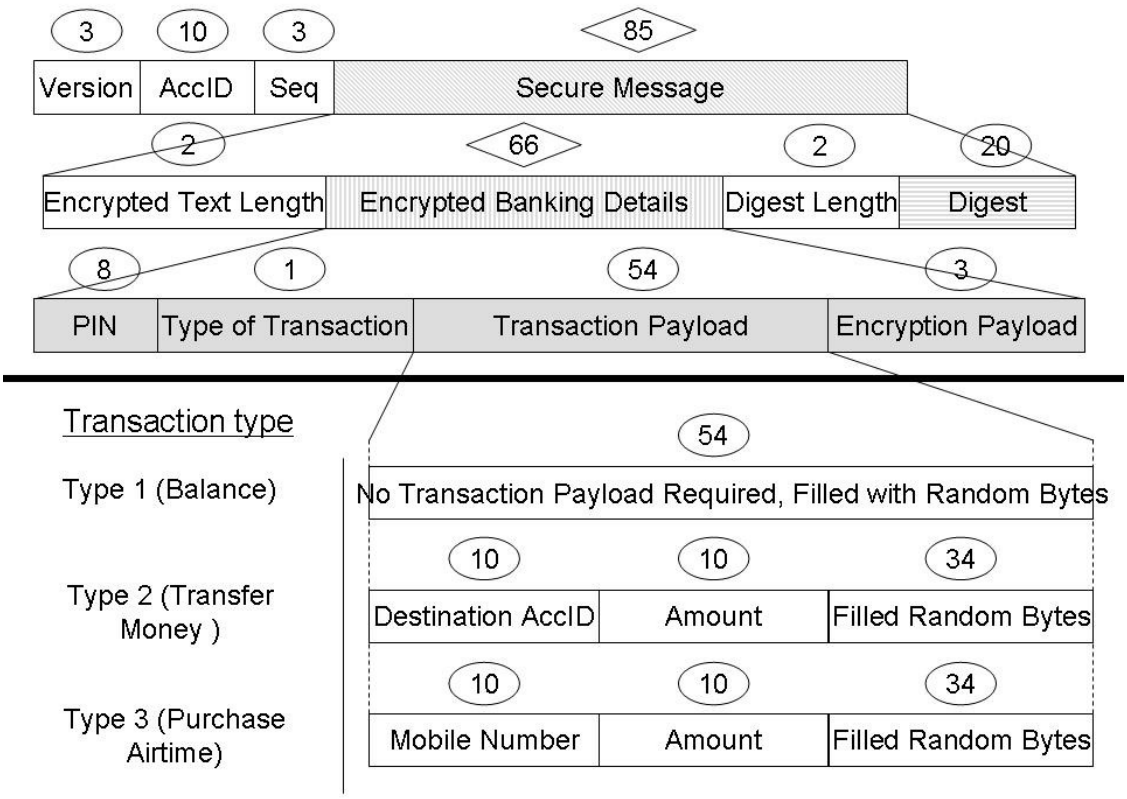


Figure 10. The Structure of a Secure SMS Message

The uses of each labelled structure are explained in the following:

- The *Version* is the mobile application version number used to generate the secure SMS message. It should contain a specified bytes pattern. The

listener application checks if the first three bytes of the received SMS message are valid for the bank application. If the message version number does not match the application version, then the message is discarded. As there are possibilities that the server can receive accidental SMS messages that are not intended for the bank server. The usage of the version bytes is to help to decrease the workload for the server side.

- The *AccID* header contains the bank account identifier of the user.
- The *Seq* header is the user's current sequence of the one-time password. This helps to synchronize with user's passwords list with the server password list. If the sequences are out-of-synchronization then an undetected attack might had happened.
- The *Encrypted Text Length* header contains the number of next bytes that are the ciphered message. The use of the encrypted text length is to assist the server program to know how many bytes from the received message should be the encrypted message.
- The Digest Length header contains the value of the number of next bytes that contains the message digest. The usage of the digest length is for the server to know how many bytes form the message digest. As different algorithm produces different size of digest, it is necessary to keep the digest length within the message.
- The *Digest* header contains the calculated digest value of the message. The use of the digest is for the server to check for message integrity. For those headers that require integrity check, their total digest is calculated using a hashing algorithm. For the secure SMS banking protocol, a single digest of the following fields is calculated: *Version*, *AccID*, *Seq*, *PIN*, *Type of Transaction* and *Transaction Payload*.

The contents of the following headers are encrypted using the one-time password entered by the user.

- The *PIN* header contains the user predefined password. This is used by the bank application to authenticate the user.
- The secured SMS message can be used for different types of transactions. The *Type of Transaction* header is used by the bank server application to identify the type of transaction it should perform.
- The *Transaction Payload* is the extra data that are used for a transaction, but it is not used for any security purpose. The content of the Transaction Payload depends on the type of transaction requested.
    - If the message contains request to check for account balance, the protocol requires no transaction payload. But for security purpose, the payload is filled with random bytes so the encrypted content can appear more scrambled and the randomness can prevent brute force attacks.
    - If the transaction is requesting to transfer money, the server requires the destination account identifier and the amount to be transferred. The extra bytes in the payloads are filled with random bytes.

- If the transaction is requesting to purchase airtime, the server requires the destination phone account identifier which normally it is the mobile phone number and the amount to be transferred from user's account to the phone account. The extra bytes in the payloads are filled with random bytes.

4.7. Protocol Security

The design protocol must conform to the principles of security service. The following subsections describe how the protocol achieves security for each principle.

4.7.1. Confidentiality

This is achieved by encrypting the message using a symmetric secret one-time password which is shared between the user and the bank server. The strength of the confidentiality depends on the security strength of the passwords generation algorithm and the strength of the ciphering algorithm used. It is assumed that only the authorized user will know his list of passwords and the passwords are never shared with other people.

4.7.2. Integrity

The message digest is the hashed value of the message security contents. If any of the contents were altered, the hashing algorithm will generate a different digest value at the receiver side. The digest calculated by the sender that is attached within the sent message are compared with the digest value calculated by the server. If the digests mismatched, the receiver will know the integrity of the message is compromised. The strength of the integrity checks depends on the strength of the algorithm that generates the digest value and the strength of the encryption algorithm used for confidentiality.

4.7.3. Authentication

For the bank server to authenticate the user, the user must provide his/her authentication detail(s) to the bank server. This authentication process is performed by validating the message PIN with the server stored PIN which is previously selected by the user during the registration procedure. The strength of the authentication depends on the password selection strategies used.

4.7.4. Non-Repudiation

Only the account holder and the bank server are supposed to have the one-time password for encryption. The bank server does not generate the same one-time password more than once. Therefore every one-time password is unique in the server's database. Each pair of one-time password and sequence number is only allowed to be used for a single user. The bank server cannot generate the same pair of password and sequence number for different users. Therefore the user cannot deny not sending the message because only that specific user has that unique pair of password and sequence number to encrypt the message.

4.7.5. Availability

The availability of this protocol depends on the availability of the cellular network provider. If the cellular network is congested, the time to deliver the secured SMS message will be time-consuming.

Message decryption and calculating message digest can cost much of processing power. The protocol minimises the workloads of the server by discarding any message that causes the security verifications to return failed. This can decrease the workloads on the server side when the attacker tries to congest the server with random messages.

The number of transactions that the server can handle at once depends on the hardware capability. If the server hardware can handle multiple incoming messages then the server can perform multiprocessing to accommodate for more requests. The protocol has no restriction on the type of hardware needed. Therefore it is up to the developers to decide the hardware specifications of which are the most appropriate for their needs.

Table 2. Comparison of security between the current SMS banking and the designed Secure SMS Banking

Current Approaches	Designed Protocol Approaches
GSM architecture uses A5 algorithm to encrypt radio signals Service provider keeps a record of transmitted SMS messages in their database.	Uses a strong encryption algorithm to encrypt the message before sending the message. The client mobile phone and the server have applications that can handle the encryption/decryption.
Single Password or PIN authentication.	Use One-Time Passwords. The used passwords are discarded. This approach ensures that even if the attacker obtains the used password he cannot authenticate him/her self.
SMS handshaking allows the exchange of keys between the sender and the receiver which allows the application to perform session negotiation.	Sending SMS is slow. The application should avoid the use of many SMS messages to set up a secure messaging connection. The use of One-Time Password to encrypt the message can provide the same security because the encryption details are never repeated.
The verification of USSD banking depends on the sender's number. The USSD message that gets sent to the bank server is encrypted between the mobile station and the base transceiver station.	Encrypting the message and the user authentication detail is essential. This approach provides less security threat because the message cannot be read by the attacker and the attacker needs to know the authentication details to perform banking transaction.

4.8. Use Cases

4.8.1. Use Case 1: Starting Application

The user (account holder) wants to use his/her mobile phone to perform a remote banking transaction. The user opens the mobile banking application and chooses the banking transaction that he/she wants to perform.

4.8.2. Use Case 2: Selecting Transaction

The choices are to check current balance, to transfer money from the user's account to another account or to purchase airtime for any mobile phone account. Once the user has selected his/her option, the mobile application requires the user to enter the banking details.

4.8.3. Use Case 3: Sending Message

Once the details are entered, the user confirms to send the message to the destination bank server. The mobile application generates a secure SMS message (encrypt content and calculate integrity digest) containing all the required security details, then it sends the message as an SMS across the GSM network to the destination server.

4.8.4. Use Case 4: Server Security Checking

When the server receives an SMS, the server application reads in the message and decrypts the banking details. It verifies the message integrity and it performs the account identification authentication. It then performs the banking transaction according to the information given in the message.

4.8.5. Use Case 5: Server Reply Message

After the transaction is done, the bank server sends a confirmation SMS message to the account holder to inform the transaction was safely completed. If any problem occurs, the server sends an error message to the account holder's mobile phone number.

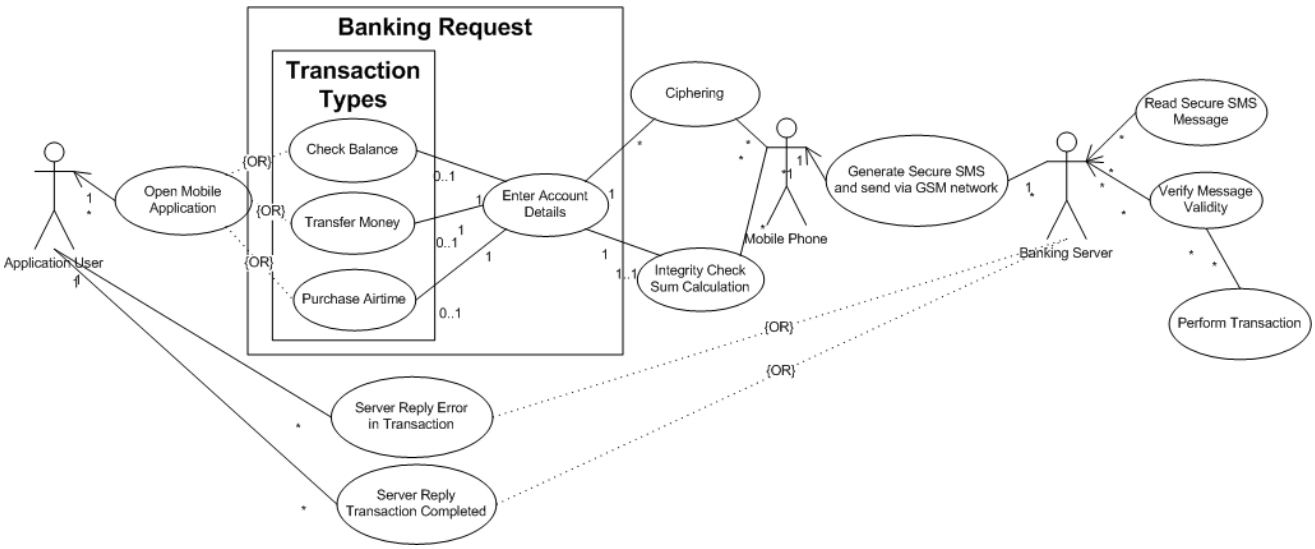


Figure 11. Overall System Use Cases Diagram

4.9. Conceptual Class Collaborations

The UML class diagram shown in Figure 12 illustrates the design of the Mobile Application package, Bank Server package and Backend Database package in respective order. Each of the classes is discussed in a subsection of each package.

4.9.1. Mobile Application Package

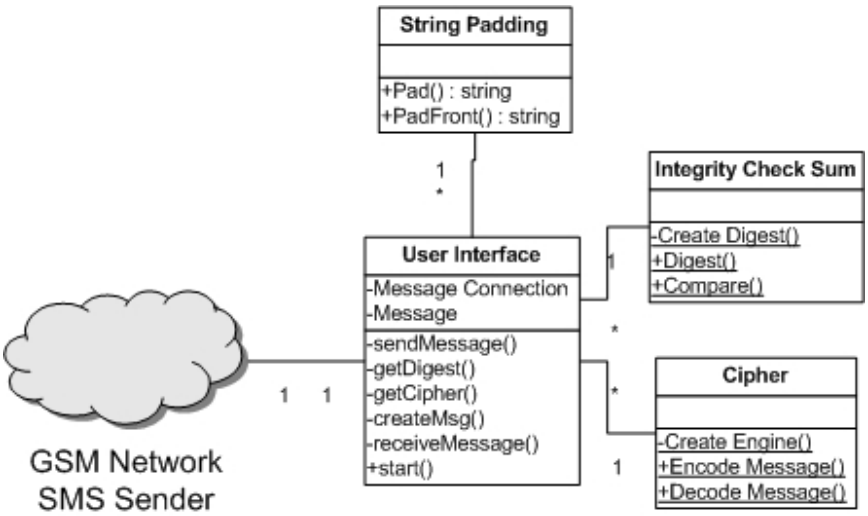


Figure 12. Mobile Application Class Diagram

4.9.1.1. User Interface Class

This class is responsible for the interaction between the account user and the mobile banking application. It act as the driver class of the client application, it consist of the graphical user interface which allows the user to enter his/her banking details.

The "start()" method is the main method and it is used for starting the application. The "sendMessage()" method is use for sending out SMS messages using the mobile phone programming interface. The "receiveMessage()" method listens for incoming coming message such as server's reply message. When a new message comes in, the method will announce a new message is received. The "getDigest()" is used for retrieving the message digest and the "getCipher()" methods is encrypting the message. The "createMsg()" method is used to convert the secure message contents into the SMS message format. The internal Message Connection object is the SMS message connector. It has methods to use the mobile phone to send out SMS messages. The Message object is the SMS message class object.

4.9.1.2. String Padding Class

Given an input string value and length value, the String Padding class has the "Pad()" method to extend the back of the input string until to the length of the input length. The extra characters are either padded with random characters or programmer defined characters. The "PadFront()" method is used to pad characters into the front of the input string.

4.9.1.3. Integrity Check Sum Class

This class is responsible for calculating the message digest for the integrity checks. The method "Digest()" is used for calling to get the message digest and the "Compare()" method is used for comparing the message integrity with a given digest.

4.9.1.4. Cipher Class

This class is responsible of ciphering the message. After the user confirms that his/her banking details can be sent, this class has method to encode the input message and the method outputs the encrypted message bytes. This class also has method to decode the encrypted message into original form. The encryption must use the user password as the key to encrypt and decrypt the message.

4.9.2. Bank Server Package

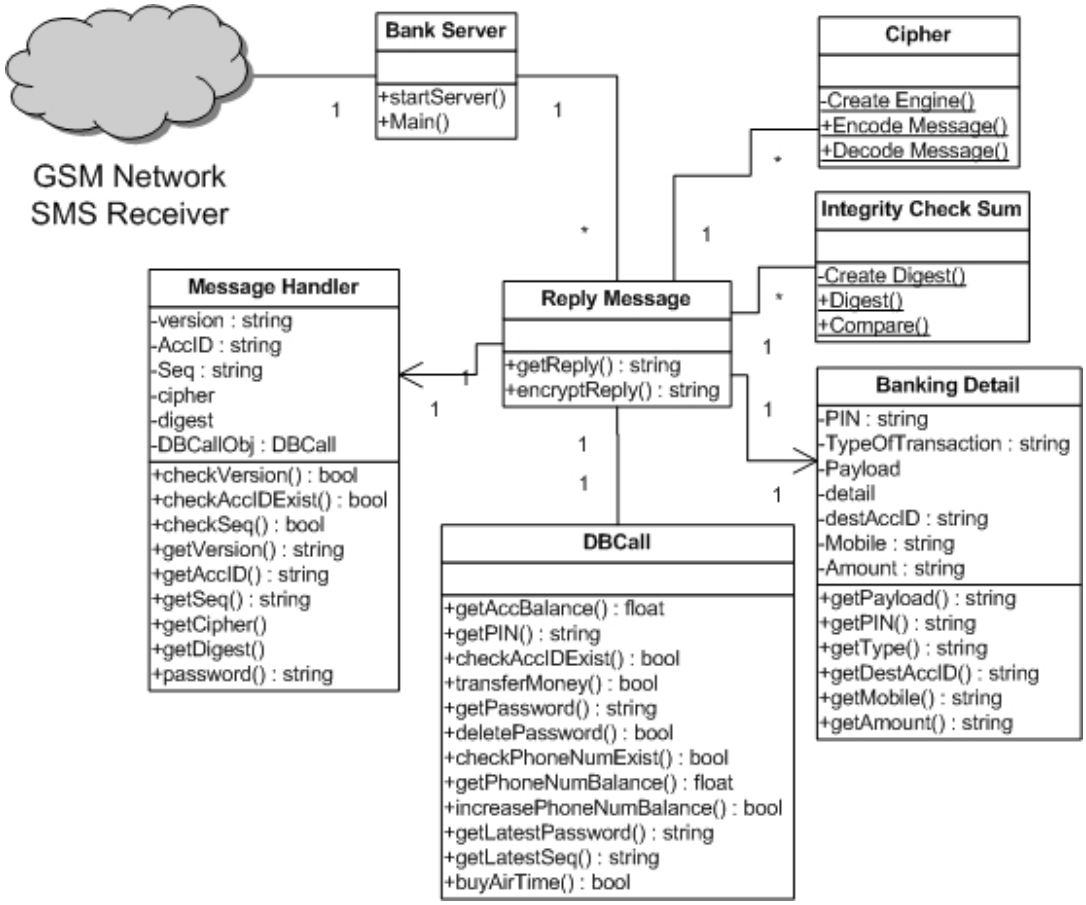


Figure 13. Bank Server Class Diagram

4.9.2.1. Bank Server Class

The *Bank Server* class runs the hardware (i.e. GSM Modem) that receives SMS messages. The "startServer()" method initialize the class to process incoming messages. The "main()" method starts the Bank Server application. Once the server is started, it listens for incoming messages for indefinitely. When a message is received, the *Reply Message* class will generate a reply message based on the content of the received

message. Once a reply is generated, the Bank Server will reply an encrypted message to the message originator.

4.9.2.2. Reply Message Class

This class handles all the received messages. The "getReply()" method takes in the received as the input parameter. This method follows the designed protocol to process the message. It processes the message by breaking the message down into the structure described in "4.6.Message Structure" section. After the "getReply()" method is called, the reply message call be password encrypted. Bank Server can call the "encryptReply()" method to encrypt the message.

4.9.2.3. Message Handler Class

The *Message Handler* class is use to break down the received message into unprocessed message fields. These fields are version number, account identifier, sequence number, cipher text and message digest. These are described in "4.6.Message Structure". After the received message is passed into the *Message Handler*, various methods can be called to retrieve the request message fields. This class has methods to check if the identification details are correct, it can check for version number, account identifier and sequence number.

4.9.2.4. Banking Detail Class

The *Banking Detail* class encapsulates the message contents after it is processed by security algorithms. It contains data that are used by the bank to authenticate the user and to perform transactions. The message fields that it contains are PIN, Type of Transaction and Transaction Payloads. The Transaction Payloads are data that are not used for security purpose and it is only used for transaction purpose. The Banking Detail class has methods to retrieve these fields' value.

4.9.2.5. Database Call (DBCall) Class

The *Database Call* class has methods to connect to the database, to retrieve values from the database and to update values in the database. The values retrieved from the database are used for the message security checks. The values get updated to database are used for banking transaction.

4.9.2.6. Cipher and Integrity Check Sum Classes

These two classes are identical to the classes described in "4.9.1.3.Integrity Check Sum" and "4.9.1.4.Cipher" sections.

4.9.3. Backend Database Package

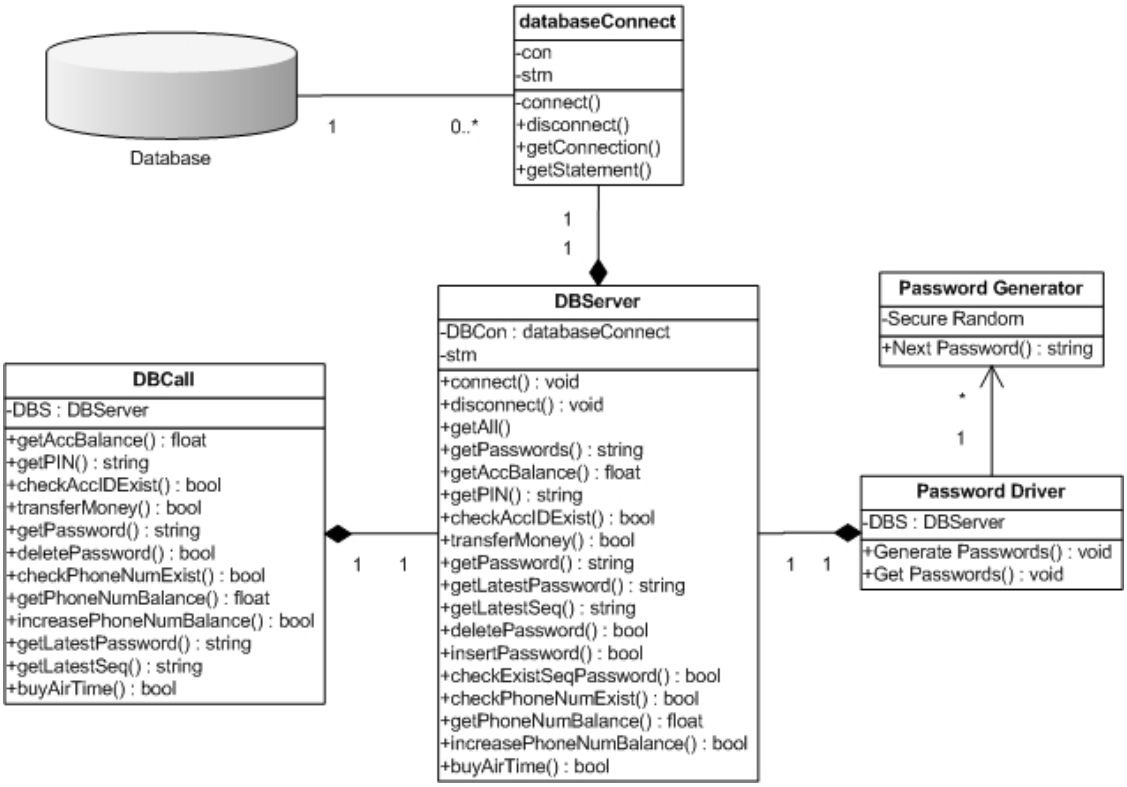


Figure 14. Backend Database Class Diagram

4.9.3.1. Database Connect Class

The Database Connect class has internal methods to build an object database connection (ODBC) with the database server. This class builds a connection to the database server during class initialization (calling the constructor). It maintains a statement object to execute different database statements. The "getConnection()" and "getStatement()" methods are use to retrieve the connected connection object and statement object.

4.9.3.2. Database Server Class

The *Database Server* encapsulates a *Database Connect* class object to keep its connection to the database. This class has various methods to insert, retrieve and update the database. The class methods take input parameters and convert it into database statement (e.g. SQL statement) and it execute the database statement.

4.9.3.3. Database Call (DBCall) Class

The *Database Call* class in the *Backend Database* package is identical to the one in the *Bank Server* package. The *Database Call* class has methods to call methods from the *Database Server* class. This class is like an interface of the *Database Server*. The interface separation can ease the developers to change their choice of database languages. The developers only have to change the database statement in the *Database Server* class, while the interface of the class maintains the same. The *Database Call* class constantly

listens for incoming connections from the bank server. The methods in the *Database Call* class have simplified the methods offer in *Database Server* class.

4.9.3.4. Password Driver Class

The *Password Driver* class is an external database application. This class has it own user interface. It generates the list of one-time passwords for different account holders. This application is designed to be used by the bank; the account client has no authority to run this application.

The method "GeneratePasswords()" takes an account holder's identifier as the input parameter and it generates a list of one-time passwords for that account holder. The generated passwords are inserted into the database directly. The program checks if the generated password exists in the database, if it already exist, the generated password is discarded and a new password will be generated. This keeps the list of passwords in the database to be unique as no duplicates are generated. The "getPasswords()" method retrieves the list of passwords for the given account identifier. The passwords are displayed on the user interface and it can be written to file.

4.9.3.5. Password Generator Class

The Password Generator class generates a random password. It has a Secure Random object which generates non-predictable random numbers. The developer can choose their choice of secure random number generating algorithm. The "NextPassword()" method produce a random password for the method caller.

4.10. Sequences Overview and Descriptions

Figure 15 illustrates the sequence diagram of the overall system. It describes the entire process of how a customer can interact with the mobile system to perform a secure SMS banking transaction. The subsections that followed explain the sequences step by step.

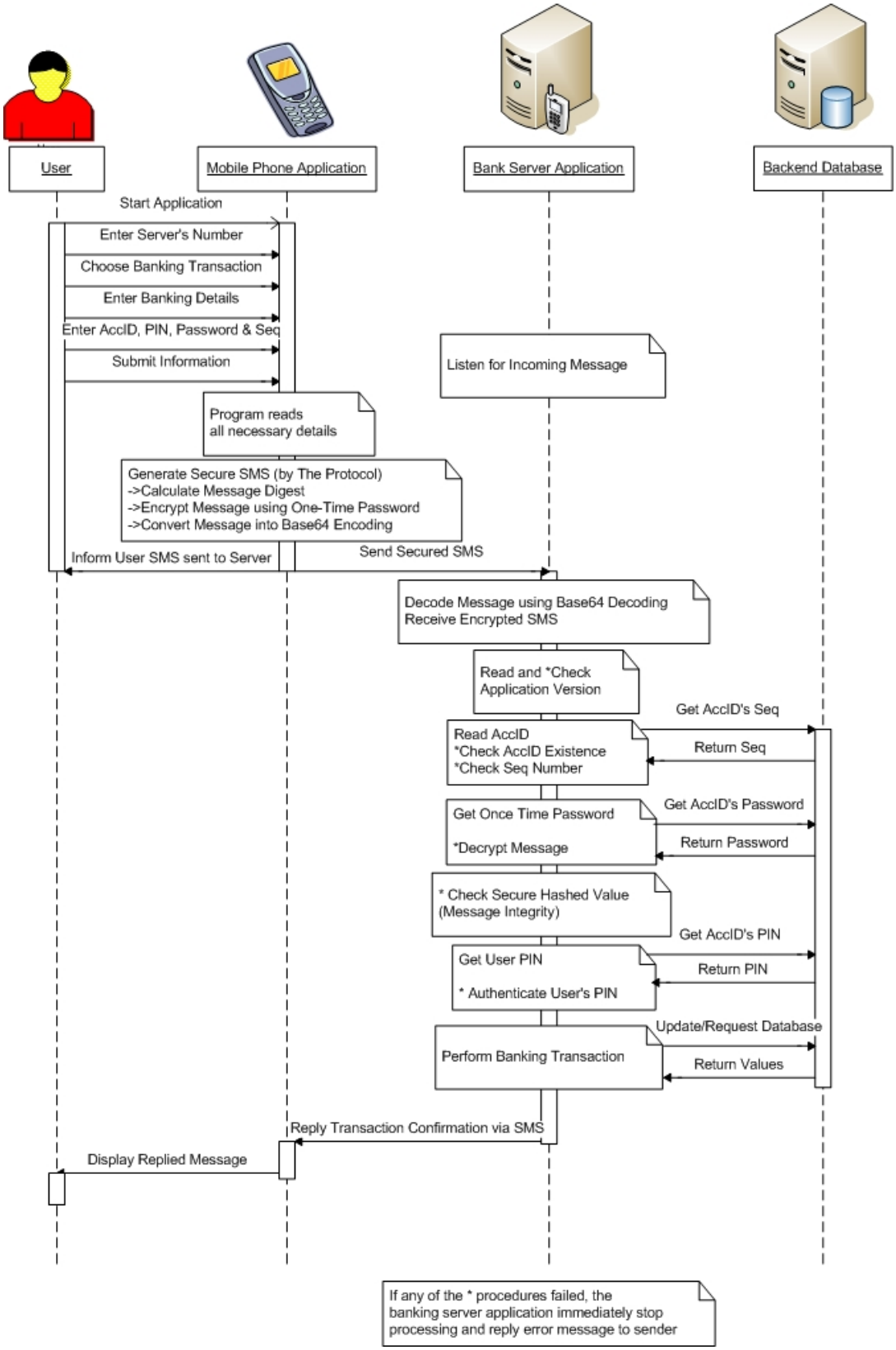


Figure 15. System Sequence Diagram

4.10.1.User - MIDlet sequences

The sequence begins when the user initiates the secure SMS mobile banking application with his/her mobile phone. After the application is started, it allows the user to choose the type of banking transaction that the user prefer (check balance, money transfer or purchase airtime). The user chooses the type of transaction he/she wants, and then the program requires the user to enter the banking details. The banking details include account number, account PIN, sequence number, One-Time Password and etc. These details are for security purposes.

Once the user has entered all the required details, the user can choose to submit his/her details to the mobile application. This action will inform the mobile application to retrieve the user's information from the application displayed form, and generates a secured message. The process to generate a secure SMS message is, the application calculates the message digest for the whole message, encrypts the security content and puts the message in SMS format.

This secured message will be sent in SMS format via the GSM network to the mobile banking server. After the application sends the message, it will display a confirmation message on the mobile screen to inform the user that the message is sent.

4.10.2.Mobile Phone J2ME MIDlet – Bank Server Application

The bank server application listens for incoming SMS messages from the listening port that is connected to the GSM modem. Once an SMS is received, the server application starts a new process thread to serve the received message.

This new thread will first read the application version number from the message. This is to check if the received message is a recognisable message generated from the correct mobile application. If the server fails to validate the version number to be recognisable, then the server stops any further processing immediately and it ignore the received message. If the message is validated to be the right version, the server application reads the required data from the backend database to authenticate the message sender (more details on authentication in section "4.10.3.Bank Server Application – Backend Database"). If any of the authentication process fails, the server will reply a message to inform the user that the transaction failed.

4.10.3.Bank Server Application – Backend Database

After the bank server application validated the message to be recognisable, it retrieves the account identifier from the received SMS message and it uses this identifier to request for the current sequence number of the account holder's passwords list. This process helps the server to compare if the list of one-time passwords on the server side is synchronized with the user's list. If the sequence is synchronized, the server retrieves the current one-time password from the database and decrypts the message.

After the message is decrypted, the server can calculate the message digest using part of the message content with the decrypted banking details, these are version number, account identifier, sequence number, PIN, type of transaction and transaction payload. It

compares the received message digest with the calculated message digest for message integrity. If it passes the integrity checks, the server application will retrieve identifier's PIN from database and compares it with the PIN attached within the message for the user authentication.

If the PIN authentication is verified, the server will perform banking transaction for the user and updates the database according to the details given by the message. Then the server increments the password sequence number in the database. Once the transaction is complete, the bank server replies a message via SMS to transaction sender to inform the message is received and the transaction was performed. The reply message is in secured form. It is encrypted using the account identifier and the PIN as the encryption key and a message digest is attached with the reply message to ensure the message integrity.

5. Implementation / Integration

5.1. Overview

The class packages described in the "4.9.Conceptual Class Collaborations" section have each been implemented as a separate application. The overall system is consisting of four different applications. These applications are the *mobile banking* application, the *bank server* application, the *backend database* application and the *passwords generator* application.

The mobile banking system is built on Sun® Java™ development platform using the Java™ programming language. A number of external libraries were used to develop the mobile banking system. The libraries used for each application are described in the following sections.

The Mobile Banking application does not communicate with the Backend Database application directly. It only communicates with the Bank Server application and the Bank Server will communicate with the Backend Database application to update values. Only the Bank Server application has authority to access the database. This system implementation conforms to the three tiers system described in "4.2.System Overview" section.

5.2. The Mobile Banking Application

Many programming languages were considered for implementation of the mobile application such as Microsoft .Net C#. These are the advantage of implementing the mobile application using Java™ technology.

- Most mobile phones being sold on the market come standard with built in Java™ virtual machine.
- Java™ mobile applications are portable.
- Java™ technology has many libraries to assist the mobile application implementation.

Therefore the decision of using Java™ to implement the mobile application was decided.

The Mobile Banking application was built using Java™ 2 Micro Edition[8] (J2ME or Java™ Wireless Toolkit) development kit running with NetBeans 5.0[9] as the integrated development environment. The *Mobile Banking* application uses the NetBeans Mobility Pack 5.0 to simulate the mobile phone environment. The application was deployed on Nokia 6020 and Samsung D600 mobile phones. Since the mobile application is built using Java™ technology, it should be able to run on any mobile phone that has Java™ virtual machine.

To enable the application to send and receive SMS message on the mobile phone, the

[8] Available from http://java.sun.com/javame/downloads/index.jsp
[9] Available from http://www.netbeans.org/

J2ME Wireless Message API (WMA) library was used. This class is come standard with the J2ME development kit. The WMA library has various classes and methods to construct an SMS message. It has classes to handle binary data (8 bits per byte) SMS messages and text characters (7 bits per character) SMS messages separately. To enhance mobile banking application security, the sent and received messages are not stored on the mobile phone.

Before a banking transaction is sent to the banking server, the message content must be processed to ensure message security. The Bouncy Castle Lightweight API[10] library provides the security tools that are needed.

The following subsections describe the implementation of each part for the Mobile Banking application.

5.2.1. User Interface

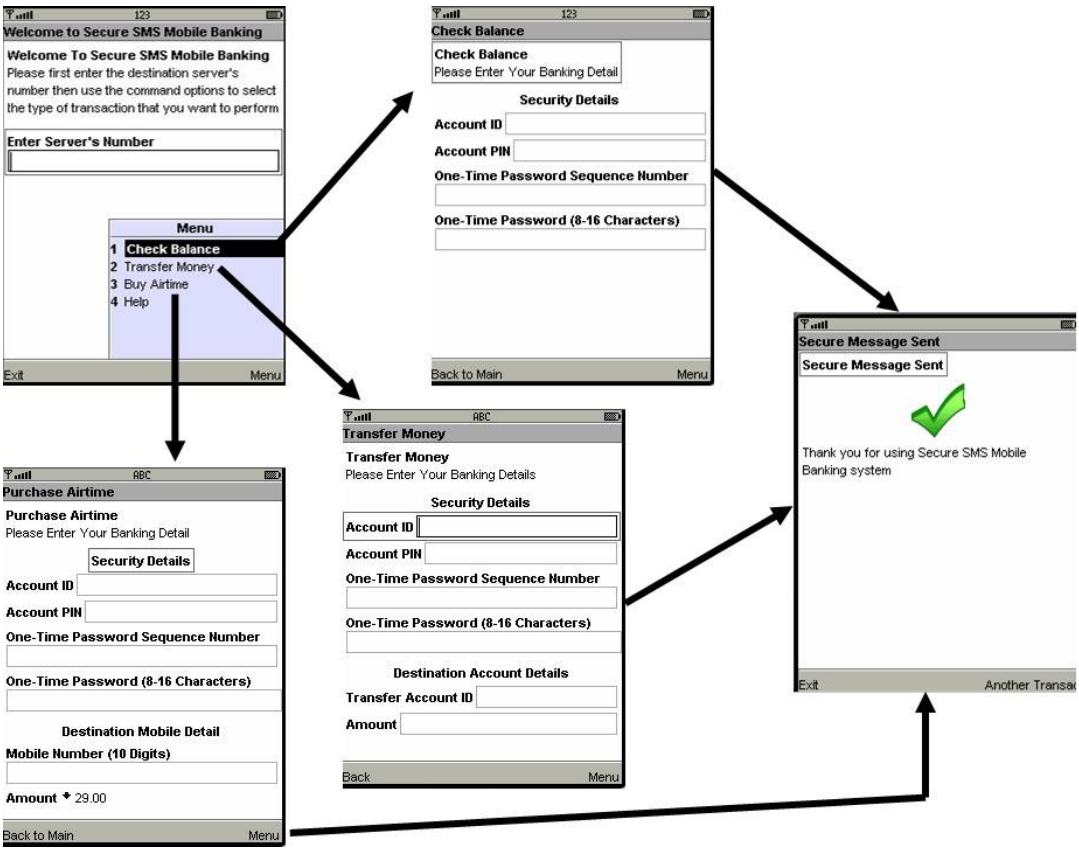


Figure 16. Screen Shots of the User Interface of the Mobile Banking Application

The figure above illustrates the screen shots of the mobile banking application user interface running with the Java™ Wireless Toolkit emulator. The first screen displays the welcome form which requests the user to enter the bank server destination number. The user selects the type of transaction he/she want to perform. The form that gets display

[10] Available from http://www.bouncycastle.org/download/lcrypto-j2me-134.tar.gz

next is depending on the type of transaction selected by the user. The user can enter the correct details into the form. The user can then choose to send the message to the banking server. This action will invoke the application to generate a secure SMS message and sends it to the destination number as the entered server's number. After the message is sent, a confirmation screen will be displayed to inform the user that the message is sent to the server.

There are restrictions on the form's textboxes. It restricts the data that the users allow to enter into the form's textboxes and these restrictions are the following:

- The phone number textbox must contain a string in numerical format and it must not be more than 15 numbers.
- The sequence number textbox must contain a number in the range of 000 to 999.
- The amount textbox cannot contain negative numbers.
- The user's account identifier cannot be the same as the transfer account identifier.
- There are minimum and maximum characters for each field. This is set to prevent the user entering the wrong details.

5.2.2. Encryption / Decryption

For this project Advance Encryption Standard (AES) algorithm is used to cipher the banking messages. The AES algorithm is provided by the Bouncy Castle library under the crypto package.

The encryption and decryption methods are implemented in the class call codec. The encoding methods take in either a string value or a byte array as the input message and encrypt the input message using the password given in the parameter. If a string value is passed as the message into the method, the method converts the string into a byte array and encrypts it into a ciphered byte array. The following Java™ code shows the implementation of the encryption method.

```
    public static byte[] encodeMessage(byte[] content, String password, String
cipherAlg)
    throws Exception{

        byte key[] = password.getBytes();   //convert password text to bytes

        BufferedBlockCipher cipherEngine = new
PaddedBufferedBlockCipher(createEngine(cipherAlg));  //create cipher engine
        cipherEngine.init(true, new KeyParameter(key)); //initialise the cipher
for encryption

        //Encrypt message
        byte[] cipherText = new
byte[cipherEngine.getOutputSize(content.length)];
        int cipherTextLength = cipherEngine.processBytes(content, 0,
content.length, cipherText, 0);
        cipherEngine.doFinal(cipherText, cipherTextLength);

        ByteArrayOutputStream out = new ByteArrayOutputStream();
        DataOutputStream dout = new DataOutputStream(out);

        dout.writeShort(cipherText.length);
        dout.write(cipherText);
```

```
        return out.toByteArray();
    }
```

The above code shows the method takes in the message content as a byte array, the password as a string. The cipherAlg parameter is for selecting the ciphering algorithm to use. It is used to ease modifying the choice of encryption algorithm. Any symmetric key encryption algorithm can be used. For this project we used the AES encryption algorithm.

The encode method first create a cipher engine, this cipher engine is the AES encryption object. The `encodeMessage` method passes values into the cipher engine to encrypt message content. When the cipher engine is initialized, the first parameter of the line `cipherEngine.init(true, new KeyParameter(key))` is set to true. This has indicated that the cipher engine is used for encryption. To use the cipher engine for decryption, the value for the first parameter must be set to false to indicate the cipher engine must perform decryption.

AES encryption requires the encryption key to be in length of 128bits, 192bits or 256 bits. For this project it is sufficient to use 128bits key length for encryption. The key to encrypt the message comes from the password parameter from the method's header. Since the key length we are using is 128bits, therefore the password must convert to 128bits which is equivalent to sixteen 8bits characters.

After encryption, the size of the output ciphered message is increased slightly. The extra bytes are the Encryption Payload field that is described in the "4.6.Message Structure" section.

5.2.3. Message Digest

The algorithm chosen to calculate the message digest is the Secure Hash Algorithm (SHA-1). The SHA-1 algorithm is also provided by the Bouncy Castle library under the crypto package.

The message digest calculation is implemented in the class call integrity. This class is use for checking the integrity of the message. It consists of two public methods, the first method is to calculate the message digest and the second method is to check if the message integrity has been compromised.

The following code shows the implementation of the method to calculate the message digest for the given content.

```
public static byte[] digest(byte[] content, String digestAlg)
throws Exception{
    //Calculate message digest
    Digest digestEngine = createDigest(digestAlg);
    int digestSize = digestEngine.getDigestSize();
    byte[] digest = new byte[digestSize];
    digestEngine.update(content, 0, content.length);
    digestEngine.doFinal(digest, 0);
    return digest;
}
```

The above method takes the bytes array content as the message that need a message digest. The digestAlg parameter is for the method caller to select the algorithm to be used for calculating the message digest. The digest engine object creates an SHA-1 message digest object. The method uses this engine to retrieve the message digest.

To check for message integrity, the following code shows the implementation of the checking for message integrity.

```
    public static void compare(byte[] digest, String message, String digestAlg)
    throws Exception{
        byte[] calculatedDigest = digest(message.getBytes(), digestAlg);
        if(calculatedDigest.length != digest.length)
            throw new Exception("Digest size mismatch");
        //compares digest byte by byte
        for(int i = 0; i < calculatedDigest.length; i++)
            if(calculatedDigest[i] != digest[i])
                throw new Exception("Digest mismatch. Integrity of the message
compromised");
    }
```

The method first calculates the digest of the input message from the input parameter. Then it checks if the size of the digests is equal, if the sizes mismatched, the method throws an exception and indicate the error. Follow by it, the method compares the values of each byte of the calculated digest to the input digest values from the method parameter. If any of the byte value is different, the method throws an exception with a message indicating the digests are mismatched and the integrity of the message is compromised.

5.2.4. Base64 Encoding

The character encoding for the secure SMS message is encoded using the Base64 encoding algorithm. It converts the data bytes of the message into 7bits characters. This is needed due to the library used for the bank server receiver (GSM modem) cannot accept bytes values that are between the ranges of -97 to -128. Therefore it is necessary to encode the data message into character format.

After encoding the message using Base64 algorithm, the message size is increased by 3/2 ratio. Therefore the initial data size needs to be reduced to accommodate for the sending message so it can fit within a single SMS format. The initial design of the *Transaction Payload* field of the message structure was 93 bytes. After the decision of using Base64 encoding, it was necessary to reduce the size of the *Transaction Payload* field to 54 bytes.

5.2.5. Sending SMS Message

Before the message is send, the content must be processed such as encrypting the message content and calculating the message digest. The content of the SMS message is a concatenation of the version number, the account identifier, the sequence number, the ciphered content and the message digest. The protocol specified that the PIN, the transaction type and the transaction payload must be encrypted. Therefore the mobile application encrypts these transaction contents into a ciphered content. The protocol also specified that the secure message must have a message digest to ensure message integrity.

After the message content is created, the content is converted to Base64 characters string. Since the mobile application is sending pure text message, this message can be sent using 7bits per character text message format. Therefore a single mobile banking SMS message can hold 160 characters.

To send SMS messages with J2ME, it requires using the WMA library. The WMA library has classes to handle SMS messages. It uses the MessageConnection class to initialize a connection to send out SMS messages. The format of the destination address is "sms://<phone number>:<port>", where <phone number> is the string of the destination phone number and <port> is the port number which the receiver is listening on. If the message is not set to send to any port, then the message is delivered to the receiver as a normal SMS message. The receiver stores the received message into its message inbox.

Before the mobile application sends an SMS, the application starts a new thread to handle the sending of the message. This is necessary because if the sending of the message fails, the process can still continue with the main process.

5.2.6. Receiving SMS Message

To implement the mobile application to listen for incoming messages, the mobile application must listen to a specified port. This port number is a 16 bits number and it can be any number chosen by the programmer. The port number used for private applications can be chosen from the range 49152-65635 and the range 0-49151 is used by privileged applications[11]. If the port is already used by other installed application then the later installed application would not be able to function properly[12]. For this project we have selected 54321 as the listening port number.

The mobile application must implement the MessageListener class that comes standard with J2ME development kit to receive incoming messages with the mobile application. To implement the MessageListener, the programmer must implement the `void notifyIncomingMessage(MessageConnection conn)` method as well. This method is automatically called by the application when the listening port receives a message. The method call `conn.receive()` is used to retrieve the received message from the message connector.

After the message is received, the application automatically launches a new form to request for the account identifier and the PIN to decrypt the received message. If the correct details are entered, the application can decrypt the message and then checks for message integrity. If both security tests pass, then the received message is displayed on the mobile screen.

[11] For the list of privileged port numbers, go to http://www.iana.org/assignments/port-numbers
[12] It is possible to reserve a port number for an application by contacting the Internet Assigned Number Authority (IANA).

5.3. The Bank Server Application

Since the Mobile Banking application and the Bank Server application shared some of the implemented classes. Therefore it is necessary to use Java™ to implement the bank server application. Similarly the Database Server application also shares some implemented classes with the bank server application therefore it was decided that both the bank server application and the Backend Database application are implemented using Java™ technology. The Bank Server application is built using Java™ 2 Standard Edition development kit. The application is deployed on a computer running with Intel® Pentium® Xeon processor using Microsoft® Windows® Server 2003 operating system.

A Metacom-Siemens TC65 GSM modem is used to receive SMS messages from the GSM network. The library used to operate the GSM modem is SMSLib[13] version 1.3.4 for Java™.

5.3.1. Receiving SMS Message

The Bank Server has an infinite loop to constantly listen for incoming messages. When a message arrives at the GSM modem, the server application will process the received message.

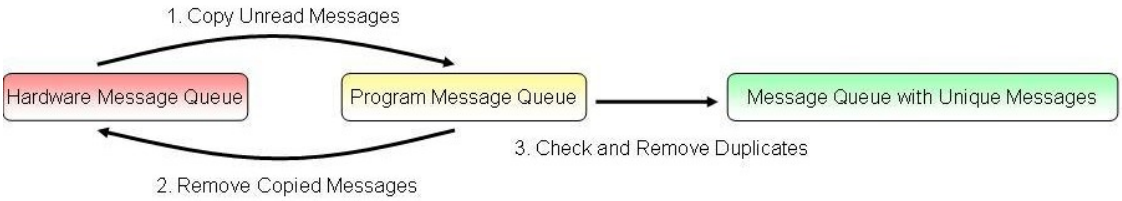


Figure 17. Bank Server Removes Duplicated Messages from the Message Queue

Figure 17 illustrates the process on how the server removes duplicate messages from the message queue. When the bank server receives an SMS message, the GSM modem places the SMS message in the hardware message queue. The Bank Server application retrieves the message by checking the message buffer queue to find if there exist any unread messages. If unread messages are found, the bank server retrieves the list of all the unread messages from the buffer queue to the program memory and empties the hardware message queue from the GSM modem. The SMSLib library used for handling SMS messages creates duplicate of unread messages. Which means for every message in the memory message list, a duplicate of that message is existed within the list. It is necessary to neglect the duplicates. Therefore the application checks if any duplicate existed in the list. If a duplicate is found, it is removed from the message list.

The Electronic Communications and Transactions Act of South Africa require all electronic transactions to be retained persistently for a period of time. Therefore after the server removes the message duplicates, it saves the received message into the database. This helps the server to check if the received message has been received before. This can prevent replay message attacks.

[13] Available From http://smslib.sourceforge.net/

After the message is stored in the database, the message is converted from characters string to data bytes using Base64 decoder. Then the content is retrieved from the decoded message. The content is passed onto the `Reply.getReply()` method to process for security verifications for the received message.

5.3.2. Communicating with the Backend Database

It is assume that the bank server has a secure communication with the backend database. For this project we used a Secure Socket Layer (SSL) over Remote Method Invocation (RMI). The SSL/RMI is implemented with the DBCall class as the remote object. The Reply class has methods that will invoke the DBCall remote object to retrieve values from the backend database.

More detailed explanation of the SSL/RMI communication is explained in the "5.4.1.SSL/RMI communication" section.

5.3.3. Security Verifications

The security verification of the SMS message follows the secure SMS protocol specified in "4.5.Protocol Sequences" section. The Bank Server receives the message and passes the message content to the reply class to verify for security.

The Bank Server first uses the message handler class to break the message content bytes into message structure. The Bank Server retrieves the message version bytes from the message handler and checks if the bytes values match the chosen version bytes values of the application. The developer can select any bytes values that they want for their application. The version bytes values chosen for this project implementation is "SMB". If the versions mismatched, the server will stop processing the message and it will with an error message. If the version bytes mismatched, it is assumed that the received SMS message was sent by the originator accidentally. Therefore no reply message will be sent to the message sender. This approach helps the Bank Server to reduce the wasting of unnecessary cost.

The following security implementations are implemented in the Reply class. This class is used for verifying the security of the message and generates a reply message for the message sender.

If the version bytes are matched, the server application retrieves the account identifier from the message and checks if the received account identifier exists in the database. If it does not exist in the database, the server application will return a plaintext SMS message to the message originator to inform the account identifier does not exist.

If the account identifier does exist in the database, the application retrieves the transaction sequence number from the received message. The application compares the retrieved sequence number with the sequence number for the account identifier from the database. If the sequence number mismatched, the server application replies an error message to inform the message sender that the sequence number mismatched.

Followed by the sequence number verification, the bank server retrieves the account identifier's current one-time password from the backend database. If the server application cannot retrieve the password, then the server replies an error message to inform the message sender that an error has occurred. If the server successfully retrieves the password, it will fill the retrieved password with space characters to make it the length of 16 characters. The padding is necessary because AES algorithm requires the key to be in the length of 128bits (16bytes), therefore it is necessary to fill the password to the standard AES key length.

After the password retrieval, the server application uses this password to decrypt the message. If the decryption fails, the codec class will throw an exception indicating that the password cannot decrypt the message content. The exception is caught and the error message is replied to the message sender. If the decryption is successful, the server will remove the used one-time password from the database and increment the account's sequence number counter by 1. Therefore once the password has successfully decrypted a message, the password will be discarded.

The decrypted content is the banking details of the account holder. The decrypted bytes are passed onto the Banking Detail class. The Banking Detail class breaks down the decrypted content into fields which are specified in "4.6.Message Structure" section. These fields are encapsulated within the Banking Detail class object. Therefore the server application can call methods of Banking Detail object to retrieve the different field's values.

After message decryption, the bank server will check if the message integrity has been comprised. The integrity check is perform by the server using the calculated message digest of the received message compares with the message digest read from the received message. If these digests mismatch, it indicates the message integrity is compromised and the server will reply an error message to the message sender.

The next security check is the account identifier authentication. The server application reads the PIN from the decrypted message and compares it with the PIN stored in the database for the given account identifier. If the PINs mismatched, then it indicates the authentication failed and the server will reply an error message to the message sender.

If all of the above security verifications passed, the server has confident that the received message is the reliable from an authentic user.
After authentication, the server examines the type of transaction that the bank should perform from the received message. It could either be checking for account balance, transfer money or purchase airtime. The transaction reply message is encrypted using a 16 characters password. The password for encryption is the concatenation of the account identifier and the account PIN. This password combination is a secret because only the user is suppose to know his/her own PIN. Therefore reply message confidentiality is achieved. To ensure the reply message integrity, a digest of the reply message is attached with the reply. Therefore the receiver can calculate a digest to test for the message

integrity. Before sending the transaction confirmation reply message, the encrypt content is encoded in Base64 format. Then it is sent as a 7bits text characters SMS message.

5.3.4. Reply Message Error Codes

If the reply message is an error message, then there will be "Error#" at the beginning of the reply message text, where the # indicates the error code of the message. The error code depends on the error type produced from the security checks. The following list is the definition of the error codes.

Security Error Code:

1 Message version bytes mismatched
2 Account identifier does not exist
3 Message sequence number mismatched with the sequence number from the database
4 Server cannot decode the received message using database one-time password
5 Message integrity check failed. Message content compromised
6 User PIN authentications failed

System Internal Error Code:

7 Server cannot retrieve password
8 IOException: Cannot read received message

Transaction Error Code:

9 System cannot determine the value for amount
10 Cannot perform money transfer
11 From account is the same as destination account
12 Cannot purchase airtime
13 Transaction type is not recognisable
14 Reply message encryption error

5.3.5. Replying Confirmation Message

As mentioned in the previous section, if the version bytes in the received message are not the expected bytes values, then this reply message is discarded and it will not be sent.

If the reply message is an error message then the reply will be sent as a plaintext SMS message to the user's mobile phone's SMS inbox. If the error is not a security error, instead a banking transaction error (reply message contains Transaction Error Code), then the message is encrypted using the account identifier and PIN. This message is sent to the receiver's listening port, for this project the listening port is 54321. Sending the SMS message to a specified port, the message becomes a push message. When the destination mobile phone receives the message, the message will cause the mobile phone to automatically launch the mobile application. The application will prompt the users for security details to decrypt the received message.

5.4. The Backend Database Application

5.4.1. SSL/RMI Communication

Before create an SSL/RMI communication channel, it is necessary to have all the security related tools. These are the server certificate, and the server key store and the client trust store. The reader can refer to the Web page written by Viswanath in [17] for a full tutorial on how to create the above mentioned tools and to create an SSL/RMI channel using Java™.

In the SSL/RMI communication, the bank server is the client end of the channel and the backend database is the server end of the channel.

When the bank server connects to the backend database, the SSL communication requires the bank server to have the client trust store file and the database server to have the server key store file. These two files are used for the negotiation of the SSL channel. Once the channel is negotiated, the communication between the server and the database is confidential.

To create an RMI connection between two remote units, we will need to build a remote object which is used by both parties. The remote object is built as an interface object and the methods are implemented on the database server. This remote object is the DBCall class.

After the SSL communication is set, the bank server can invoke the remote methods by calling methods from the remote object. This allows the bank server to call the database application methods remotely and retrieves values from it over a secure line.

5.4.2. JDBC Connection

To write Java™ program to communicate with the database, the program needs to have a Java Database Connectivity (JDBC) connection with the database. The choice of database for this project is MySQL® database. Therefore we need to use the MySQL® Java™ library[14].

Before executing any database statement, the program needs to set up a connection with the database. Therefore the URL of the database is needed. The format of the JDBC URI is "jdbc:mysql://<database URL>" where the database URL is the location of the database. The following code snippet shows how to use Java™ to set up a JDBC connection.

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
//connects to specified URL's database
con = DriverManager.getConnection(jURL, username, password);
stm = con.createStatement();
```

The first line of the above code initializes a new instance of the JDBC driver. Then the program uses the DriverManager to create a connection to the given database URL. The

[14] Available From: http://dev.mysql.com/downloads/connector/j/3.1.html

user name and password is to verify to the database server application. The last line of the code, it creates a statement object from the connection. The statement object is use for executing SQL statements.

5.4.3. MySQL® Database

The SQL database schemas are as follow:

Account(AccID, Name, Balance, PIN, CusPhoneNum)
OTP(AccID, Seq, Password)
PhoneAccounts(PhoneNum, Balance)
DeletedOTP(AccID, Seq, Password)
ReceivedMessage(Message, Time, Originator)

The *Account* table is to store the registered account holder's information. The *OTP* table is the one-time passwords table; it stores all the one-time passwords for the registered users. The *PhoneAccounts* table stores the registered phone accounts. The *DeletedOTP* table stores all the used one-time passwords. The South African Electronic Communication and Transaction Act require all received transaction to be store persistently for a period of time. Therefore the *ReceivedMessage* table is use to store all the received messages of the bank server.

5.5. The Passwords Generator Application

The Passwords Generator application was built using Java™ 2 Standard Edition (J2SE) development kit running with NetBeans 5.0 as the integrated development environment.

The passwords generator application is only used by the bank. This application is use for generating one-time passwords for the mobile bank account holders.

5.5.1. User Interface

Figure 18 illustrates the user interface of the passwords generator. The user must first connect to the database server. When the user selects the option "List AccID", a list of account identifiers is displayed in the left hand side list box of the user interface. Then the user can select the wanted account identifier and chose whether to generate a new passwords list or to get the passwords list for the selected account identifier. The get passwords option is for the user to retrieve the list of passwords from database and the list gets displayed in the right hand side list box of the user interface. The program ask the user if the user wants the list to be outputted to a file, if the user selects yes the passwords list will be outputted as an HTML page.

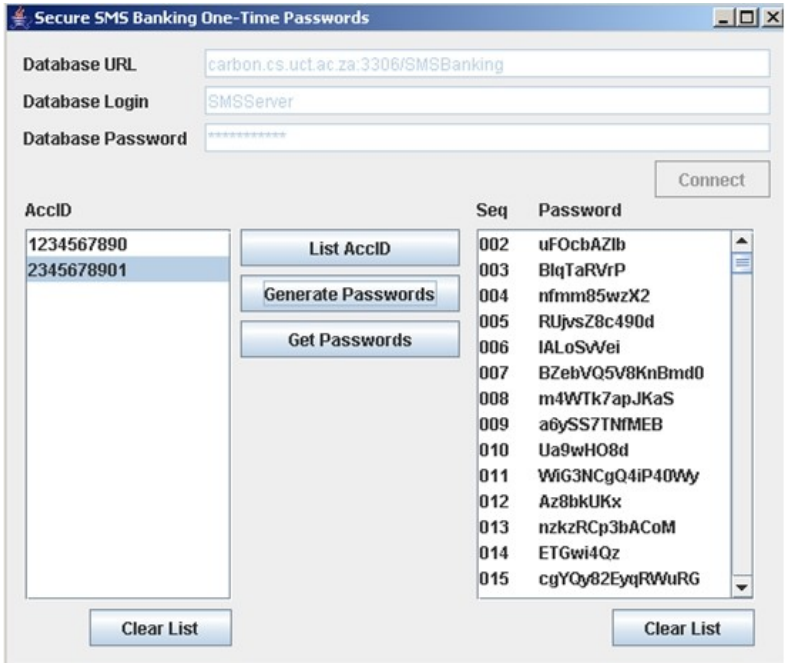


Figure 18. Passwords Generator User Interface

5.5.2. Generating Secure Random One-Time Passwords

The strength of the symmetric key encryption depends on the strength of the key generation algorithm. It means that if the generated key is predicable then the key generation algorithm is weak.

The generated key must be formed by characters where the users can enter using their mobile phone. Therefore the key generation algorithm must generate random characters such that it is enterable by using a mobile phone and it must be non-predictable.

The algorithm we used for the key generation for this project is base on secure random number generation. The program initially has a set of allow characters for the passwords. Then the program generates a random number as the index number to selects a character from the allowed character set and inserts this selected character into the generating password. The final length of the generated password is between eight to sixteen characters long. The length of the passwords is also randomly generated.

The character selections for the generating password require a secure random number generator. The secure random number generator chosen for this project is SHA1 pseudo random number generator. This algorithm generates random numbers that are non-predicable because the SHA1 algorithm is a one way hashing function. Therefore there is no inverse function so the number generation steps cannot be predicted.

The random number generator requires a seed value to start generating random numbers. The generator takes advantage of time program execution time to be random and uses the time value as the seed value. Therefore the seed value is not predictable so the generated numbers are random.

6. Final Testing

Various tests were performed to identify errors and to ensure the application can run correctly. There are two parts of testing for this project. The first part is to test and ensure the designed protocol is secure. The second part is to test the implementation is correct. Usability was considered during the design and the implementation of the project but as the main concern of this project is security and cost efficiency therefore no actual user testing was done.

The main goal of this project is to design and implement a secure SMS channel between the mobile phone and the bank server. Therefore the test cases are targeting at the security of the designed solution.

The mobile phone application test cases are done on Nokia 6020 mobile phone.

6.1. Protocol Testing

6.1.1. Confidentiality – AES

The level of security of the designed protocol depends on the strength of the encryption algorithm used. The type of algorithm needed for this project is symmetric key encryption algorithm. The algorithm must perform encryption relatively fast in the mobile phone environment. Therefore the AES algorithms were considered to perform the message encryption.

NSA[15] has conducted a review and analysis of using AES to protect classified information. AES is an NSA approved cryptographic algorithm to be used for United States' national security information and system at all classification levels [19]. The use of 128 bits key length is approved to be sufficient to protect classified information up to the US national secret level.

The speed of encrypting a message depends on the size of the original message. Since the designed protocol uses SMS to transmit secure details, therefore the upper bound of the size of the original message cannot exceed the size of an SMS message which is 160 characters.

To test the speed of encrypting an SMS message, a timer was included in the implemented mobile application for testing purpose. The program stores the current time of the mobile phone into memory before the encryption begins. When the encryption finishes, the program also stores the time into memory. Therefore the time to encrypt the message is the finish-time minus the begin-time. Several test cases were performed and it shows that to encrypt an SMS message using a mobile phone (Nokia 6020) takes less than 200 milliseconds, indicating that it is relatively fast.

[15] National Security Agency (NSA) is a United States agency within the US Department of Defense responsible for signals intelligence, encryption, cryptography, and related topics.

The second testing for confidentiality is that the encrypted message can be decrypted. Since we are using symmetric key encryption, we must be able use the encryption key to decrypt the message. We used the AES implementation from the Bouncy Castle crypto library. We used black box testing to test the library's implementation. We first encrypted the message, and then we used invalid keys that are similar to the valid encryption key to decrypt the message. The results shown that the library's method returns exception if the wrong key is used. When the valid key is used to decrypt the message, the original message is returned.

6.1.2. Integrity – SHA1

A message digest is use to maintain the message integrity for each SMS message. The speed of calculating the message must be efficient and it must calculate the message digest relatively fast in the mobile phone environment. The message digest calculation algorithm used for this project is SHA1.

The SHA1 produces a 160 bits output called the message digest. The SHA1 is considered to be secure because it is computationally infeasible to find a message which corresponds to a given message digest or to find two different messages which produce the same message digest [20]. Any change to the transferring message, there is very high probability that the output message digest will be different and the receiver can notice the message content is compromised.

A timer was also used to test the execution speed of the SHA1 algorithm. The results show that on average for the mobile phone to calculate the message digest using SHA1 takes less then 150 milliseconds.

A second test case was performed to ensure the alteration of the original message will produce a different message digest. This was done using the J2ME simulator running on a computer. The original message's digest bytes were compare to the altered message digest bytes. We tested by changing a single character in the message to prove if a small change in the original message can produce a different message digest. The results shown that the message digests are completely different.

6.1.3. Authentication

The authentication detail (PIN) of the user is protected within the encrypted banking details. The attacker cannot read the authentication detail of the user therefore the attacker cannot use the authentication detail to perform masquerading attacks.

6.1.4. Non-repudiation

The design of the pairs of One-Time Passwords and Sequence Numbers made the user cannot refute of sending the message to the bank server. This was explained in "4.7.4.Non-Repudiation" section. To test for non-repudiation, it requires testing for the passwords generator, which is explained later in "6.2.4.One-Time Passwords Generation" section.

6.1.5. Replay Attacks

Assuming the attacker managed to get hold of the transmitting message and he/she performs replay attacks. For every received message, the bank server needs to check for the sequence number for the specific account identifier given in the received message. If the message's sequence number does not match the sequence number from the bank database, then the message is discarded.

To further enhance replay attacks prevention, the bank server stores every received message into the database[16]. When a new message is received, the bank server can check it against those messages that are stored in the database.

To test for replay attacks, we deliberately send the same messages to the bank server multiple times. The server received the first message and performed the transaction, when the next identical message is received; the message is ignored and discarded because of the received message is already exist in the database.

6.1.6. Masquerading Attacks

Masquerading attack is when the attacker pretends to be a legitimate account user. It is assume that the attacker managed to get hold of the legitimate account identifier. The attacker cannot perform banking transaction because he does not have the account identifier's PIN. We further assume the attacker managed to get hold of the user's PIN. The attacker still cannot perform bank transaction because the attacker does not have the required One-Time Password to correctly encrypt the banking details for the bank to interpret.

To test for masquerading attacks, we created a message with a valid account identifier. We then used an invalid One-Time Password to encrypt the banking message. When the message gets delivered to the bank server, the server cannot decrypt the message using the database password, therefore the message decryption failed. Since the message cannot be decrypted by the bank server, the bank server does not have to check for the message PIN because it cannot be read.

6.1.7. Brute Force Attack

If the user gets hold of the transmitting message, it is possible for the attacker to crack the cipher message to read the security details; therefore secure algorithms are needed to ensure message contents are protected. The strength of the banking details protection depends on the strength of the encryption algorithm and the strength of encryption keys generation.

If we assume the attacker uses brute force attack to try crack the message content, then the following calculation shows how much time the attacker needs to crack the message.

[16] South African Electronic Communications and Transactions (ECT) Act, 2002 Chapter III, Part 1, 16.Retention, has stated that all received electronic transactions should be stored in persistent storage.

We assume the keys are purely random. The following sets of characters are allowed to be in the generated key:

[a-z] size = 26, [A-Z] size = 26, [0-9] size = 10

Therefore the possible characters' size is 26+26+10 = *62*.

We assume that the attacker uses an advance computer to perform the attack and it can process 10^6 decryptions per second.

The key size is between 8 to 16 characters (any keys that are less than 16 characters are padded with space characters). Therefore the attacker has to try $62^8 + 62^9 + 62^{10} + 62^{11} + 62^{12} + 62^{13} + 62^{14} + 62^{15} + 62^{16} = 4.845 \times 10^{28}$.

And we further assume on average it would take the attacker half the number of tries to before he/she cracks the message content. Therefore after 2.422×10^{28} tries the attacker can read the message content.

The average number of years that requires the attacker using brute force attack to crack the encrypted message would be

$$\frac{2.422 \times 10^{28}}{10^6 \times 60 \times 60 \times 24 \times 365.25} \text{ years} = \mathbf{7.677 \times 10^{14}} \text{ years}$$

The above calculation shown that on average, it is computationally infeasible for the attacker to use brute force attack to crack the encrypted message.

The use of One-Time Password provides extra security for the protocol. If we assume the attacker managed to gets hold of the user's account PIN, and the attack knows the sequence number of the user. The attacker still requires the One-Time Password of the user to encrypt the message for the bank to perform the attacker's transaction.

6.2. Implementation Testing

6.2.1. Mobile Banking Application

The first test case for the mobile banking application is to test the restriction implemented for the user interface. NetBeans provides the options to build input boxes to be restricted to certain types of inputs. The bank server phone number can only contain phone numbers; certain fields cannot be longer than the expected characters length; the sequence number can only contain digits and the amount can only be a valid non-negative decimal number. After testing, it shown that the restrictions implemented in the user interface has passed the test case.

To test for protocol implementation correctness, we test for how a secure SMS message is generated and sent. We performed a white box testing to analysis each step of the designed protocol implemented in the mobile banking application. Then we performed a black box testing to see if the SMS message can be sent using the mobile application.

To receive SMS messages from the bank server, we performed a black box testing to test if the mobile application is listening to the given application port. To test if the application is listening for incoming message, we used the J2ME WMA simulator to send SMS messages to a specific port. The result shown when a message is sent to the

specified port, the message is received by the mobile phone. And if the message is sent to a non-listening port, the message is delivered to the mobile phone as a normal SMS message.

6.2.2. Bank Server

A test case was needed to ensure the bank server is listening for incoming messages. For the bank server to handle SMS messages, we used the open source SMSLib. A black box testing approach was done for this test case. The results show that when the bank server receives a message, it is read into the program memory. Due to the implementation of the library, the received message is duplicated in the message queue. Therefore a decision was made to remove the duplicated messages from the memory queue this was discussed in the "5.3.1.Receiving SMS Message" section. The SMSLib library cannot read the message bytes from -97 to -128; therefore a decision of needing the secure SMS message to be encoded into Based64 format was made.

The second test case for the bank server is testing for the processing of the received messages. The bank server must first decode the message using Base64 decoding and then follows the secure SMS protocol. Tests were performed for each of the scenarios below:

- Read message from memory message queue, use Base64 algorithm to decode the received message. This was tested separately from the protocol. We fed in an encoded message and used the decoding method to decode the message to see if the original message returns.
- Check for message version: If the received message version bytes are incorrect, the server ignores the received message.
- Check for account identifier existence: If the account identifier does not exist in the database, the server replies a message to the message originator informing the given account identifier does not exist.
- Check for sequence synchronization: If the sequence numbers mismatch, the server reply error message.
- Decrypt message: if the server cannot decrypt the received message, it replies an error message.
- Check message digest: if the message digest indicates the message content is compromised, the server replies error message.
- Authenticate user: if the given PIN cannot authenticate the user, the server replies an error message.
- Performing Transaction: if the bank cannot perform the given transaction, the server replies an error message. If the bank server successfully performed the transaction, the server sends encrypted reply message to the message sender.

The third test case is to test if the bank server is connected to the backend database. If the connection cannot be established, an exception is thrown by the application.

6.2.3. Backend Database

The first test case for the database is to test if it can connect to the server. The backend database is the data server; therefore it always listens for bank server's request for connection. If a bank server is received, but the server cannot connect to the database, then an exception is thrown to indicate an error occurred.

The second test case is to test if the application can retrieve values from the database. This test case requires testing for the database language SQL statements are corrects. Then it is necessary to check if the returned value is the correct requested value from database.

The third test case is to test if the application can update or insert values into the database. This requires the testing of the SQL statement and the values are stored in the correct attributes of the database schema.

6.2.4. One-Time Passwords Generation

The most important test case for this application is the testing of the application generating unique passwords. If a duplicated password is generated, the application indicates an error. To test this, we retrieved a password from the database and tried to insert the retrieved password into the database. The database application indicates that it cannot insert the value because the database already contains that password.

Table 3. Test Cases Results

Mobile Banking Application	
User interface restriction	Passed
Protocol correctness	Passed
Receiving SMS messages from a listening port	Passed
Bank Server	
Receiving SMS messages	Duplicated messages received by the library. SMSLib cannot read bytes with the value of -97 to -128.
Process received message	Passed
Database connection	Passed
Backend Database	
Allow server to connect to the database	Passed
Retrieve value from database	Passed. If the wanted value does not exist, an exception is thrown.
Updating database	Passed. If the inserted value is not in the correct format, an exception is thrown.
One-Time Passwords Generation	
Generating unique password pairs	Passed

7. Validations and Results

This section explains the validation the hypotheses described in the "3.4.Hypotheses" section.

7.1. Secure SMS Protocol

The first two hypotheses to be validated in this experiment are the following:

- A secure SMS messaging protocol can be designed and implemented that offers better security.
- The designed protocol can be integrated with mobile banking to form secure SMS banking.

Background research in the "2.Theory / Background" section revealed that there are many security problems with the current GSM architecture and its SMS protocol. The main security concern is that if the user wants to send sensitive data using SMS, the message can be intersected. A third person from the cellular service provider with authority can read or alter the content of the message. Therefore the security of the message is compromised.

In "4.Design" section, we provided a designed secure SMS protocol. The protocol ensures the transmitting SMS message is secure and the message can be delivered to the destination securely. The designed protocol conforms to the principles of security service.

In "4.Design" and "5.Implementation / Integration" sections, we had software designed and integrated the secure SMS protocol into a secure mobile banking solution. The implemented applications follow the secure SMS protocol to transfer and receive mobile banking messages. The mobile banking solution was tested and it shows that transaction can be performed in real time.

Currently there is no standard protocol for secure SMS transmission. In this project we designed a protocol which covers the security shortfalls of the standard SMS protocol and the protocol was designed to follow the principle of security service. The designed protocol was proven that it can be integrated with a mobile banking solution. Therefore our first two hypotheses can be accepted.

7.2. Comparing Secure SMS Banking with current Mobile Banking Solutions

There are many different types of media to send banking transactions to the bank server using mobile phones. In this section, we illustrate the properties of each type of mobile banking.

Currently in South Africa, the company called WIZZIT uses pure USSD messages to let their customers to perform mobile banking.

There are two methods of SMS banking adopted by banks in South Africa. The first method is to use USSD to authenticate the user and to announce to the bank server to prepare for incoming banking transaction. Then the user sends a plaintext SMS message to the bank server to inform the type of banking transaction to perform. This approach is adopted by First National Bank in South Africa. The second method of SMS banking is to use WIG to transfer banking SMS messages. Both methods are considered in this comparison section.

Many banks in South Africa have adopted using GPRS channel to transmit banking transaction messages.

The Secure SMS Banking is the designed protocol for SMS banking and implementation was developed for this project.

The following table illustrates the comparison of different types of mobile banking.

Table 4. Comparison of Secure SMS Banking with current mobile banking Solutions

	USSD Banking	**Current SMS Banking**		**GPRS Banking**	**Secure SMS Banking**
		USSD+SMS	**WIG+SMS**		
Cost	Most cellular operators offer USSD service for free. The user only pays for the banking services.	Transferring USSD string is for free. This solution requires one SMS message to be sent for each transaction.	Requires transmitting many SMS messages between the server and the mobile phone for a transaction.	Using GPRS is charged by the amount of data transferred. Therefore the cost depends on how much data is needed to transmit a transaction.	Requires transmitting a single SMS to the bank server to perform mobile banking.
Security	USSD string is sent to the bank server as plaintext. User authentication relies on the SIM card's IMEI.	Both USSD and SMS are transmitted as plaintext messages to the bank server. User authentication relies on the SIM card's IMSI.	WIG has built in security tools. It supports end-to-end security. User authentication relies on the SIM card's IMSI.	Standard Bank uses 128 bits encryption to transmit GPRS banking transaction. But it did not mention the encryption algorithm used.	AES 128 bits One-Time Password encryption. SHA1 message digest. Uses sequence number to prevent replay attacks. It has user Authentication.
Transmission Speed	The speed of all the mobile banking solutions depends on the cellular network provider. It also depends on the strength of signal received by the user's mobile phone. Therefore it depends on the location of the user, the traffic of the network, the number of base towers in the area around the user's mobile and etc. All these factors can influence the speed of transmission, thus no actual experiment can be conducted.				

Reliability	Synchronous communication. Immediate response.	USSD is synchronous. User gets immediate response from the bank server.	SMS through WIG channel is asynchronous. The WIG channel will wait for the server to reply. SMS messages are stored in the message buffer queue at the service provider until it is delivered.	Synchronous connection. Can lose connection if the signal is weak. If the connection is lost then it requires rebuilding the connection to the bank server.	SMS messages are stored in the message buffer queue at the service provider until it is delivered.
Compatibility	Any mobile phone that can support USSD can use such mobile banking solution.	Any mobile phone that can support USSD and SMS can use this service.	Requires mobile phone to be SIM Application Toolkit (SAT) compatible. It is SIM card dependent.	Requires mobile phone to be WAP capable and GPRS, EDGE or 3G enabled.	Depends on the implementation of the protocol. For this project, it requires the mobile phone to be Java™ compatible.
Usability	Requires no menu. User enters the USSD number as dialling phone numbers.	Requires no menu. The user interface depends on how the user sends SMS messages using his/her mobile phone.	Menu based user interface.	Menu based user interface or mobile phone internet browser interface.	J2ME application user interface.

From the table above, we can judge that there are advantages of using secure SMS to perform banking transaction. The following explain some of the advantages of using secure SMS banking:

- SMS is delivered in asynchronous mode. When a message is submitted for sending, it is stored in the message buffer queue until the message is delivered. Therefore it is more reliable because it can guarantee message delivery.
- Secure SMS banking uses well researched security algorithms to ensure the transmitting message is protected.
- The cost of sending is relatively cheap. The protocol was designed to use minimum number of SMS to send the banking transaction.

We have shown that the designed Secure SMS Banking solution has security advantages over the current mobile banking solutions. Therefore we can accept the third hypothesis.

8. Conclusion

The designed protocol solution provides all the necessary security tools to perform a protected mobile banking transaction. The designed protocol provides an end-to-end security communication solution from the mobile phone to the bank server. The solution has achieved all the success criteria. Therefore we can claim the overall project was successful.

There is one minor security crisis with the designed protocol. The user's account identifier and sequence number are sent as plaintext format. Therefore the protocol does not hide the user's identity. This security concern was initially considered, but it was decided to be neglected. This is for the reason that, to successfully hide the user's identity it will require the mobile phone to negotiate for an encrypted session with the bank server. It requires the mobile phone to transfer public keys with the bank server to perform asymmetric encryption. Such approach requires the mobile phone to communicate with the bank server multiple times for handshaking before sending actual banking details. Since one of the requirement for the project is to provide a cost effective solution. Therefore the decision of neglecting this problem was made. It is assumed that the GSM architecture provides sufficient encryption strength to prevent non-cellular operator from reading the user's identity. However, the cellular operators do have full access to the identity of the user's account identifier but they do not have all the necessary details to masquerade the user to perform a mobile banking transaction. The solution to over come this issue is discussed in the "9.Future Work / Maintenance" section.

The first primary goal of this project is to discover all the security shortfalls with SMS banking. The second goal is to design and implement a solution to fix all the discovered security shortfalls with SMS banking. Both of these goals are achieved and it was successful to use the new solution for SMS banking.

Although the project is successful but the security of SMS banking is limited. This is due to the fact that SMS messages are sent asynchronously therefore there is no guarantee of real time response from the bank server. Secondly the size of an SMS message is 160 7bits characters therefore the uses of security tools are limited within the size of an SMS message. An alternative solution to this problem is to use Multimedia Message Service (MMS) messages instead of SMS, but this would require a new design of the protocol solution.

9. Future Work / Maintenance

9.1. Application Distribution

In the "4.4.Design Assumptions" section, we had assumed the user had the mobile banking application pre-installed on their mobile phone. In reality, we need to consider how the mobile application is distributed to the users. This is crucial because the attacker can create a similar user interface program to pretend to be the legit application. When the account holder uses the attacker's application; the user enters his banking details and the fraud application sends the user's information to the attacker's receiver. Therefore the security details are breached and the attacker can use the victim's banking details to perform banking transaction.

These are some of the solutions to prevent such attack:

- The first option is the bank can provide users with the mobile banking application when the customer registers for a mobile banking account. The bank can upload the application onto the customer's mobile phone via Bluetooth technology. This approach can ensure the user has the correct and legit mobile application.
- The second option is the bank can provide the mobile banking application on a digitally signed CD. The customer can use his/her personal computer to verify the digital signature and certificate that are stored inside the CD. Once the CD is verified, the user can trust the application is come from the legit distributor, then the user can deploy the application onto his/her mobile phone.
- The third option is the bank can use a third trusted authority to digitally certify the mobile banking application. The user's mobile phone should be setup to trust the same third party authority. If both the user and the bank server have trust the third party, the bank server can distribute their certified mobile application on the Web. The user can download the certified mobile application via WAP or GPRS. This type of installation is called the Over-the-Air (OTA) installation. When the user finish downloading the application, the mobile phone can verify if the downloaded application is certified by a trusted authority.

9.2. Secure SMS Messaging Solution

Many people use SMS to send sensitive data without knowing the security vulnerability behind the GSM network. For example, some people use plaintext SMS messages to distribute passwords. This is not secure because the service provider keeps the record of all transmitting message in their database. Therefore a secure SMS messaging solution is needed.

The designed protocol for secure SMS does not necessarily limits to be use for mobile banking application. It can easily be adapt to support secure SMS messaging solutions for peer-to-peer communication.

To adapt this project's secure SMS protocol for secure peer-to-peer messaging, it requires some slight changes with the protocol. The main security principles for secure peer-to-peer messaging is to keep the message content confidential and to maintain the message integrity. The principles of non-repudiation and authentication are optional; it depends on the requirement of the user's need and they can reduce usability.

The secure peer-to-peer messaging application requires the receiver and the sender to have the list of generated One-Time Passwords. The use of One-Time Passwords can be optional, as the users can agree on the same password to decrypt the message. If the idea of One-Time Passwords is not used, then the sequence number can be optional. The sequence number must be incremented at both sides when the message is successively delivered. When the receiver accepts an incoming message, the receiver application should reply a message delivered respond to the sender to notify the message was delivered successfully.

The encrypted banking details of the message structure can be use to store the encrypted message content. The calculation of the message digest uses the same message structure fields for algorithm's input.

9.3. Concealing User Identity

As mentioned in the "8.Conclusion" section, the user identity is not concealed. In this section we will discuss the approach to conceal the user's identity when sending a secure SMS message to the bank server.

To conceal the user's identity, it is necessary to use asymmetric encryption algorithm to hide the user's identity. This requires the negotiation of the bank server's public key. The mobile application will first need to send a request to the bank server for retrieving the server's public key. When the server receives the request, the server will reply its public key to the message originator. After the mobile application receives the server's public key, the mobile application uses the server's public key to encrypt the user's account identifier. Using this approach, only the bank server can decrypt the user's account identifier.

To avoid the handshaking of the bank server's public key, the key can be hard-coded into the application. This can avoid the mobile application to ask for server's public key, which uses less SMS messages. But the problem is, when the bank server changes its public key, the user needs to download the application with the correct server's public key.

The problems of using the asymmetric encryption to hide the user's account identifier is, the size of the output of the encrypted account identifier gets increased enormously compare to the size of the plaintext account identifier. Therefore many bytes are wasted

for the encryption payload. The encrypted message size will lead to the use of more than one SMS message to send the secure SMS from the mobile application. This approach is very cost inefficient.

References

[1] First National Bank Cellphone Banking Website. *Cellphone Banking – How Do I*. Available from: https://www.fnb.co.za/personal/transact/accessyouraccounts/cellHowDoI.html (2006); accessed 27 October 2006.

[2] Eberspächer, J., Vögel, H.-J., Bettstetter, C. *GSM Switching, Services and Protocols 2nd Edition*. John Wiley & Sons Inc., West Sussex, 1999.

[3] Lord, S. Trouble at Telco: When GSM Goes Bad. In *Network Security*, 2003(1):10 – 12, 2003.

[4] Margrave, D. *GSM Security and Encryption*. Available from: http://www.hackcanada.com/blackcrawl/cell/gsm/gsm-secur/gsm-secur.html (1999); accessed 27 October 2006.

[5] Biryukov, A. Shamir, A. Wagner, D. Real Time Cryptanalysis of A5/1 on a PC. In *Fast Software Encryption Workshop*, 2000

[6] *GSM calls even more secure thanks to new A5/3 Algorithm*. Available from: http://www.cellular.co.za/news_2002/101702-gsm_calls_even_more_secure_thank.htm (2002); accessed 27 October 2006.

[7] Wagner, D. *GSM Cloning*. Smartcard Developer Association and ISAAC security research group. Available from: http://www.isaac.cs.berkeley.edu/isaac/gsm.html (1998); accessed 28 October 2006.

[8] Ratshinanga, S., Lo, J., Bishop, J. A Security Mechanism for Secure SMS Communication. In *Proceedings of SAICSIT*. Stellenbosch, Western Cape, 2004, pp1 – 6.

[9] Roberts, P. *Nokia Phones Vulnerable to DoS Attack*. Available from: http://www.infoworld.com/article/03/02/26/HNnokiados_1.html?hardware (2003); accessed 28 October 2006.

[10] Enck, W., Traynor, P., McDaniel, P., La Porta, T. Exploiting open functionality in SMS-capable cellular networks. In *Proceedings of the 12th ACM conference on Computer and Communications Security*. Alexandria, VA, USA, 2005, pp. 393 – 404.

[11] Hassinen, M., Markovski, S. Secure SMS messaging using Quasigroup encryption and Java SMS API. *SPLST'03*, Kuopio, Finland June 17 – 18, 2003.

[12] Mallat, N., Matti, R., Tuunainen, V. K. *Mobile Banking Services*. *Communications of the ACM*, 47(5):42 – 46, 2004.

[13] Cobbett, J. *Feisty Challenger to Mzansi, MTN*. Available from: http://www.moneyweb.co.za/news/more_moneyweb/493633.htm (2005); accessed 28 October 2006

[14] Ashoka. *WIZZIT Bank – Low Cost banking for the unbanked*. Available from http://www.changemakers.net/journal/300508/displayfec.cfm?ID=101 (2003); accessed 28 October 2006.

[15] MTN. *MTN: Mobile Banking*. Available from: http://www.mtn.co.za/?pid=219666 (2006); accessed 28 October 2006.

[16] Forum Nokia. *A Brief Introduction to Secure SMS Messaging in MIDP Version 1.0*. Available from http://sw.nokia.com/id/5274b81c-12d0-43bb-8d89-26f6a1ae111f/A_Brief_Introduction_to_Secure_SMS_Messaging_in_MIDP_en.pdf (2003); accessed 29 October 2006.

[17] Viswanath, K. *The New RMI*. Available from: http://today.java.net/pub/a/today/2005/10/06/the-new-rmi.html (2005); accessed 29 October 2006.

[18] Stallings, W. *Network Security Essentials Applications and Standards, international second ed*. Prentice Hall, 2003.

[19] Committee on National Security System. CNSS Policy No. 15, Fact Sheet No. 1, National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information. Available from: http://www.cnss.gov/Assets/pdf/cnssp_15_fs.pdf (2003); accessed 02 November 2006.

[20] Eastlake, D. Jones, P. *US Secure Hash Algorithm (SHA1)*. Available from: http://www.ietf.org/rfc/rfc3174.txt (2001); accessed 02 November 2006.

Appendix A: Acronyms

AES	Advance Encryption Standard
AUC	Authentication Centre
BSC	Base Station Controller
BTS	Base Transceiver Station
EDGE	Enhanced Data rates for Global Evolution
EIR	Equipment Identity Register
GMSC	Gateway MSC
GPRS	General Packet Radio Service
GSM	Global System for Mobile communication
HLR	Home Location Register
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
ISC	International Switching Centre
MMS	Multimedia Message Service
MS	Mobile Station
MSC	Mobile Switch Controller
NSA	National Security Agency
OMC	Operation and Maintenance Controller
PIN	Person Identification Number
SHA1	Secure Hash Algorithm 1
SIM	Subscriber Identity Module
SMS	Short Message Service
SMSC	SMS Centre
TMSI	Temporary Mobile Subscriber Identity
USSD	Unstructured Supplementary Services Data
VLR	Visitor Location Register
WIG	Wireless Internet Gateway