NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SHORT MESSAGE SERVICE (SMS) SECURITY SOLUTION FOR MOBILE DEVICES

by

Yu Loon Ng

December 2006

Thesis Advisor:	Gurminder Singh
Co-Advisor:	John Gibson

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December/2006		**3. REPORT TYPE AND DATES COVERED** Master's Thesis
4. TITLE AND SUBTITLE Short Message Service (SMS) Security Solution for Mobile Devices			**5. FUNDING NUMBERS**
6. AUTHOR(S) Yu Loon Ng			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

This thesis focuses on the security of Short Message Service (SMS) and the Global System for Mobile communication (GSM) network and the use of encryption to protect SMS messages. A detailed study of the GSM network and SMS protocol, and encryption schemes was conducted to understand the properties of different encryption schemes and their applicability to SMS messages. An experiment was conducted to measure the actual performance of various encryption schemes on a modern smart phone device. An analysis of the encryption scheme properties and the performance measurement was then conducted to select a suitable scheme for SMS encryption. The selected scheme was implemented in the form of a Secure SMS Chat application to validate the viability of the selected encryption scheme. Potential applications of secure SMS in military settings are also discussed.

14. SUBJECT TERMS GSM security, SMS security, mobile device security, encryption, mobile device performance analysis, secure chat			**15. NUMBER OF PAGES** 113
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

SHORT MESSAGE SERVICE (SMS) SECURITY SOLUTION FOR MOBILE DEVICES

Yu Loon Ng
Major, Singapore Navy
B.Eng., Nanyang Technological University, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2006

Author:	Yu Loon Ng
Approved by:	Gurminder Singh Thesis Advisor
	John Gibson Co-Advisor
	Peter Denning Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis focuses on the security of Short Message Service (SMS) and the Global System for Mobile communication (GSM) network, and the use of encryption to protect SMS messages. A detailed study of the GSM network, the SMS protocol and various encryption schemes was conducted to understand the properties of different encryption schemes and their applicability to SMS messages. An experiment was conducted to measure the actual performance of various encryption schemes on a modern smart phone. An analysis of the encryption scheme properties and the performance measurement was then conducted to select a suitable scheme for SMS encryption. The selected scheme was implemented in the form of a Secure SMS Chat application to validate the viability of the selected encryption scheme. Potential applications of secure SMS in military settings are also discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my thesis advisors, Dr. Gurminder Singh and Mr. John Gibson, for their patience and constant guidance.

I would also like to express my gratitude to my family for their continuous support and understanding.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Short Message Service (SMS) is a text message service that enables users to send short messages to other users on the Global System for Mobile communication (GSM) network. SMS uses a store-and-forward mechanism similar to SMTP mail service. Instead of mail servers, SMS Centers (SMSC) are used to store the SMS messages before they are forwarded to the mobile user's service provider or another SMSC. Although the network connections between the SMSC and nodes in a GSM network are usually protected by Virtual Private Network (VPN) tunnels, the SMS messages are stored unencrypted at the SMSC. This means that employees of SMSC operators, or others who can hack into the system, can view all the SMS messages passing through the SMSC. Many SMSCs also retain a copy of the SMS messages for audit, billing and dispute resolution purposes [1]. If an attacker manages to compromise the SMSC, the attacker can also read the SMS traffic. One of the more high profile victims of such an attack in recent years was England football captain David Beckham, whose SMS exchange with his personal assistant Rebecca Loos was intercepted and published in a tabloid [2]. Two employees from European phone operator mmO2 were dismissed for helping their friend obtain copies of his girlfriend's SMS messages [3].

B. STATEMENT OF PROBLEM

Encryption provides a means of protecting sensitive communications over a public network but it imposes overhead in terms of additional computing. Mobile devices are generally faced with constraints on computational power and battery life. These constraints impose limits on the amount of encryption operations that can be performed without seriously affecting the usability of the device. Therefore, symmetric encryption is commonly used in mobile devices because of its efficiency relative to asymmetric encryption, such as PKI. That is why most current commercial SMS encryption solutions use password-based symmetric encryption. Passwords are used as a key distribution mechanism to synchronize the encryption keys. However, the use of passwords reduces the strength of the cipher to the strength of the password when open algorithms, such as Data Encryption Standard (DES) or Advanced Encryption Standard (AES), are used. The onus is on the user to select a strong password.

Although asymmetric encryption offers the additional advantage of simple key distribution and strong encryption, asymmetric encryption is not used because it is computationally demanding.

However, mobile devices have experienced dramatic improvements in computing speeds and memory capacity, matching those of desktop computers a few years ago. Advances have also been made in battery technology and the energy efficiency of components, thereby extending the operating life of mobile devices. Given these developments, it remains to be shown whether or not modern devices are still limited in their ability to harness the advantages of asymmetric encryption to secure messages like SMS.

C. SCOPE OF RESEARCH

This thesis focuses on the use of encryption to secure SMS messages. The encryption requirements of voice traffic and other data traffic will not be discussed. The characteristics of different encryption schemes and their performance on a modern mobile device are presented. The properties of SMS were assessed with respect to their impact on encryption selection. Based on the measurement results, a suitable encryption scheme for SMS is selected, and deployed. A typical application is used to validate the selection.

D. RESEARCH OBJECTIVES

1. Primary Research Question

The primary research objective is to compare the performances of different encryption schemes on a modern mobile device and determine a suitable scheme, or combination of schemes, for protecting SMS messages. The aim is to determine if there are better ways of protecting SMS messages than just using symmetric encryption, thereby alleviating the constraints imposed on encryption use by the difficulties of symmetric key management.

2. Subsidiary Research Questions

Based on actual measurements, it will be possible to determine the overhead, such as power consumption, timing, and transmission, associated with encryption operations for different schemes. With this information, it may be possible to devise combinations of encryption schemes to meet different security requirements for different applications.

E. RESEARCH METHODOLOGY

The research comprises of two major areas, namely the comparison of different encryption schemes and the identification and deployment of a selected scheme.

The available encryption schemes will be studied and compared based on their security properties and characteristics through literature research. The properties of SMS will be studied in detail to understand its characteristics and to determine security requirements. The comparison of the performance of different encryption schemes will be done using results from actual measurements. An experiment will be set up to measure the power consumption, timing overhead and transmission overheads associated with encryption. Conclusions can then be drawn taking into consideration these factors.

The demonstration application selected to validate the choice of encryption is a “Secure Chat” application based on SMS. The application was chosen because it has practical applications and is demanding in terms of real time user interactivity.

F. THESIS ORGANIZATION

The remainder of the thesis is organized as follows. Chapter II provides an overview of the security issues surrounding SMS, from the GSM infrastructure to the mobile device. This Chapter also highlights some applications where the security of the SMS is of paramount importance if it is to be used as a delivery mechanism in the application.

Chapter III discusses the issues surrounding the use of encryption in mobile devices. It highlights some of the key considerations when choosing a suitable encryption scheme. The results of an experiment to measure some of the performance metrics are also discussed in this Chapter.

Chapter IV describes the Secure Chat demonstration application that was developed to show how SMS messages can be secured using encryption.

Chapter V concludes the thesis and provides recommendations for further research in this area.

II. OVERVIEW

A. OVERVIEW

The GSM network has grown rapidly since its introduction in the early 1990s, with the second billionth GSM user connected in Q2 2006 [4]. The number of SMS messages sent has also seen explosive growth, with an estimated one trillion SMS messages sent globally in 2005 [5]. With the vast amount of information transacted using SMS, it is important the SMS text messages be adequately protected against eavesdropping and modification.

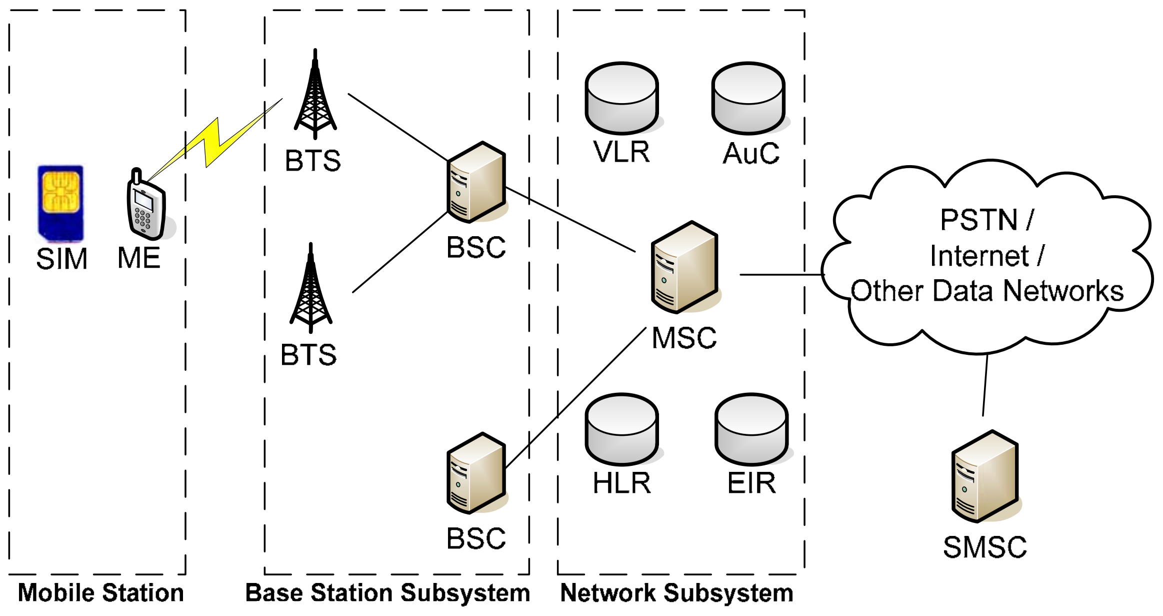
A discussion on SMS security is especially challenging because SMS messages transverse across different transmission media, undergo multiple protocol translations, and are processed by different devices operated by different organizations. As such, the overall security of SMS will be only as strong as the weakest link in the whole chain.

This Chapter discusses the security of SMS in three parts: at the GSM infrastructure level, at the SMS application layer, and at the mobile device. The last section of the Chapter discusses other potential uses of SMS if its security can be assured.

B. GSM SECURITY

1. GSM Technology

The standards for GSM are governed by the European Telecommunications Standards Institute (ETSI) [6]. A typical GSM system is comprised of three subsystems: the Mobile Station, the Base Station Subsystem and the Network Subsystem. Figure 1 provides an overview of a typical GSM network with the key components and the SMS Center (SMSC).



AuC – Authentication Center
BSC – Base Switching Center
BTS – Base Transceiver Station
EIR – Equipment Identity Register
HLR – Home Location Record
ME – Mobile Equipment

MSC – Mobile Switching Center
PSTN – Public Switched Telephone Network
SIM – Subscriber Identity Module
SMSC – Short Message Service Center
VLR – Visitor Location Record

Figure 1. Overview of the GSM Network

The Mobile Station is the mobile component of the GSM system and includes the Mobile Equipment (ME) and the Subscriber Identity Module (SIM).

The Base Station Subsystem includes Base Transceiver Stations (BTS) and Base Switching Centers (BSC). On one end, the Base Station Subsystem interfaces with the Mobile Station and manages the mobility of the Mobile Equipment across different Base Transceiver Stations. On the other end, the Base Station Subsystem interfaces with the Network Subsystem to connect to the external networks and other services.

The Network Subsystem is the core of the GSM system and provides functionalities such as call connections, management of subscribers, mobility, and interfaces with the other networks such as Public Switched Telephone

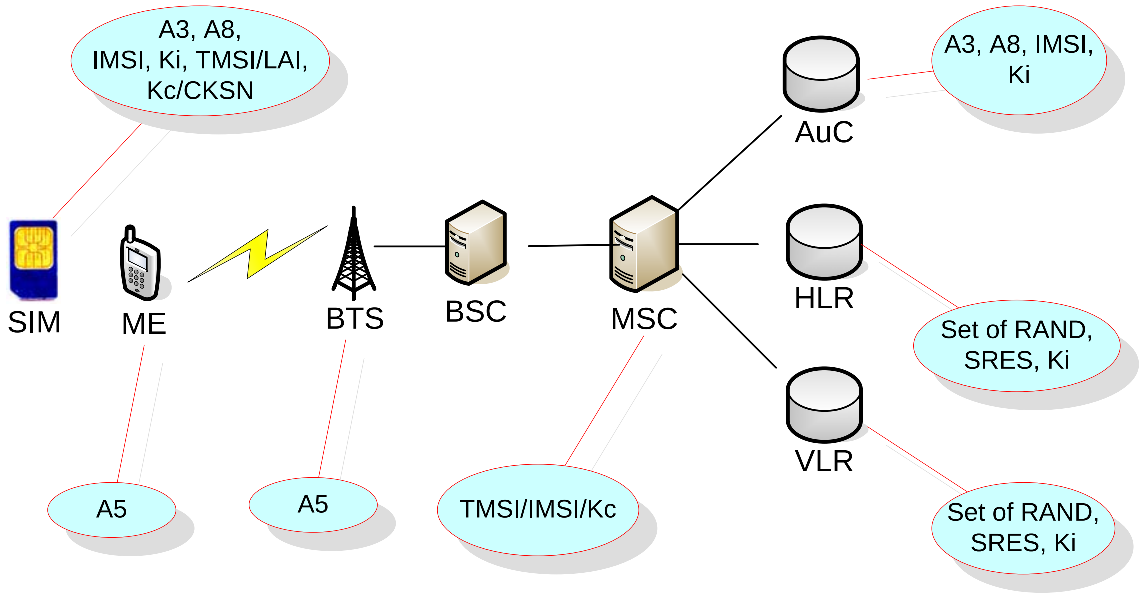
Network (PSTN), Internet and other data networks. These functionalities are implemented through the core components: Mobile Switching Center (MSC), Home Location Record (HLR), Visitor Location Record (VLR), Authentication Center (AuC) and Equipment Identity Register (EIR).

2. GSM Security Features

From the initial conception, GSM was designed with security in mind. However, the primary motivations were to eliminate cellular fraud, which was prevalent in analog cellular systems, and to protect communications against interception over the air [7]. The security aspects of GSM are described in *ETSI GSM 02.09* [8] and the four basic security services that were expected to be provided were subscriber anonymity, authentication, signaling data and voice protection against eavesdropping, and identification of user and mobile equipment [9]. In order to achieve these security objectives, several security components were required:

- Authentication Algorithm (A3)
- Authentication Center (AuC)
- Ciphering Algorithm (A5)
- Ciphering Key Generating Algorithm (A8)
- Ciphering Key Sequence Number (CKSN)
- Ciphering Key (Kc)
- International Mobile Subscriber Identity (IMSI)
- Individual Subscriber Authentication Key (Ki)
- Location Area Identity (LAI)
- Random Number (RAND)
- Signed Response (SRES)

These components are implemented in three different system elements; the Subscriber Identity Module (SIM) [10], the GSM network [11] and the GSM handset. Figure 2 shows the distribution of these components in the GSM network.



A3 - Authentication Algorithm	Kc - Ciphering Key
A5 - Ciphering Algorithm	Ki - Individual Subscriber Authentication Key
A8 - Ciphering Key Generating Algorithm	LAI - Location Area Identity
AuC – Authentication Center	ME – Mobile Equipment
BTS – Base Transceiver Station	MSC – Mobile Switching Center
BSC – Base Switching Center	RAND - Random Number
CKSN - Ciphering Key Sequence Number	SRES - Signed Response
HLR – Home Location Record	SIM – Subscriber Identity Module
IMSI - International Mobile Subscriber Identity	VLR – Visitor Location Record

Figure 2. Security Components in a GSM Network (After Ref. [12])

The application of these components to achieve the security objectives are described in detail in the following Subsections.

3. GSM Authentication

The GSM network uses a challenge-response mechanism for authentication [12]. Figure 3 shows the authentication process.

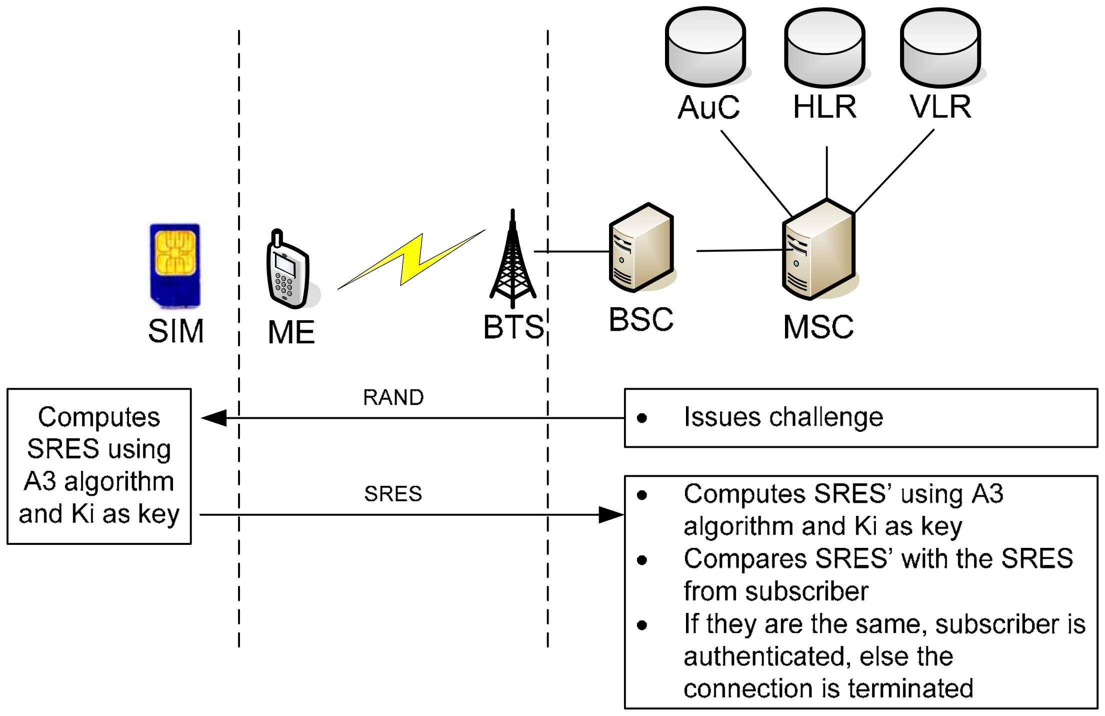

AuC
HLR
VLR
SIM
ME
BTS
BSC
MSC
RAND
SRES
Computes SRES using A3 algorithm and Ki as key
Issues challenge
Computes SRES' using A3 algorithm and Ki as key
Compares SRES' with the SRES from subscriber
If they are the same, subscriber is authenticated, else the connection is terminated

Figure 3. GSM Authentication Mechanism

A 128-bit random number (RAND) is generated by the HLR and sent to the Mobile Station (MS). The MS encrypts the RAND by using the authentication algorithm (A3) and the individual subscriber authentication key (Ki). The output is a 32-bit signed response (SRES) that is sent back to the network. Upon receiving the signed response (SRES) from the subscriber, the GSM network repeats the same computation to produce SRES'. If SRES and SRES' are the same, the identity of the subscriber is authenticated. If SRES and SRES' do not match, the connection is terminated and an authentication failure message is sent to the MS.

Throughout the entire authentication process, the individual subscriber authentication key (Ki) is never transmitted over the radio channel. Ki is only present in the SIM, AuC, HLR, and VLR. The calculation of the signed response is processed within the SIM to protect confidential subscriber information such as the IMSI or Ki.

4. Data Confidentiality

Data confidentiality is achieved through the use of the key generation algorithm (A8) and the encrypting algorithm A5. Figure 4 illustrates the encryption process.

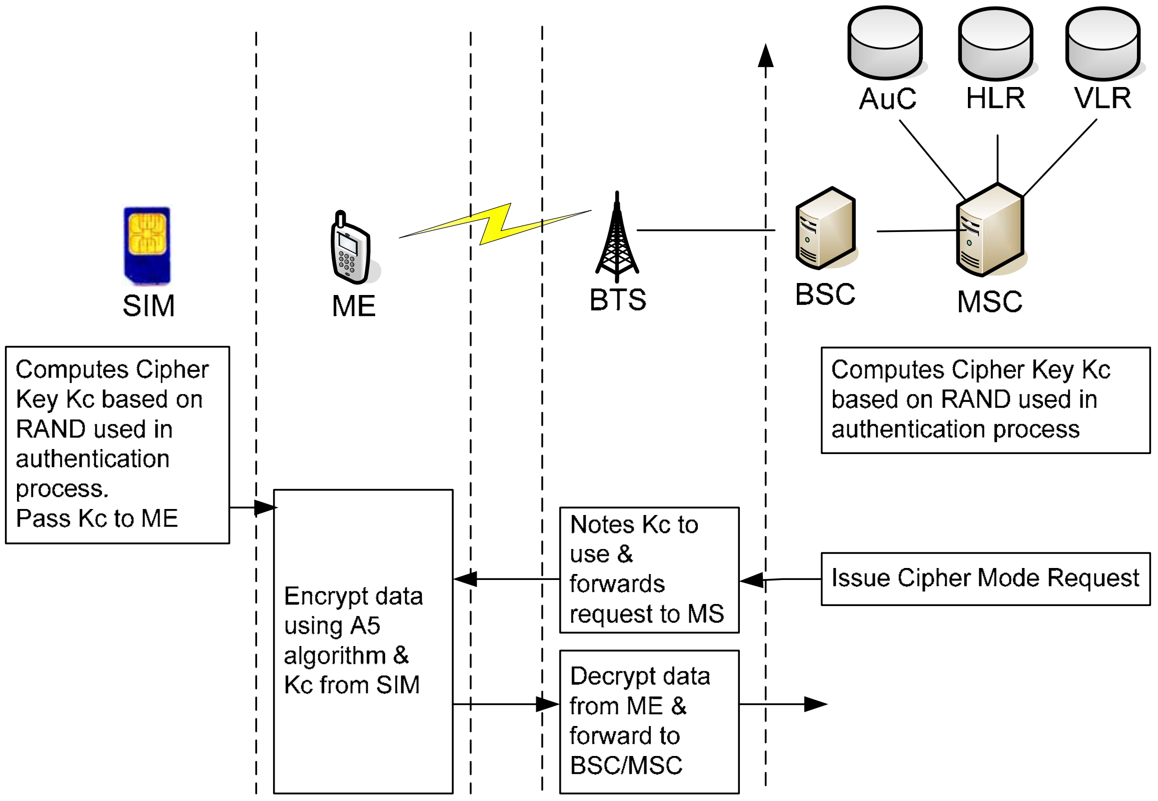

AuC
HLR
VLR
SIM
ME
BTS
BSC
MSC
Computes Cipher Key Kc based on RAND used in authentication process. Pass Kc to ME
Computes Cipher Key Kc based on RAND used in authentication process
Encrypt data using A5 algorithm & Kc from SIM
Notes Kc to use & forwards request to MS
Issue Cipher Mode Request
Decrypt data from ME & forward to BSC/MSC

Figure 4. GSM Encryption Mechanism

The SIM uses the random number (RAND) used in the authentication process and the individual subscriber key (Ki) to generate the 64-bit cipher key (Kc) based on the key generating algorithm (A8). Once generated, the cipher key is used as the key to the A5 algorithm for subsequent encryption of data between the Mobile Equipment (ME) and the Base Transceiver Station (BTS). On the network end, the Kc is generated in the same manner by the HLR and passed to the BTS. After the authentication process, a cipher mode request is sent to the ME to decide on the cipher to use. Once the cipher is agreed upon, all subsequent radio traffic between the ME and the BTS is encrypted using Kc. The same Kc is used for the entire session of communication. The GSM standard allows for regular key change through re-authentication of the ME for added security. However, this is not implemented for many systems. As a result, the same Kc may be used for days. Similar to the authentication process, the computation of the ciphering key (Kc) takes place within the SIM. Therefore, the individual subscriber authentication key (Ki) does not leave the SIM.

5. Subscriber Identity Confidentiality

The confidentiality of the subscriber identity (IMSI) is achieved through the use of the Temporary Mobile Subscriber Identity (TMSI). When the ME is first switched on in a new MSC/VLR area, the real identity (IMSI) is used and a TMSI is assigned by the network to the ME. Thereafter, the TMSI is used for all subsequent communications between the ME and the GSM network. Both the IMSI and TMSI are stored in the SIM. Figure 5 shows the TMSI allocation process.

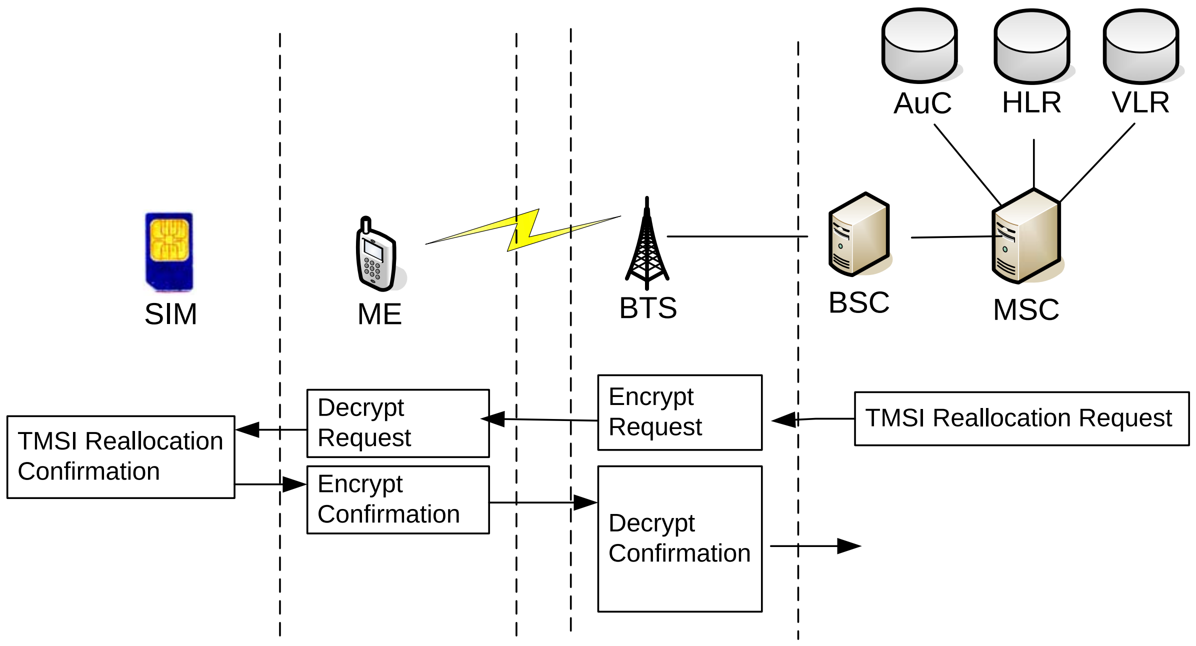


Figure 5. TMSI Reallocation

After the authentication and encryption process is complete, the TMSI is sent to the MS. The MS responds by confirming reception of the TMSI. The TMSI is valid in the location area in which it was issued. To support roaming of subscribers to other networks, the Location Area Identification (LAI) is used in addition to the TMSI to determine the location and identity of the subscriber.

6. SIM Security

Although the Subscriber Identity Module (SIM) physically resides with the Mobile Equipment (ME), it is regarded as an important part of the GSM infrastructure because it is the piece of hardware that represents the subscriber. As described in the previous sections, the SIM houses many of the security components for the GSM Network. All authentication operations take place within the SIM and none of the keys or ciphers leaves the SIM. The SIM may also be protected with a Personal Identification Number (PIN). SMS messages are also stored in the SIM. Generally, the SIM is considered a piece of tamper-proof hardware. Although hacks against smart cards are available, extraction of information directly from the card is generally difficult and it requires physical access to the card and specialized equipment. It is easier to make a clone of the card by making use of information on the key generating algorithm. The following Section describes this vulnerability in greater detail.

7. GSM Network Vulnerabilities

Several vulnerabilities in the GSM network have been exposed over the past years. Most of them involve the breaking of the encryption algorithms used: A3, A5 and A8. These encryption algorithms were originally developed in secrecy and were not subjected to public review [13]. Subsequently, when the codes for the algorithms were leaked or crypto-analyzed, vulnerabilities were found in these algorithms or in their implementations [14].

The A3 and A8 algorithms were mainly broken because most GSM providers use the COMP128 algorithm to implement A3 and A8. COMP128 is a hash algorithm that takes a 128-bit key (in this case Ki) and a 128-bit input (in this case the random number challenge issued by the HLR) and produces a 96-bit output. The first 32 bits are used as the signed response (SRES) and the remaining 64 bits is used as input for the A5 algorithm. Once the 128-bit key for COMP128 can be derived, the SIM card can be cloned. If the SIM card can be cloned, the entire GSM authentication mechanism falls apart because the GSM network can no longer differentiated between the different users. The most recent attack on COMP128 used a partitioning attack and reduced the attack time to less than a minute [15]. This means that an attacker only needs a minute of physical access time to derive the key and clone the SIM. Over-the-air cloning was accessed to be technically feasible by building a fake base station at a cost of about US$10K [14]. For the determined attacker, this is certainly achievable.

The A5 encryption algorithm is a stream cipher that protects the over-the-air transmission between the ME and the BTS. The A5 algorithms are available in different versions:

- A5/0 utilizes no encryption.
- A5/1 is the original A5 algorithm used in Europe.
- A5/2 is a weaker encryption algorithm created for export and used in the countries outside Europe
- A5/3 is a strong encryption algorithm that is created as part of the 3rd Generation Partnership Project (3GPP) for the 3G systems.

Attacks against the A5 algorithm have been published as early as 1997. In 2003, a group of researchers from Israel published practical attacks on the stronger A5/1 algorithm that could be carried out in real-time [17]. This showed that the GSM network can no longer be relied on to provide confidentiality of information even on the radio links. The GSM standards do not impose security requirements for land line connections. Therefore, the implementation of any form of encryption on the land lines is left up to the telecommunications operators.

The GSM network can be subjected to Denial of Service attacks using electronic jammers. Since the GSM operating frequencies are known, generating a stronger radio signal to overwhelm the BTS and MS is trivial. However, a recent paper published by Pennsylvania State University described how a remote Denial of Service attack can be conducted on a GSM network by using SMS [18]. The idea was to flood the control channel of a particular GSM cell with SMS messages. When the control channel is overwhelmed, call establishments and roaming are severely impacted in the targeted cell.

8. SMS Center (SMSC) Security

The SMSC is often considered an integral part of the GSM network. However, with the rapid growth in SMS applications, many independent SMSC operators have sprouted in the industry. They lease connections from the telecommunications service provider and provide services such as SMS advertising, news broadcasts, chats, etc. In terms of security, this has huge implications.

a. Policy Enforcement

Originally, the GSM network could be considered a relatively closed network with connections only to other telecommunications operators. The telecommunications operator owns the infrastructure, including the radio links. Thus, a unified security policy could be applied across the entire network, assuming that the operator has a minimal set of security policies that make a difference. Any security breaches from employees in the network could be investigated easily. With a connection to a third party SMSC, the trust is essentially extended to the SMSC. However, the telecommunications operator does not own the SMSC. Therefore, there is no way of ensuring that the same level of security can be enforced at the SMSC. Even though the connection may be secured with a Virtual Private Network, the host security of the SMSC cannot be determined.

b. Host Security

Although SMSC provides a specialized service, the applications are usually hosted on platforms that run general purpose Operating Systems, like Unix or Windows. These Operating Systems have their own set of security vulnerabilities and require regular patching. Security mechanisms, such as access control, physical security, policy enforcement, and security administration, need to be in place to ensure the security of SMSC.

c. *Network Security*

The SMSC usually rides on the Internet infrastructure for cost reasons and to tap into the huge number of Internet users. By connecting to the Internet, the SMSC has essentially bridged the GSM network with the Internet and introduced the vulnerabilities and threats of the Internet to the GSM network. One can argue that many GSM operators already support General Packet Radio Service (GPRS), which is also connected to the Internet. However, the key difference is that the GSM operator owns the GPRS infrastructure. Therefore, the telecommunications operator can decide on what the defensive mechanisms are required to enforce the security policy. However, the GSM operator cannot mandate what mechanisms the SMSC must have in order to be connected to it.

C. SMS APPLICATION LAYER SECURITY

1. SMS Protocol

The Short Message Service (SMS) was created as part of the GSM Phase 1 standard. Each short message is up to 160 characters in length when Latin alphabets are used and 70 characters in length when non-Latin alphabets, such as Arabic and Chinese, are used [19].

SMS is a store and forward service. In other words, SMS messages are not sent directly from sender to recipient, but always via an SMS Center (SMSC). Each mobile telephone network that supports SMS has one or more messaging centers to handle and manage the short messages. Some of the features of SMS that have led to the popularity of SMS are [20]:

- SMS supports confirmation of message delivery. The sender of the message can choose to receive a return message back to indicate whether the SMS has been delivered or not.
- SMS can be sent and received simultaneously with other traffic. SMS uses the control channel as a transport mechanism, unlike voice, data and fax calls which use dedicated radio channels for the duration of the call.
- SMS compression and concatenation have been defined and incorporated into the GSM SMS standards. As such, the original 160 character limitation can be overcome.

- SMS is not bandwidth intensive. This allows telecommunications service providers to offer attractive pricing plans, which includes free SMS messages. Packages with 900 free SMS messages are offered for under USD20 in some service plans in Singapore [21].

Besides the technological properties, the attractive social aspect of short text messaging has also contributed to the success of SMS. Text messaging is non-intrusive and discreet, and is particularly suitable in certain social settings like meetings or social gatherings. Therefore, SMS has become the primary mode of communications for many. Besides the casual exchange of information among friends, the use of SMS has also expanded to other industries such as gaming, banking, education, remote sensor monitoring, advertising, voting, etc. Further potential applications using secure SMS are discussed in last Section of this Chapter.

2. SMS Security Specifications

The technical specifications for SMS and SIM are described in ETSI TS 03.48. The intent was to spell out the specifications required to achieve end-to-end security between Mobile Stations and SMS Centers. However, all the specifications did was to define additional fields that could be used in the user-defined portion of the SMS Transfer Protocol Data Unit (TPDU) to describe the security properties that the SMS will have.

The SMS application server or the SIM can set the first byte of the User Data Header to a value of Ox70 to indicate that the User Data Header will be followed by a Command Header, which in turns describes the security parameters used to secure the data. The first two bytes of the Command Header denote the total length of the Command Header and the Secured User Data. The next byte is the length of the rest of the Command Header. Figure 6 shows the SMS_SUBMIT TPDU structure when the security headers are used.

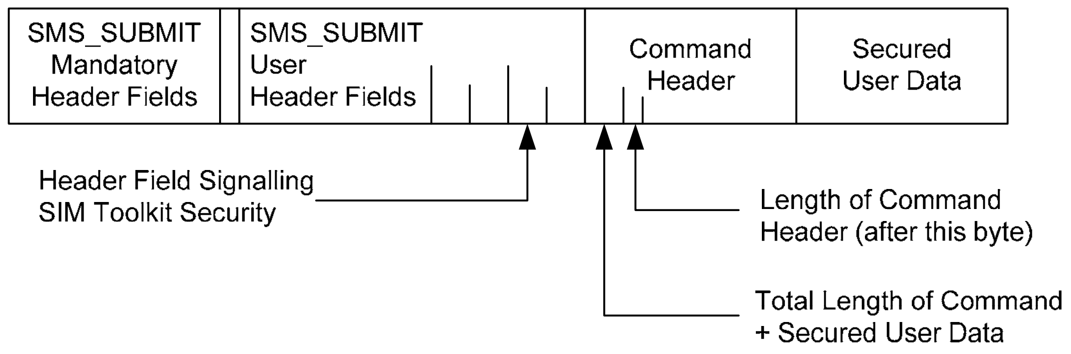
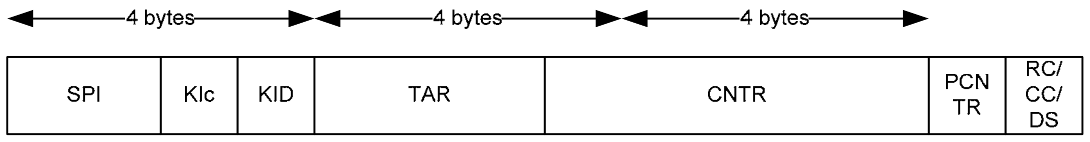

SMS_SUBMIT Mandatory Header Fields
SMS_SUBMIT User Header Fields
Command Header
Secured User Data
Header Field Signalling SIM Toolkit Security
Length of Command Header (after this byte)
Total Length of Command + Secured User Data

Figure 6. SMS_SUBMIT TPDU with Security Headers (After Ref. [19])

The Command Header essentially describes how the user data is being encrypted. The Command Header consists of seven fields as follows:

- Security Parameter Index (SPI)
- Ciphering Key Identifier (KIc)
- Key Identifier (KID)
- Toolkit Application Reference (TAR)
- Padding Counter (PCNTR)
- Integrity Value (RC/CC/DS)

Figure 7 is a graphical representation of the Command Header.



SPI – Security Parameter Index (2 bytes)
Kic – Ciphering Key Identifier (1 byte)
KID – Key Identifier (1 byte)
TAR – Toolkit Application Reference (3 bytes)

CNTR – Counter (5 bytes)
PCNTR – Padding Counter (1 byte)
RCC/CC/DS – Integrity Value (variable)

Figure 7. Structure of Command Header

The SPI is a collection of flags used to describe the security parameters. This provides the recipient with sufficient information to undo the sequence of operations to recover the data. The byte value coding for the SPI is shown in Figures 8 below, where PoR refers to Proof of Receipt and RE is the Receiving Entity, who will create the PoR.

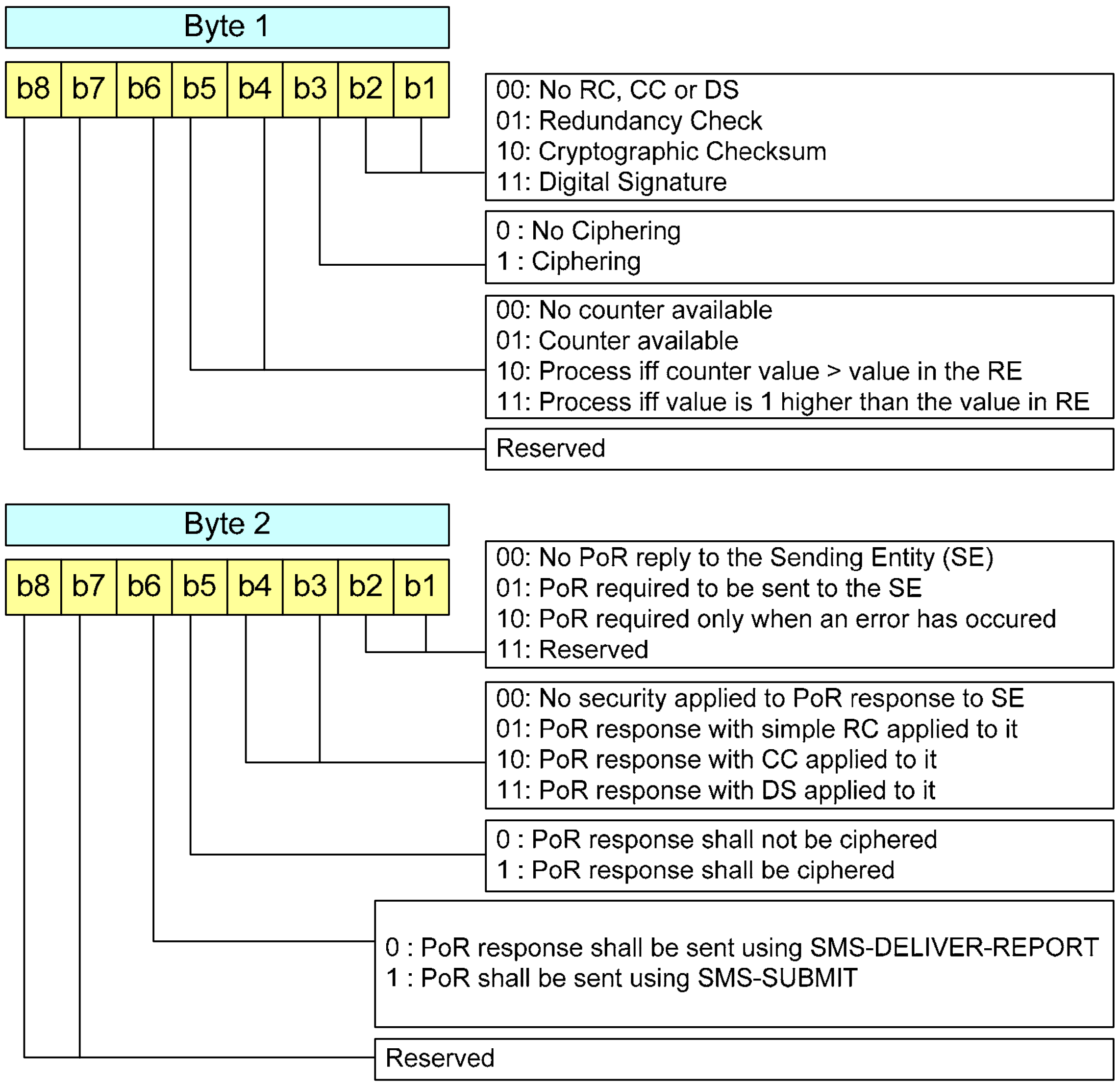

Byte 1
b8
b7
b6
b5
b4
b3
b2
b1
00: No RC, CC or DS
01: Redundancy Check
10: Cryptographic Checksum
11: Digital Signature
0 : No Ciphering
1 : Ciphering
00: No counter available
01: Counter available
10: Process iff counter value > value in the RE
11: Process iff value is 1 higher than the value in RE
Reserved
Byte 2
b8
b7
b6
b5
b4
b3
b2
b1
00: No PoR reply to the Sending Entity (SE)
01: PoR required to be sent to the SE
10: PoR required only when an error has occured
11: Reserved
00: No security applied to PoR response to SE
01: PoR response with simple RC applied to it
10: PoR response with CC applied to it
11: PoR response with DS applied to it
0 : PoR response shall not be ciphered
1 : PoR response shall be ciphered
0 : PoR response shall be sent using SMS-DELIVER-REPORT
1 : PoR shall be sent using SMS-SUBMIT
Reserved

Figure 8. Security Parameter Index Coding [After Ref. [22])

The KIc describes the key and the ciphering algorithm used. The specifications allow for the implementation of proprietary encryption algorithms. Figure 9 shows the coding of the KIc values. It can be seen that no key exchange mechanism is built into the specifications. It is assumed that the agreement on the key to be used has already been established.

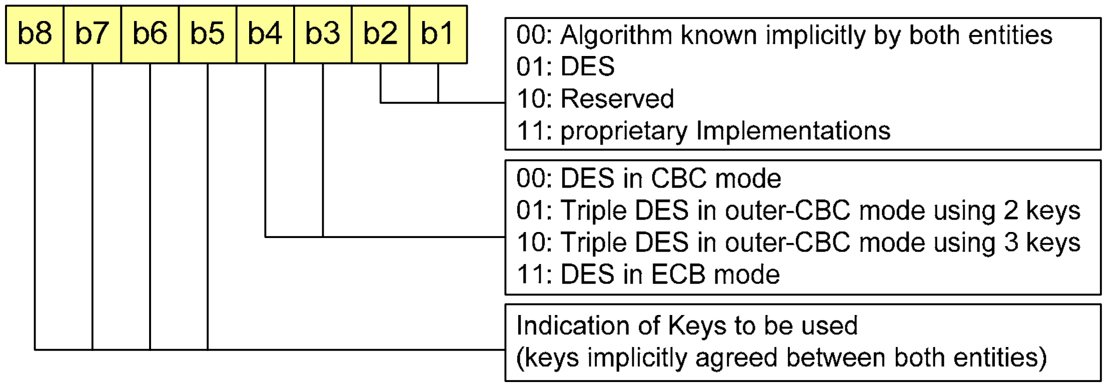

b8
b7
b6
b5
b4
b3
b2
b1
00: Algorithm known implicitly by both entities
01: DES
10: Reserved
11: proprietary Implementations
00: DES in CBC mode
01: Triple DES in outer-CBC mode using 2 keys
10: Triple DES in outer-CBC mode using 3 keys
11: DES in ECB mode
Indication of Keys to be used
(keys implicitly agreed between both entities)

Figure 9. KIc Coding (After Ref. [22])

The KID refers to the key and algorithm used to compute the redundancy check (RC), cryptographic checksum (CC) or digital signature (DS) of the secured data. The coding is very similar to the KIc and is shown in Figure 10.

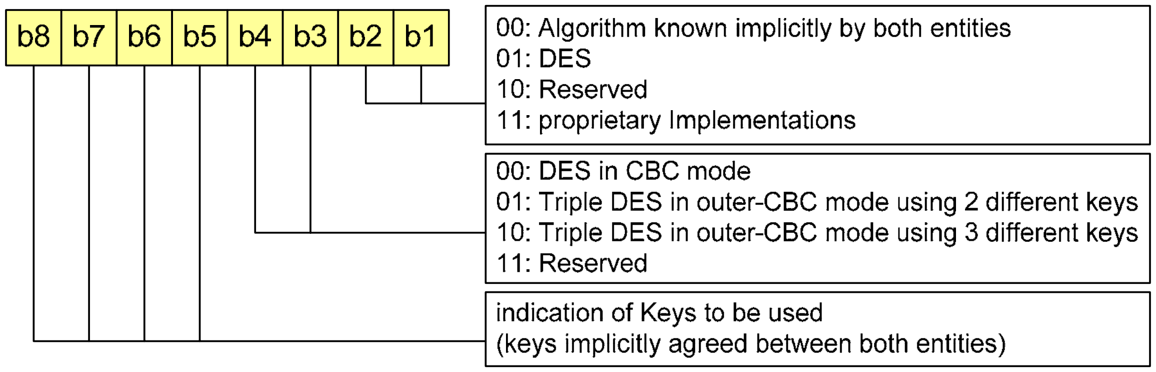

b8
b7
b6
b5
b4
b3
b2
b1
00: Algorithm known implicitly by both entities
01: DES
10: Reserved
11: proprietary Implementations
00: DES in CBC mode
01: Triple DES in outer-CBC mode using 2 different keys
10: Triple DES in outer-CBC mode using 3 different keys
11: Reserved
indication of Keys to be used
(keys implicitly agreed between both entities)

Figure 10. KID Coding (After Ref. [22])

The Toolkit Application Reference (TAR) is used to indicate which application should handle the secured data, similar to the use of port numbers in Transmission Control Protocol (TCP). However the definition of its use is very fuzzy in the specifications. The official description is “coding is application dependent.”

The Counter (CNTR) indexes the messages between the application server and the SIM. The main purpose is to create a nonce to prevent replay attacks. However, the management of the counter value is challenging if the application or the SIM needs to keep track of the counter values in conversations with multiple parties. As such, a weaker method of counter was implemented in some applications using time stamp values in the CNTR field.

The Padding Counter (PCNTR) is the number of padding bytes at the end of the secured data. This is typically required in block ciphers, where the data is encrypted in fixed block sizes. If the data is not in multiples of the block size, the last block needs to be padded to the block size.

A Redundancy Check (RC), Cryptographic Checksum (CC) or Digital Signature (DS) is used to verify the integrity of the secured data.

It is apparent that the SMS application layer only provides options for describing the security context between the SMS applications and SIM. Data confidentiality protection, integrity protection, and anti-replay mechanisms can be described. However, the specific implementations of all these mechanisms are left to the application developer. No specific requirements were placed at the application layer to secure SMS. Ultimately, SMS still rides on the security provided by the GSM network. The specifications merely provide application developers with options to describe the security measures that are implemented.

D. MOBILE DEVICE SECURITY

As the SMS message arrives at the mobile device, it is subjected to another set of threats. This Section highlights some of the threats and risks in a mobile device.

1. Physical Security

One of the biggest threats for cell phones and mobile devices is physical theft or loss due to their high value and small size. In an independent survey conducted by The Ponemon Institute in August 2006, 81% of the surveyed companies experienced one or more lost or missing laptop computers containing sensitive or confidential business information in the past 12 month period [23]. A recent data breach involved the loss of a United States Veteran’s Administration (VA) employee’s laptop computer containing the names and Social Security numbers of almost 27 million living veterans. The laptop was stolen from the employee’s home office. As cell phones become more portable and powerful in terms of processing power and memory storage capacity, it can be expected they will be subjected to the same types of threats and losses as laptops today.

2. Security Features

Mobile devices are also restricted by their input devices. Small keypads or touch panels are used to reduce the overall size of the device. Security mechanisms such as password protection may be implemented. However, the input of the passwords is not as efficient compared to a desktop computer. The choice of password is often reduced to numbers for the purpose of convenience. This severely reduces the password space that an attacker needs to go through in a brute force attack. The use of a complex password, comprising upper and lower case alphabets and special characters, will affect the usability severely on a mobile device. That is why many access control mechanisms are reduced to a Personal Identification Number (PIN) on mobile phone devices.

3. Information Leakage

Most modern smart phones have huge memory capacities. Many of them feature expansion slots for memory devices like Secure Digital (SD) cards. Today, a four gigabyte SD card can be bought for less than US$80. Using expansion slots, the amount of memory available to the mobile device is virtually unlimited. This means that huge amount of data can be stored, and potentially lost.

Furthermore, the modern mobile devices feature a rich set of connectivity options including high speed connections, like 802.11g. If the mobile device is connected to the corporate network, a huge amount of information can be leaked to the mobile device in a very short time.

Besides sensitive corporate data, personal information, such as address books, phone books, email correspondences, all reside on the mobile device. Once the data is compromised, the owner, family members, and friends may be subjected to identity theft, depending on the type of information kept by the owner in his address book.

4. Operating System

All popular Operating Systems for mobile devices today implement a monolithic kernel without protection ring or domains. This means that any user gaining physical access to the device has access to the supervisor mode. There is no separation between user mode and supervisor mode. The access control is, at best, enforced via password or PIN protection. Therefore, anyone holding the device has access to all the information on the device and all the networks that the device is authorized to connect to. A common overlooked connectivity is the HotSync function, which is activated when the phone is being charged at a docking station connected to a laptop or Personal Computer (PC). A network connection is automatically established to the laptop or PC for synchronization of data. Usually this connection bypasses all of the device password protections. By connecting to the laptop or PC, the mobile is also connected to the rest of the network to which the laptop or PC is connected.

E. SECURE SMS APPLICATIONS

The previous Sections have shown that the entire SMS system is riddled with security issues. Yet, many businesses are accepting the risk because the benefits of SMS far outweigh the potential cost of information compromise. If the security of SMS can be improved and the security risk reduced, many more applications may be able to reap the benefits of the matured GSM networking technology and SMS messaging system. The rest of this Section discusses why security is required in applications such as IMAS and some potential uses of SMS in military operations. Chapter IV describes the implementation of one such application.

1. Integrated Mobile Alert System (IMAS)

The IMAS [25] project was conceptualized at the Naval Postgraduate School (NPS) and is currently under development. The aim of the project was to provide a common method for people to stay connected in order to receive alerts across a wide variety of platforms. The project uses an extended online calendar to capture user's context information. Besides the date, time, location and event information in the online calendar, the user can also describe how he can be reached when certain events occur, what events to which he would like to be alerted, and what he needs to know about the event.

Although the objective sounds logical and straightforward, the project makes use of some key concepts that are crucial for enabling true mobile computing. First, the system must possess information value awareness by which to recognize, first of all, what is urgent or important to the user. Second, the system must be aware of the device that the user is holding and the network connectivity of that device. Based on these factors, the IMAS can then repurpose the content and deliver the alert to the user in its most usable form. The IMAS architecture is shown in Figure 11.

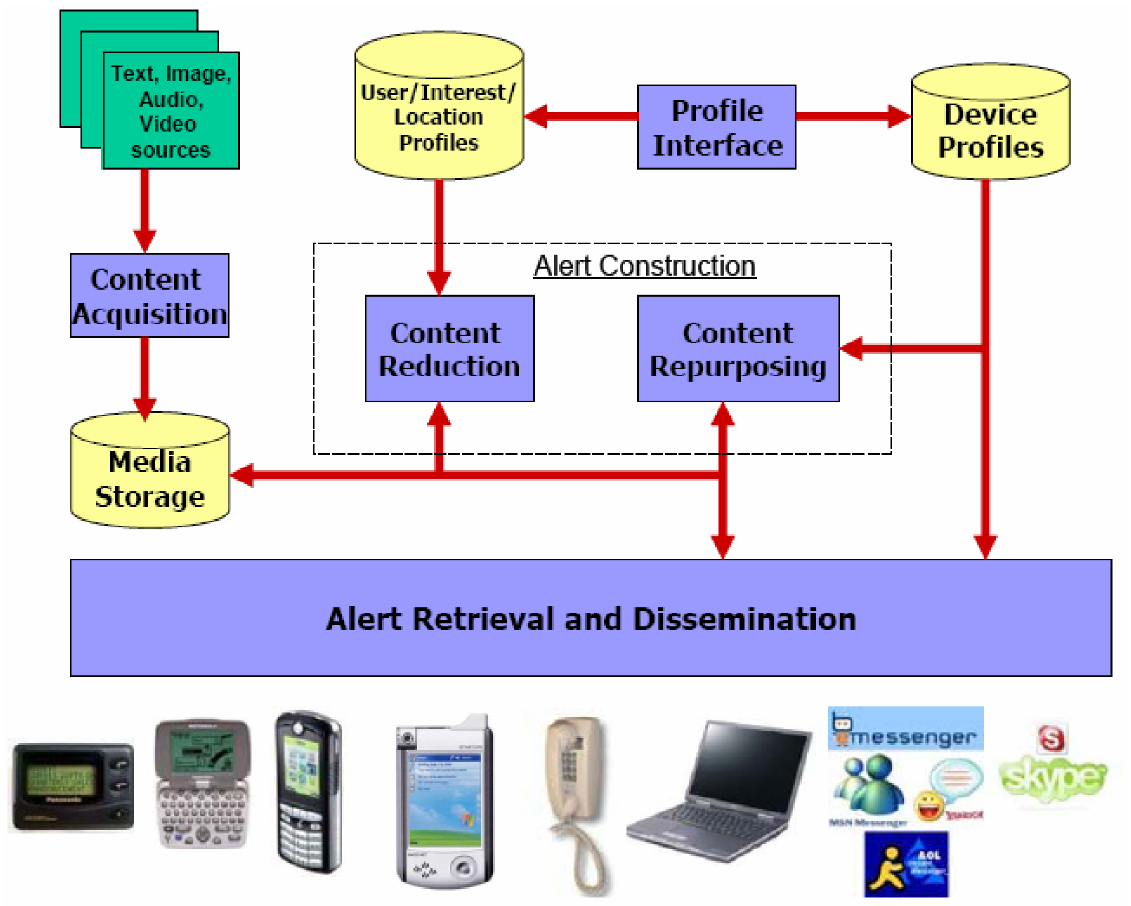


Figure 11. Integrated Mobile Alerts System Architecture (From Ref. [25])

Currently, the online calendaring and user profiling part of the system have been implemented and tested [26]. The User Profile allows the user to specify devices available during his various context settings. For example, a user may state that when he is in an off-site meeting the only way of contacting him is his cell phone. In this mode, he would like to receive urgent emails as SMS. SMS as an alert delivery mechanism in IMAS has also been implemented. In this case the security of SMS is of paramount importance because urgent emails that require immediate attention are likely to be sensitive in nature and need to be protected against eavesdropping.

2. SMS for Military Applications

Military organizations are particularly concerned with the confidentiality, integrity and availability of information in conventional military operations. Given the GSM and SMS vulnerabilities described in the previous Sections, none of the current security properties is sufficient for military high assurance applications. However, the emergence of asymmetric threats and the increased participation of the military in Humanitarian Aid Disaster Relief (HADR) operations have resulted in the need for the military to work closely with civilian institutions or Non-Government Organizations, such as Red Cross. Military communications devices and networks may not be suitable for civil-military communications because of Operational Security (Opsec) limitations. Furthermore, the civilians are not trained to operate the military communications devices. GSM and SMS may be able to fill this void in the civil-military communications network.

The military has recognized the pervasiveness of the GSM network and the need to leverage successful mature Commercial-Off-The-Shelf (COTS) technologies, such as GSM. Secure phones that are Type I certified have been developed and are in use in the military today [27]. However, the use of these phones is limited to voice traffic or modem connections, and the distribution of phones is limited due to cost and key management concerns. The rest of this Section discusses some current concerns of the military on the use of SMS and potential SMS applications that may be useful for the military, if the security of SMS messages can be assured.

a. Current Concerns about SMS Usage

The key concerns in the military are the confidentiality and integrity of SMS messages. It is often perceived that all military information must be protected with the highest level of assurance. However, security must be viewed in the context of the mission and the value of the information that needs to be protected. Not all information needs to be protected to the same level as national secrets. Overprotecting the data results in a waste of resources and places unnecessary constraints on the system. Besides the value of the content, the volatility of information is also very important when considering the security solution. A classic example is information on D-Day; the value of information is extremely high before D-Day. After D-Day, the information no longer needs to be protected. Tactical information on troop location is another example. The exact geographical position only needs to be protected from the enemy for the duration of the operation, assuming the operation is not a clandestine one. Therefore the issue of confidentiality and integrity protection must be viewed in the context of the value and volatility of information.

Another area of concern is availability of service. GSM frequency is well known and electronic jamming is trivial. However, the same concern applies to GPS. GPS are known to be vulnerable and commercial GPS jammers are readily available. However, the military continues to rely on GPS for its operations. The key is to ensure that GPS jammers can be located and destroyed. The same can be applied to GSM. Just as GSM jammers can be bought, GPS jamming locators are also commercially available. They are typically used by GSM service providers to locate radio interference sources and to maintain a certain level of Quality of Service. The effective and creative use of such devices is the key in maintaining superiority in the frequency spectrum.

With vendors constantly marketing high bandwidth devices, the bandwidth provided by SMS is perceived as insufficient for dynamic, fast moving, military applications. Again, the bandwidth usage must be viewed in context of the operation. A SMS message can be delivered end-to-end in seconds. A SMS message is sufficient to encode the position, course, speed and status of a soldier or unit. Not every soldier is required to stream video. Therefore, SMS can still fulfill the needs of some applications.

The physical loss of mobile devices is a concern for all mobile communications equipment. What happens if the soldier is captured? What if device falls into the hands of the enemy? Can the enemy masquerade as an insider of the network and foil the operation? These are questions that constantly arise during the development of any military mobile communication system or device. The same questions apply of GSM handsets.

The following Sections highlight the advantages of SMS over conventional military communications devices and specific scenarios where the SMS may be useful for military applications.

b. Advantages of SMS

Besides being private and non-intrusive, GSM and SMS also offer the following advantages from a military perspective:

- GSM infrastructure and handsets are cheap as compared to their military counterparts.
- GSM handsets are commercial commodity and do not project a military look. In the realm of Information Warfare and Signature Management, this is a good alternative for secure communications in scenarios where the use of military equipment may be complicate working relationships with civilian organizations or Non-Government Organizations (NGOs).
- GSM has global coverage and is expected to cover 90% of the world population in 2010 [24]. The military can ride on the existing infrastructure for initial quick response.

c. Intelligence Collection

The intelligence community has special interest in maintaining secure clandestine communications networks. Secure SMS provides another option to the existing communications options. However, other mechanisms to ensure the anonymity of the source of the message must be in place for such deployments to be effective.

d. Civil-military / HADR Operations

In some scenarios, a nation may require aid from the US military after a natural disaster strikes. However, due to political sensitivities, the US military may not want to be seen playing a dominating role in the HADR operations. This is where the use of commercial networks may seem more acceptable to the host nation. NGOs are also more receptive towards the use of commercial handsets compared to a military communications device. However, the military may still want to provide some level of information assurance in the communications with the civilian counterpart.

e. Low Data Rate Urban Communications

Communications in build-up areas with tall concrete buildings is challenging. Satellite communications are limited because they require line-of-sight to the satellites in order to operate properly. SMS is especially useful in such environments because of its store-and-forward mechanism. The SMS is held in the system and delivered once the device comes online. Therefore, SMS can complement any existing communications setup by providing an effective backup communications for low data rate applications such Blue Force Tracking in urban environments. The message delivery notification has been built into the SMS specification to ensure reliable delivery of SMS messages. However, there are no message priority mechanisms to prioritize urgent messages. Therefore, SMS may be suitable for use the primary war fighting net.

f. Secure Chat

The ability to conduct secure text chats among tactical units may be useful in certain missions. Current tactical military communications are still largely voice-based. Voice communications may be effective in a dynamic combat situation when real time conversation and feedback is crucial, and the soldier may be required to be constantly on the move. However, secure chat may be useful in silent surveillance operations or for an exchange of quick updates among tactical commanders. Text eliminates any ambiguities and misunderstandings in voice communications.

g. Encryption for Every Soldier

It has been reported that soldiers and sailors send sensitive, hopefully unclassified, data such as ship movement and troop deployment to their love ones over the public communications network. It is important that such information be denied to the adversary as far as possible because the soldiers are likely to be deployed in other countries and using a foreign GSM infrastructure. Therefore, a simple application layer software encryption solution for SMS can provide an added level of security. The aim is not to encourage soldiers to divulge sensitive information over public networks. Suitable OPSEC policy must still be enforced. Even if the messages are non-sensitive, inferences can be made from a collection of such messages. Adding a layer of encryption makes such inferences difficult as it is less likely the unintended recipient of the information will be able to analyze it when it is encrypted.

III. ENCRYPTION SCHEME SELECTION

A. OVERVIEW

Encryption is the process of disguising information in such a way as to hide its substance [28]. Modern encryption methods can be divided into symmetric key algorithms and asymmetric key algorithms. The One Time Pad is unique because it is the only encryption scheme that is unbreakable even in theory. The discussion in the rest of this Chapter will focus on these three types of encryption.

This Chapter is comprised of four main parts. The first part of this Chapter provides an overview of the different schemes of encryption and their relevance to securing SMS. The second part discusses the key considerations when selecting an encryption scheme for deployment. The third part describes an experiment that was conducted to measure the performance of symmetric and asymmetric encryption schemes on a modern cell phone. The final part summarizes the findings in a selection matrix that may be useful for application developers, who plan on deploying encryption for SMS messages.

B. ENCRYPTION SCHEMES

1. Symmetric Cryptography

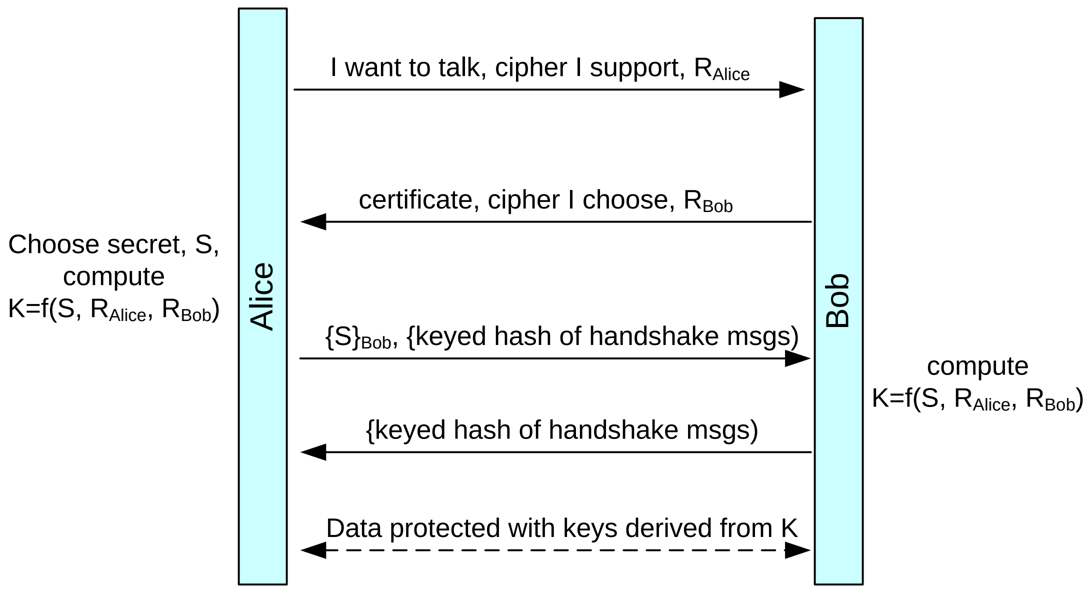
In symmetric encryption, the sender and receiver must have a pre-shared key that is kept secret from all other parties. The sender uses the key for encryption, and the receiver uses the same key for decryption. The key advantage of symmetric encryption is that it is computationally fast and efficient. This makes symmetric encryption the ideal choice for mobile devices. The A3, A8 and A5 algorithms used in GSM are all symmetric encryption algorithms. Other strong symmetric algorithms available today include Triple Data Encryption

Standard (TripleDES) and Advanced Encryption Standard (AES), which have been approved for use by National Institute of Standards and Technology (NIST), and are publicly available.

The key disadvantages of symmetric encryption are the need to pre-share the keys among the senders and recipients and the keys must be exchanged securely via some trusted communications channel or through some key exchange mechanisms. In an infrastructure setup like GSM, this is manageable because all the subscribers share common keys with the service provider. If the subscribers need to communicate with each other, the service provider acts as the middleman and encrypts/decrypts the messages, as required. However, if symmetric encryption were to be used at the application layer, the key exchange would have to be managed separately and this can be quite a challenge because all the users of a group must use the same key. If the key is compromised, a new key must be redistributed to every user. If there is a need to partition the communications into sub-groups, different sets of keys must be created and distributed for each sub-group. A separate key is still required for the entire group. This complexity grows as the number of users and sub-groups increases.

Secure key exchange mechanisms, such as Internet Key Exchange (IKE) and Secure Socket Layer, have been developed to facilitate key exchanges across public networks. However, these protocols assume relatively high bandwidth, real-time connectivity between the sender and recipient. For example, the set up of an SSL session requires an exchange of at least four messages, as shown in Figure 12, before the secure session is established. For SMS, sending each message may take a few seconds. The exchange of four SMS messages for each session will affect the usability severely.

Therefore, the ability to deploy symmetric encryption at the application layer for SMS will depend on the ability to exchange keys securely. The key exchange can take place through physical transfer via storage devices, or if the device is cradled and connected to a PC, with a VPN connection. In order to reduce the key distribution complexity, a star topology may be adopted, such that all the clients will send all SMS messages to an application server for relay. The disadvantage of such a set up is the delay in transmission of messages because each message is effectively transmitted twice. However, it offers the advantage of simplifying key exchange.


Alice
Bob
I want to talk, cipher I support, R_{Alice}
certificate, cipher I choose, R_{Bob}
Choose secret, S, compute $K=f(S, R_{Alice}, R_{Bob})$
$\{S\}_{Bob}$, {keyed hash of handshake msgs}
compute $K=f(S, R_{Alice}, R_{Bob})$
{keyed hash of handshake msgs}
Data protected with keys derived from K
Figure 12. Secure Socket Layer (From Ref. [29])

2. Asymmetric Cryptography

In an asymmetric key algorithm, there are two separate keys: a public key and a private key. The public key is published and enables any sender to perform encryption; the corresponding private key is kept secret by the receiver for decryption. The key exchange in asymmetric encryption is much simpler because the public can be freely distributed. There is no requirement for separate keys for sub-group communications. The sender will encrypt the messages with the public keys of the recipients. This provides extreme flexibility when the group composition may change dynamically, such as in military operations.

However, the key disadvantage of asymmetric encryption is that it is computationally more expensive than symmetric encryption because of the long keys. Therefore, asymmetric encryption is seldom used in mobile devices. Even on desktop computers, asymmetric encryption is used together with symmetric encryption for performance reasons. Section D of this Chapter attempts to shed some light on the resource consumption of asymmetric encryption on a modern Smartphone device through actual measurements.

The more commonly used asymmetric algorithms are Rivest Shamir Aldelman (RSA) and the Digital Signature Algorithm (DSA). Elliptic Curve Cryptography (ECC) is another approach to asymmetric algorithm that allows shorter key lengths to be used, making it suitable for mobile device applications. However, ECC is still under active research and has not been widely deployed in systems.

For the encryption of SMS messages, the use of asymmetric encryption alone may be feasible due to the short message length. The measurements described in Section D also provide indications of the feasibility of such implementation.

3. One Time Pad

One Time Pad (OTP) is a unique form of cryptography because it is the only unbreakable cipher, even in theory. OTP operation is straightforward and simple. The key is a string of random numbers that is as long as the message itself. Each key is used for only one message; the key is never reused and assumed to be perfectly random. A random key sequence added to a non-random plaintext message produces a completely random cipher text [28]. Therefore every plain text message is possible and there is no way for the cryptanalyst to determine which plaintext is the correct one.

The biggest challenges in using OTP are the generation of truly random key sequences and the key exchange. Similar to symmetric encryption, all parties in the secure conversation must have the same key. Furthermore, the key is now as large as the amount of data that is to be exchanged in the whole network. Furthermore, the use of keys must be synchronized among users such that the keys are never reused.

The use of OTP may have great potential for SMS encryption because of the recent advances in memory technology. Mobile devices are now equipped with memory card slots for external memory devices such as Secure Digital (SD) cards. Memory capacities for SD cards have grown and the prices have dropped tremendously. A four gigabyte SD card may be purchased online at a price of less than US$100. Assuming that a 160 byte key sequence is used to encrypt every SMS message, 25 million messages can be encrypted using OTP. If 10 messages are exchanged every minute in a network 24 hours a day, a four gigabyte key bank is enough to sustain 4.7 years of usage without the need for a key change. Therefore, the use of OTP may have great potential in applications where the content of the SMS is highly confidential.

C. KEY CONSIDERATIONS

There are several key considerations when choosing a suitable encryption scheme for an application. The most important consideration is the deployment scenario for the use of the encryption scheme. The value of the information to be protected, the expected threats and the physical disposition of the users are key considerations when deciding on a suitable encryption scheme. However, due to the varied nature of the deployment scenarios, these factors are not discussed here. This Section assumes that the risk assessment has already been conducted and SMS is assessed to be a viable communications channel based on the security policy. The rest of this Section highlights the important considerations when choosing an encryption scheme for mobile devices.

1. Algorithm Strength

The strength of an algorithm is derived from the mathematical properties of the algorithm. For example, RSA derives its strength from the difficulty of factoring large numbers. If a method that radically speeds up the factoring of large numbers is discovered, the RSA algorithm can be broken [29]. This is the fundamental assumption behind the use of algorithms. Since, nobody can predict the breakthroughs in mathematical methods, developers writing applications that use encryption must be mindful of the assumptions behind the algorithms and be updated of the latest developments.

While no one can be absolutely certain about the strength of an algorithm, it is generally accepted that an algorithm that has been subjected to public peer review is more secure, except against the most determined and resource rich state agencies [29]. The GSM A5 encryption algorithm is an example of a failure in encryption algorithm that has been developed in secrecy. Once the closed algorithms are leaked or reverse engineered, they will be crypto-analyzed. Therefore, secret algorithms only provide added security if they are as closely guarded as the secrets that they are meant to protect, and the algorithm is designed by cryptography experts who know exactly what they are doing, and the algorithm has gone through some internal, independent reviews.

2. Key Length

The other vital component in the security of an encryption system is encryption key. If the encryption algorithm is publicly available, then the strength of the encryption is only dependent on keeping the key secret. An attacker can conduct a brute force attack by trying all possible key combinations. The challenge is then deciding on the suitable key length, such that a brute force attack will not be successful in a timeframe shorter than the lifespan of the message that it is protecting. In Table 1, Denning summarized the effort and time required for such an attack based on the rate at which the attacker is able to test each key [30].

Row	Rate	Second	Hour	Day	Week	Month	Year
1	10^5	17	28	33	36	38	42
2	10^6	21	32	36	39	41	45
3	10^7	23	35	40	42	45	48
4	10^8	27	38	43	46	48	51
5	10^9	30	42	46	49	51	55
6	10^{10}	33	45	50	52	55	58
7	10^{11}	37	48	53	56	58	61
8	10^{12}	40	52	56	59	61	65
9	10^{13}	43	55	60	62	64	68
10	10^{14}	47	58	63	66	68	71
11	10^{15}	50	62	66	69	71	75
12	10^{16}	53	65	70	72	74	78

Table 1. Length of Key That Can be Broken [After Ref. [30])

The *Rate* column shows the number of keys the attacker can try in a second. The entries under the *Second*, *Day*, *Week*, *Month*, *Year* columns correspond to the number of bits of keys that can be broken.

For example, the first row corresponds to a search rate of 100,000 keys per second (i.e. if a machine is able to test 100,000 keys per second); in which case, a 17-bit key can be broken in seconds, a 28-bit key in hours, a 33-bit key in days, a 36-bit key in weeks, a 38-bit key in months and a 42-bit key in years. The entries were calculated based on the worst case assumption that it was necessary to try each possible bit combination before finding the correct key. On an average-case assumption, the key can be found halfway through the key space.

The shaded cells in Table 1 show the successful efforts of cracking that have been demonstrated. The 56-bit key was cracked in 1999 in less than 23 hours, corresponding to a search rate between *Rows 7* and *8* in the table.

According to Moore’s law, each successive row, representing a tenfold improvement in processing speed, corresponds to a five year timeframe. Based on this projection, the present key search rate should roughly correspond to *Row 9*.

The values in Table 1 are relevant for symmetric encryption, where the key length is the main determining factor of the strength of the key. The common key lengths for symmetric encryption in use today are in excess of 100 bits: 192 bits for TripleDES and 128 or 256 bits for AES. Therefore, the key lengths can be regarded as safe against a brute force attack. The weaker link in the entire system goes back to the strength of the algorithm. However, there are also other crypto attack techniques such as statistical attacks and side channel attacks[1] that reduce the key space through which the attacker needs to search. Once the key space is small enough, a brute force attack may prove effective.

Unlike symmetric algorithms, the asymmetric encryption algorithms derive their strength from the difficulty of factoring large numbers that are the product of two large prime numbers. Therefore, although the key lengths used in asymmetric algorithms are much longer than symmetric encryption, the attacker does not need to try every possible key combination. If the factoring difficulty of asymmetric encryption algorithm is taken into account and compared to the difficulty of brute force attack against symmetric encryption, Schneier [28] proposed a comparison between symmetric and asymmetric key lengths, shown in Table 2.

[1] Side channel attacks refer to attacks based on information gained from the physical implementation of a cryptosystem, such as timing information, power consumption or electromagnetic leaks [32].

Symmetric Key Length	Asymmetric Key Length
56 bits	384 bits
64 bits	512 bits
80 bits	768 bits
112 bits	1792 bits
128 bits	2304 bits

Table 2. Symmetric and Asymmetric Key Length Comparison

Table 2 provides a reference for relative comparison of key lengths. However, this is still based on the current mathematical knowledge regarding factoring large numbers. Once a newer and faster method of factoring is discovered, the values in the table no longer hold.

3. Key Management

Key management is an important part of any encryption solution. The strongest algorithms and the longest keys are useless if the keys are not generated, distributed and destroyed in a secure manner. Stealing keys is an attractive option for the attacker because the attacker does not need to expend resources to break the encryption algorithm, if he can break the key generation algorithm. The attack of the COMP128 algorithm is an example of an attack on the key generation algorithm.

A big challenge in key generation is ensuring that the keys generated are truly random and thus not predictable. The difficulty depends on the key lengths and the frequency of key change. For One Time Pad, the keys are as long as the messages and the keys must never be repeated. This may be extremely difficult depending on the volume of traffic.

There are also known weak keys in certain algorithms. Therefore, after the key generation process, it is important that the keys be checked to ensure that they do not belong to the pool of known weak keys.

After the keys have been generated, they must be distributed via a trusted communications channel. Key distribution for a large network can be cumbersome. For symmetric encryption, the number of key exchanges required in a network with n users is $n(n - 1)/2$. The key exchange for asymmetric encryption is simpler because the public key can be exchanged via a public network. However, mechanisms must be built into the system to ensure the authenticity of the public keys.

4. Power Consumption

Power is a key constraint for mobile devices. Therefore, most mobile devices have extensive power management features to conserve battery power as much as possible. Encryption operations are computationally intensive. Therefore, the power consumption of encryption operations must be taken into account when deciding on an encryption scheme. Section D of this Chapter describes an experiment that was conducted to measure the power consumption of different encryption operations.

Although the power consumption associated with each encryption operation may be small, encryption operations occurs very frequently in network encryption. For example, in Internet Protocol Security (IPSec), every Internet Protocol (IP) packet is signed and encrypted. This translates to three encryption operations per IP packet that is transmitted or received: one for confidentiality protection, one for hashing, and one for digital signature. The cumulative consumption can be quite significant, depending on the amount of network traffic.

5. Speed

The speed and efficiency of the encryption operation has a direct impact on the network bandwidth and the usability of the system. Saltzer and Schroeder [31] wrote about psychological acceptability as one of the security design principles. Security solutions that are implemented must be usable and as transparent to the user as possible. If the solution affects the usability, the user will not use it or may even try to disable or bypass the security solution. In the case of SMS encryption, the users are used to a nominal time required to send an SMS message, typically less than 10 seconds to receive a “Message sent” reply from the SMSC. If the encryption slows down the sending process significantly, the users will be frustrated and may choose to disable the encryption.

6. Overheads

Encryption operations add overhead to the length of the original message. Many encryption algorithms encrypt data in fixed block sizes. If the data is larger than the block size the last part of the message is usually padded to the full block size and encrypted. The overhead for padding may not be significant for large messages. However, it may be quite significant for short message, such as those using SMS. For asymmetric encryption, the block sizes are relatively big because they are related to the key lengths. For example, the encryption of a single byte of data using RSA with 1024-bit key length yields a 256-byte output; a 2048-bit key length yields an output of 344 bytes. These overheads translate to additional transmission overheads, which in turn increases the power consumption. Therefore, the choice of an appropriate key length may reduce the overall power consumption without compromising the security.

D. PERFORMANCE MEASUREMENTS

The previous sections described the power consumption, speed and overhead considerations for encryption solutions. The aim of the measurements is to provide a means to assess an actual performance experiment of some encryption schemes on a specific modern device. Due to the varied hardware and software implementations on mobile devices, this set of measurement figures cannot be taken as definitive. However, it provides a comparison among the different encryption schemes and a coarse estimate for programmers planning to implement software-based encryption in their applications.

The main tasks of this experiment were collect empirical data on the power consumption, the time associated with encryption and the data size overheads imposed by selected symmetric encryption and asymmetric encryption schemes.

1. Instrumentation Setup

a. Hardware

The mobile device used for this experiment is the Eten-M600 Smartphone. The hardware specifications for the device are listed in Table 3. This phone was selected because it has most of the features that can be expected to be found in future mobile devices. The Operating System used is the latest Windows Mobile™ 5.0 with a rich Application Programming Interface (API) support for application developers.

Operating System	Windows Mobile™ 5.0 software for Pocket PCs
Processor	Samsung S3C 2440 400 MHz Processor
Memory	256 MB Flash ROM, 64 MB SDRAM
Display	2.8", 240 x 320, 65,536 colors LTPS TFT- LCD
Dimensions (LxWxH)	111.7 x 60.7 x 22 mm
Weight	174 g
Communications	GSM quad-band 850/900/1800/1900 MHz, GPRS Class B / Multi-slot Class10 Bluetooth v2.0 compliant, WiFi IEEE802.11b
Camera	Built-in 2.0 Mega Pixels, up to 1600 x 1200 resolution
Expansibility	SDIO/SD/MMC card slot
Interface/Audio	Built-in microphone and speaker, external stereo headset jack
Interface/Data	USB Sync, headset jack, Cradle with 2nd battery charger

Table 3. E-TEN M600 Hardware Specifications

Figure 13. E-TEN M600 (From Ref. [33])

b. Software Development

The development environment used was Visual Studio 2005, together with the Windows Mobile 5.0 Software Development Kit (SDK) for Pocket PC, as recommended by Microsoft [34]. ActiveSync 4.2 was also required for debugging and deployment of the solution to the mobile device. The programming language used was C#.

c. Performance Measurement Application

Different approaches were tried to obtain accurate measurements of the performance data. However, due to the limitations imposed by resolution of power measurement in the API, and the failure in the State and Notification API (SNAPI), the approaches did not work. The details on the failed approaches are attached at Appendix A. In the final approach, the program flow is as shown in Figure 14.

The encryption process uses Microsoft's implementation of RijndaelManaged and the RSA algorithms in the Microsoft CryptoAPI (CAPI). The program code is attached at Appendix A. The performance measurements for One Time Pad operations are not measured because the mathematical operation of OTP is very simple, comprised by a few XOR functions. Therefore, the time and power consumption requirements are assumed to be significantly lower than symmetric encryption.

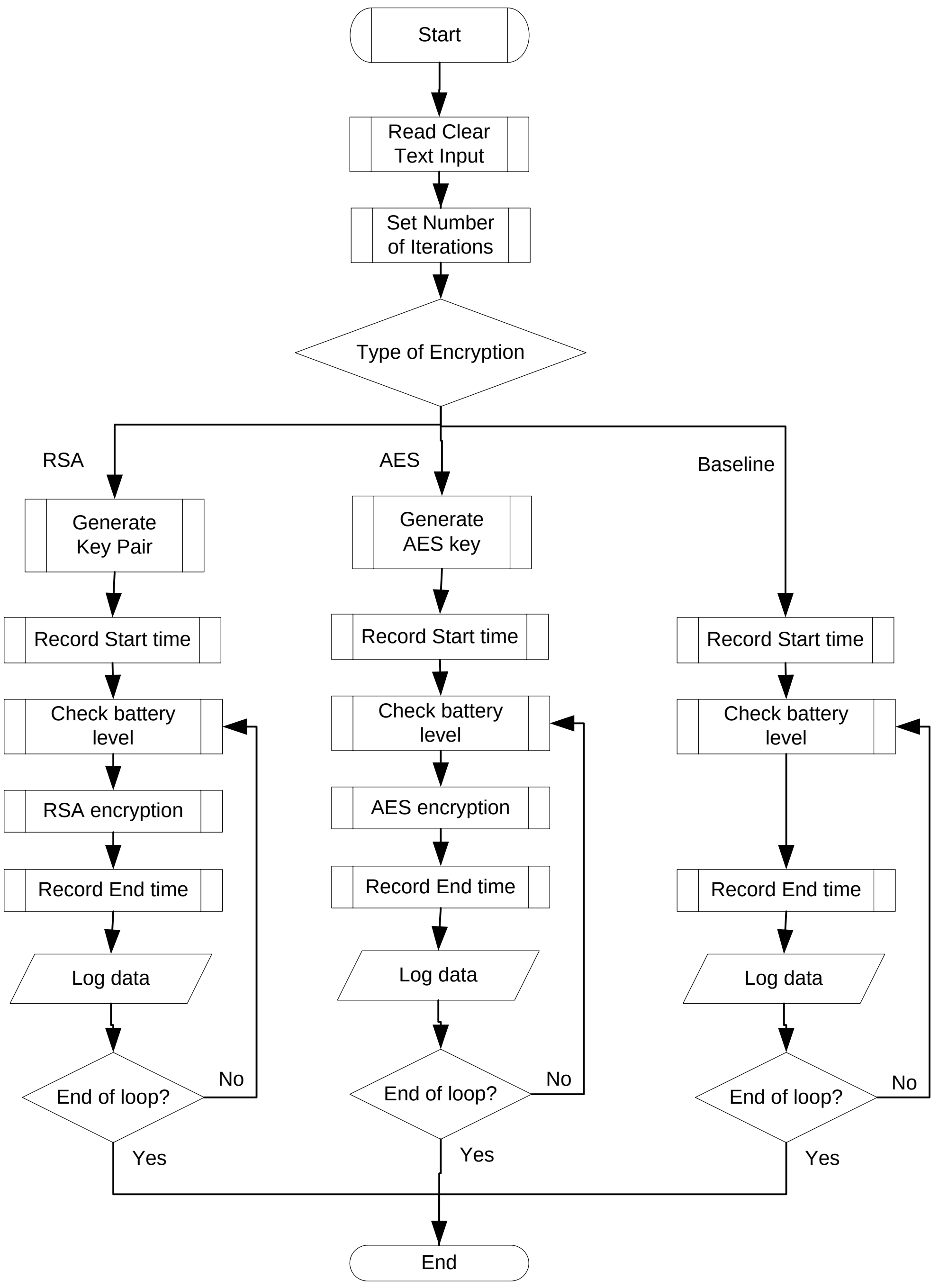
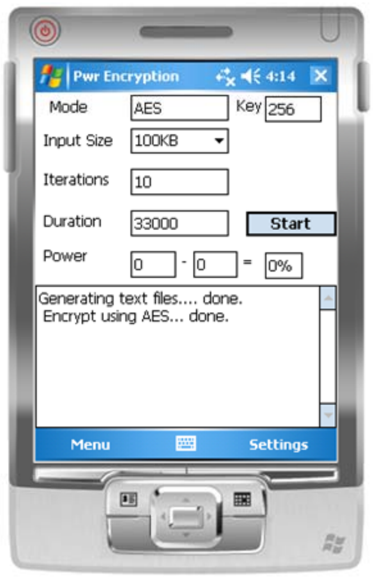


Figure 14. Flow Diagram for Performance Measurement Application

d. User Interface

The main screen captures for the application is shown in the figures below.

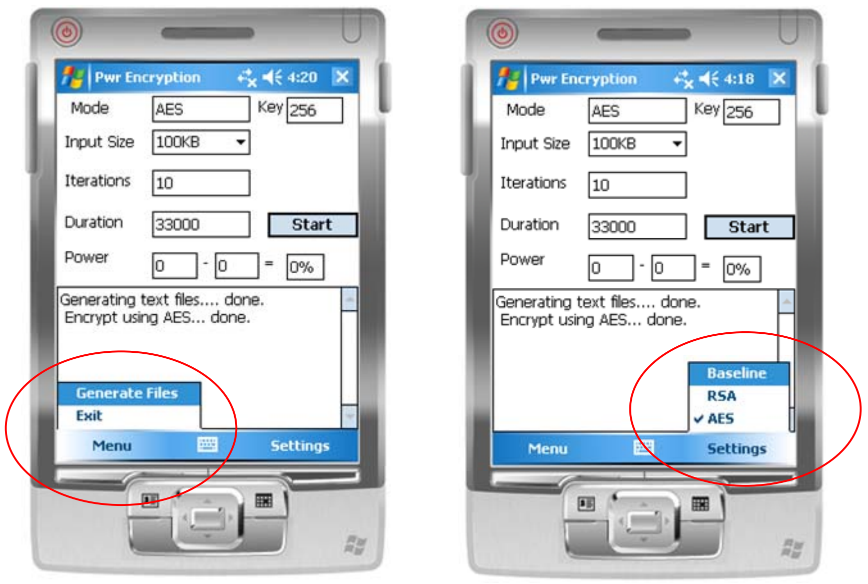


Mode AES Key 256
Input Size 100KB
Iterations 10
Duration 33000
Power 0 - 0 = 0%

```
Generating text files.... done.
 Encrypt using AES... done.
```

Figure 15. Performance Measurement Application Main Screen

The *Mode* field indicates the current encryption mode setting, which can be RSA, AES or Baseline. The *Key* field displays the current key length setting. The *Input Size* field is selectable from 160 bytes, 10 kb and 100 kb. However the measurements are conducted using inputs of 100kb only. The *Iterations* field allows users to key in the number of iterations for the encryption. The *Duration* field shows the time that has elapsed since the start of the first encryption. The *Power* fields, from left to right, show the starting battery power level, the current battery power level and the power consumption, respectively. The scrolling text box below is used to display status and other debugging information.



Mode AES Key 256
Input Size 100KB
Iterations 10
Duration 33000
Power 0 - 0 = 0%

```
Generating text files.... done.
 Encrypt using AES... done.
```

Mode AES Key 256
Input Size 100KB
Iterations 10
Duration 33000
Power 0 - 0 = 0%

```
Generating text files.... done.
 Encrypt using AES... done.
```

Figure 16. Performance Measurement Application Menu

The left menu has the option to Generate Files. This function generates input text files, which are exactly of the sizes of 160 bytes, 10kb and 100 kb, and filled with random ASCII characters. The same function is also used to create and refresh log files after the measurements. The right menu shows the current mode of encryption selected and allows users to change the mode as desired.

2. Assumptions and Limitations

a. Battery Properties

The data collection requires multiple charging and recharging of the battery. It is assumed that the chemical properties of the battery remain constant throughout the measurement process. A Lithium Ion battery like the one used in the experiment typically lasts 300-500 discharge/charge cycles [36].

b. Linearity

It is assumed that the discharge of the battery is linear and that the battery strength provided by the API is accurate. In actual fact, battery discharge is almost never linear. Typically, the discharge characteristics are more linear in the mid range. Therefore, in this experiment, only the middle bands between 40% and 80% are used as measurement data.

c. Hidden Processes

The manual checklist to disable processes can only disable user accessible processes. There may be other system processes that may be running the background that are invisible to and inaccessible by the user. It is assumed that the demands placed by such processes are constant and can be eliminated by subtracting the baseline measurement data from the other data.

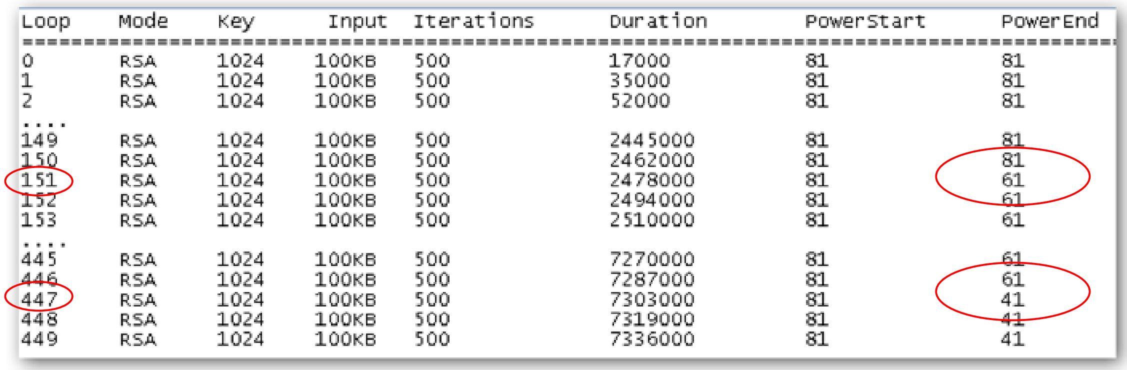
d. Correct Algorithm Implementation

The experiment is based on Microsoft's implementation of the encryption algorithm. It is assumed that these implementations are correct and representative of the types of encryption schemes.

3. Data Collection

Before the start of each measurement, a checklist was used to ensure that the critical settings that may affect the readings are consistent. For example, the wireless access for Wifi and GSM are turned off because they may introduce variable power consumption figures based on the detected infrastructure signal strength. The number of active process running on the device is also kept consistent so that the available memory and CPU demands are kept consistent. The detailed checklist is attached at Appendix A.

The log file that is generated was manually analyzed to determine the number of iterations that will result in the change in battery power levels. Figure 17 shows a sample of the output file.



```
Loop     Mode     Key      Input  Iterations        Duration          PowerStart        PowerEnd
====================================================================================================
0        RSA      1024     100KB   500              17000             81                81
1        RSA      1024     100KB   500              35000             81                81
2        RSA      1024     100KB   500              52000             81                81
....
149      RSA      1024     100KB   500              2445000           81                81
150      RSA      1024     100KB   500              2462000           81                81
151      RSA      1024     100KB   500              2478000           81                61
152      RSA      1024     100KB   500              2494000           81                61
153      RSA      1024     100KB   500              2510000           81                61
....
445      RSA      1024     100KB   500              7270000           81                61
446      RSA      1024     100KB   500              7287000           81                61
447      RSA      1024     100KB   500              7303000           81                41
448      RSA      1024     100KB   500              7319000           81                41
449      RSA      1024     100KB   500              7336000           81                41
```

Figure 17. Sample Output Log File

In the log file in Figure 17, the battery power level changed from the 81%-100% band to the 61%-80% band at the 151st iteration and moved on to the next band of 41%-60% at the 447th iteration. Therefore, 296 iterations of iterations consumed 20% of the battery power level.

However, the total consumption figure includes the logging of the data for each iteration. Therefore, another set of baseline results, comprising all operations not related to encryption, was collected, and subtracted from the measurement results to derive a more accurate indication of the power consumption attributed to the encryption operation.

Sufficiently large sample sizes were collected to ensure that each set of reading is statistically robust. The mean and standard deviation is then calculated based on a normal distribution. The raw data collected are attached at Appendix A. A summary of the results is shown in Table 4.

4. Analysis of Results

Table 4 shows a summary of the performance measurement results. The input size for the clear text was arbitrarily chosen as 100kb for comparison. The respective key lengths chosen for the encryption represent typical key lengths in use today that are generally regarded as secure.

	Key Length	Block Size	Input Size	Time (ms)		Power Consumption (mAH)	
	(bits)	(bytes)	(kb)	Mean	Std Dev	Mean	Std Dev
RSA	1024	117	100	16212.31	164.44	107.00	15.77
RSA	2048	117	100	25643.18	40.16	157.68	34.96
RSA	2048	245	100	11458.50	1172.65	80.02	24.34
AES	128	16	100	536.53	3.74	2.41	0.57
AES	256	16	100	586.59	3.45	2.91	0.39

Table 4. Performance Measurement Results

The baseline measurement results are shown in Table 5.

	Time (ms)		Power Consumption (mAH)	
	Mean	Std Dev	Mean	Std Dev
Baseline	128.61	26.65	0.56	0.21

Table 5. Baseline Measurement Results

Table 6 shows the adjusted performance results, after excluding all other operations not related to the encryption operation, such as data logging.

	Key Length	Block Size	Input Size	Time (ms)		Power Consumption (mAH)	
	(bits)	(bytes)	(kb)	Mean	Std Dev	Mean	Std Dev
RSA	1024	117	100	16083.70	191.09	106.44	15.98
RSA	2048	117	100	25514.57	66.81	157.12	35.17
RSA	2048	245	100	11329.89	1199.29	79.46	24.55
AES	128	16	100	407.92	30.39	1.85	0.79
AES	256	16	100	457.98	30.10	2.35	0.60

Table 6. Adjusted Performance Measurement Results

a. Performance Differences at Different Key Lengths

For a fixed input block size of 117 bytes for RSA encryption, the timing performance for RSA encryption using a 2048-bit key length was about 58 percent more compared to a 1024-bit key length, and consumed about 33 percent more power.

However, a larger block size of 245 bytes can be supported with a 2048-bit key length, resulting in less number of encryption cycles required. With a

245-byte block size, the timing performance was actually approximately 23% better than the performance at 1024-bit key length. The impacts of the key length and input size will be discussed further in a later Section.

For AES encryption, the timing performance at 128-bit key length was approximately 12 percent faster than at 256-bit key length and consumed about 22 percent less power. The block size for AES encryption is fixed at 128 bits (16 bytes). Therefore the total number of encryption operations to encrypt 100kb of data is the same. However, the internal mixing cycles of AES algorithm increases with longer key lengths.

b. Timing Comparison between RSA and AES

According to Table 2, the strength of 128-bit symmetric encryption is close to 2048-bit asymmetric encryption. From the timing results in Table 6, it can be seen that the time required for RSA encryption, using a 1024-bit key length and a block size of 245 bytes, is approximately 30 times longer than 128-bit AES encryption. In contrast to claims that asymmetric encryption is many orders of magnitudes slower than symmetric encryption, the empirical data showed that the actual time required for asymmetric encryption may not be significantly longer than symmetric encryption. In absolute terms, the difference is even less for smaller input lengths. Based on the results, encryption of a 100kb clear text using RSA with a 2048-bit key length and a 245-byte block size required approximately 12 seconds. For an SMS text of 160 bytes, the time required is significantly less, on the order of hundreds of milliseconds. The time required for symmetric encryption is even shorter. In terms of SMS communications, where the transmission of an SMS typically occurs in orders of seconds, the timing overhead imposed by symmetric and asymmetric encryption may not be perceivable by the user.

c. Power Performance Comparison between RSA and AES

Each encryption operation using RSA 1024-bit key length consumes approximately 106mAH of power, when encrypting 100kb of data. As compared to AES encryption, the consumption for RSA encryption is about 48 times higher. The battery capacity of the E-TEN M600 is 1440mAH. The battery capacities for similar devices available today are generally between 1200mAH and 1550mAH. This means that the E-TEN M600 will consume about 10 per cent of its battery power after encrypting 800 SMS messages using RSA encryption. The criticality of the power consumption will depend on the deployment scenario. If a user sends an encrypted SMS message and decrypts a SMS reply every minute, 800 SMS messages would be sent in 6.5 hours, and consume 10 per cent of the battery power. If the user is expected to be able to recharge the battery within this time frame, then the power consumption is not an issue.

d. Overheads Comparison

Table 7 summarizes the overheads incurred in terms of size when encrypting a 160-byte SMS message using the RSA encryption algorithm with different key lengths. Figure 18 provides a graphical representation of the data. The output sizes are categorized in terms of the number of SMS messages that are required to transmit the output. In Microsoft's implementation of RSA, the maximum key length is 16384 bits. The data in Table 7 stops at 4096 bits for ease of analysis. Data for the remaining key lengths can be extended easily. It is expected that the same pattern will continue and repeat itself. The overhead figure is calculated in percentage terms of the eventual SMS output size against the input size of one SMS message.

Key Length (bits)	**Absolute Output Size (bytes)**	**Output Size (No. of SMS)**	**Overhead**
768-936	264-320	2	100%
690-1344	328-448	3	200%
1368-1920	228-320	2	100%
1928-2880	324-480	4	300%
2888-3832	484-640	5	400%
3856-4096	644-684	6	500%

Table 7. RSA Encryption Overheads at Different Key Lengths

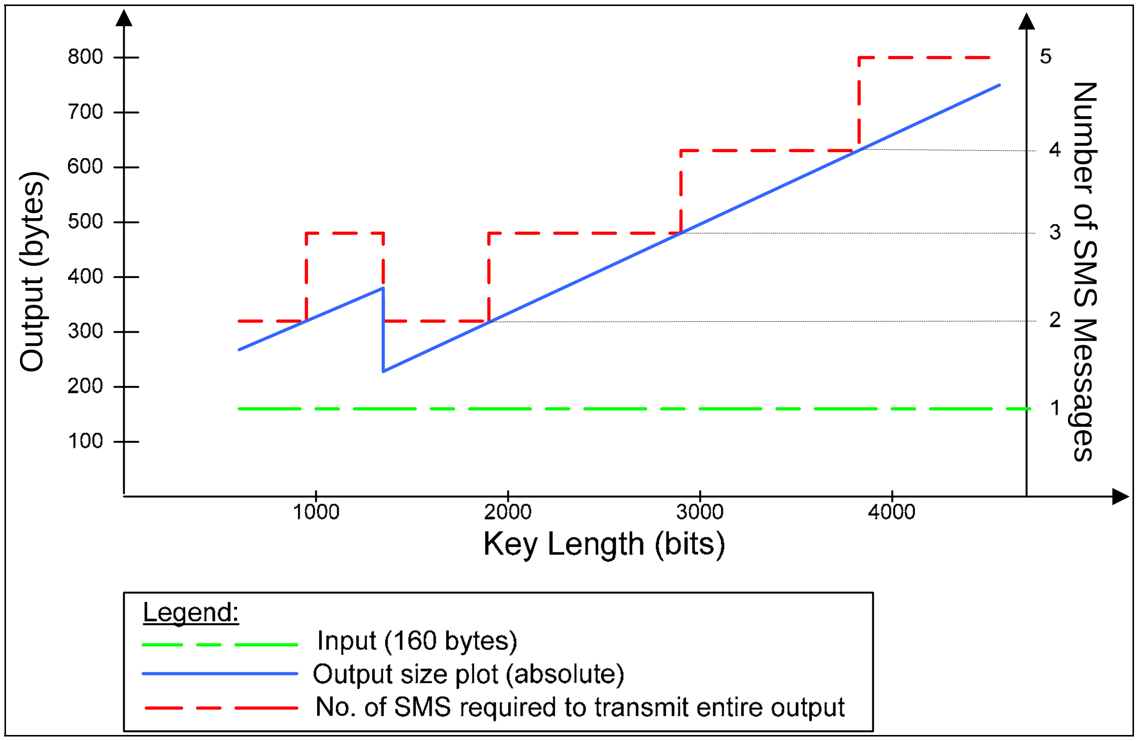


Figure 18. RSA Encryption Overhead for One SMS Message

From Figure 18, it can be seen that the length of the output increases with the length of the key. However at key lengths of 1360 bits and below, the maximum input size allowed is less than 160 bytes (one SMS message length). Therefore, the message has to be separated into two blocks of data and encrypted twice. The total output size is the results of the concatenation of two encryption outputs. At key lengths of 1368 bits and above, the maximum input size is equal or greater than 160 bytes. Therefore, the entire SMS message can be encrypted in one cycle. This explains the larger output sizes at key lengths shorter than 1360 bits.

This result shows that if the number of transmissions is to be minimized for a 160 byte input, the optimal key length is 1368 bits. The graph may be extended for different input sizes and key lengths.

In contrast, the output size for AES encryption does not vary with the key length for an input of 160 bytes; the output is fixed at 236 bytes and two SMS messages are required to transmit the entire output. This is because AES encrypts the data in block sizes of 128-bit blocks. This translates to 10 blocks of clear text that is encrypted individually. The key length affects the internal rounds of mathematical operations in AES but it does not affect the eventual size of the output.

E. SELECTION MATRIX

Table 8 summarizes the security properties of the various encryption schemes and their performances for SMS encryption. The aim of the table is to assist application designers in choosing a suitable encryption scheme to encrypt SMS for a particular deployment scenario. It is assumed that the suitability for SMS as a transport mechanism has already been considered.

The next Chapter of the thesis describes a simple chat application that was implemented using only asymmetric encryption to verify the practicality of such an implementation.

	Confidentiality	Integrity	Key Management	Power	Time	Overhead
Symmetric	Good	No. Yes, if used with hashing.	Easy key generation. Difficult key distribution.	Low	Fast	Low
Asymmetric	Good	No. Yes, if used for digital signature	Easy key generation. Easy key distribution but need to ensure authenticity of public key	Acceptable for short key lengths. Depends on usage and accessibility to charging station.	Acceptable for short key lengths	High for long key lengths
OTP	Perfect	No	Difficult key generation. Difficult key distribution.	Very Low	Very fast	No overhead

Table 8. Encryption Scheme Selection Matrix

IV. DEMONSTRATION APPLICATIONS

A. OVERVIEW

The measurement results of the experiment in Chapter III showed that the use of asymmetric encryption for SMS is not prohibitively high in a modern mobile device. The implementation of asymmetric encryption for SMS allows for confidentiality and integrity protection without complex key exchanges, and provides opportunities for many applications requiring secure exchange of SMS messages. This Chapter describes the implementation of a Secure Chat demonstration application that uses asymmetric encryption to encrypt and digitally sign SMS messages.

B. SECURE CHAT

1. Aim

The aim of this application was to verify the feasibility of providing confidentiality and integrity protection for SMS messages by using asymmetric encryption. Observations were also made with regard to the practicality of such an implementation.

2. Security Requirement

Every SMS message sent from the device is digitally signed and encrypted. The messages are decrypted by the recipient and the digital signature is also verified by the recipient to detect any modification of the message.

3. Assumptions and Limitations

The algorithm used is the RSA algorithm provided in the Microsoft Crypto API. RSA was selected because it provided native support for encryption and digital signature. It is assumed that the RSA algorithm with a 1024-bit key length for both encryption and digital signature is sufficient for the required confidentiality and integrity protection.

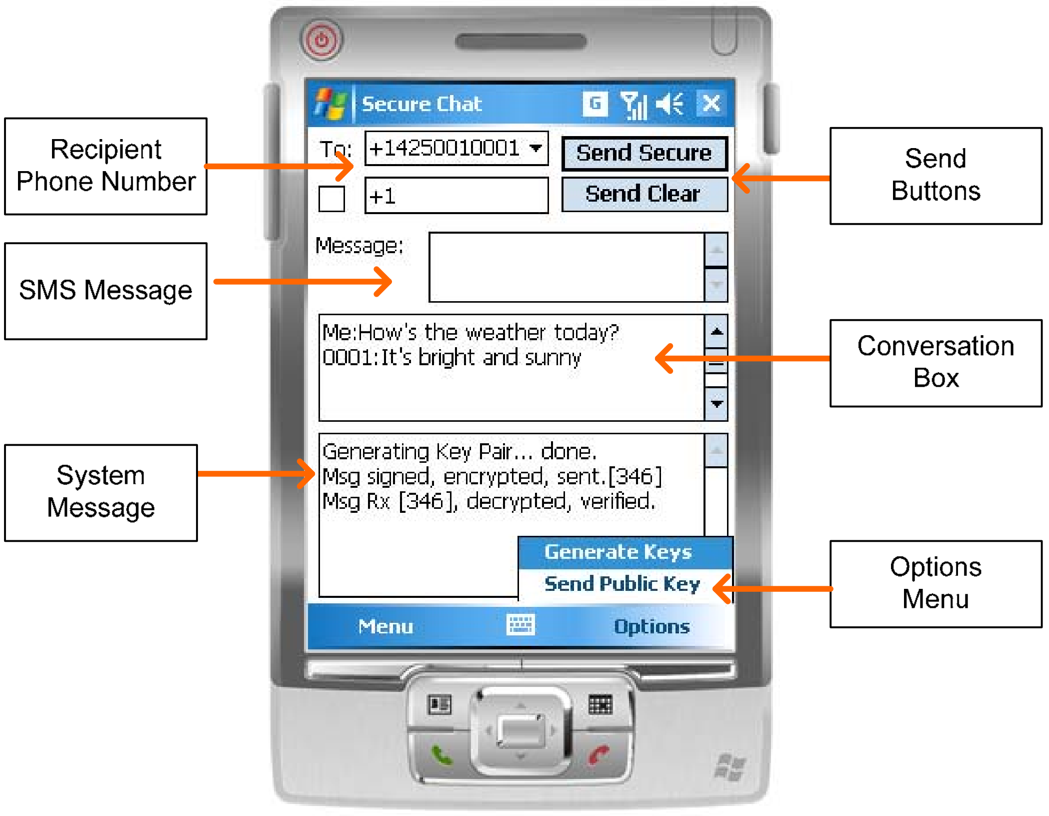
A trusted channel for key exchange is assumed to be available. This could be in the form of physical transfer using SD card or a VPN connection to a trusted server.

5. Design and Implementation

The design of the application adopted a user-centric approach and began with the design of the user interface.

a. User Interface

The main screen of the application is shown in Figure 19.



Me:How's the weather today?
0001:It's bright and sunny

Generating Key Pair... done.
Msg signed, encrypted, sent.[346]
Msg Rx [346], decrypted, verified.

Figure 19. Secure Chat User Interface (Main Screen)

The *Recipient Phone Number* area of the screen is comprised of a drop down combo box that lists available phone numbers and a text box for the user to key in a new recipient number. The checkbox beside the free text box must be checked in order for the application to accept the text box input as the recipient phone number.

There are two *Send Buttons*: one for sending secure messages and one for sending the message in clear. The aim is to provide a single interface if the user needs to send unencrypted messages to parties outside the secure conversation. This option should be removed in more secure applications to prevent the user from accidentally sending the message in clear text. However, all incoming unencrypted messages will be transferred to the default Windows Outlook Mobile, and not be trapped by the Secure Chat application.

The *SMS Message* box allows the user to key in the message to be sent. The maximum length is 117 bytes because that the maximum input length accepted by RSA with a 1024-bit key length. Expanding the length beyond 117 bytes will result in another round of encryption and more overheads. It is assumed that 117 bytes is a sufficient length for the purpose of this demonstration application.

The *Conversation Box* displays the ongoing conversation in a typical chat application. Outbound messages are prefixed by “Me:” and the inbound messages are marked by the last four digits of the sender’s phone number. The user can use the scroll bars to scroll through the history of the conversation.

The *System Messages* text box displays system messages such as key generation status, and the encryption, signature and sending processes.

The *Option Menu* offers two selections for generating RSA Public-Private key pair and for sending the Public Key via SMS. It should be noted that the sending of Public Keys without additional authentication is subject to man-in-the-middle attacks.

b. Program Flow

The flow chart for the application is shown in Figure 20.

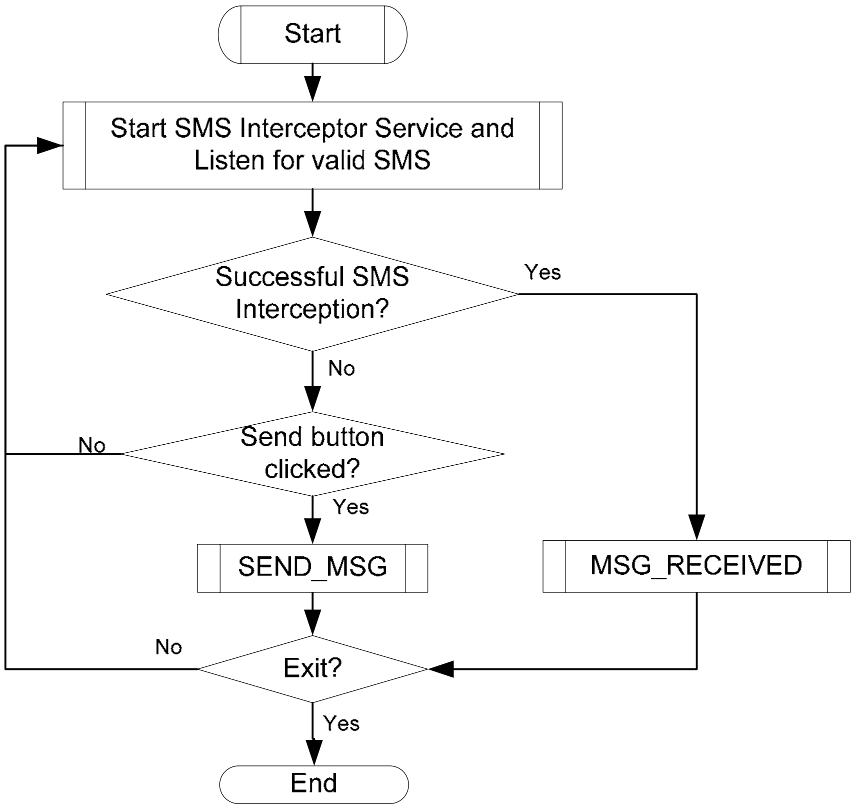
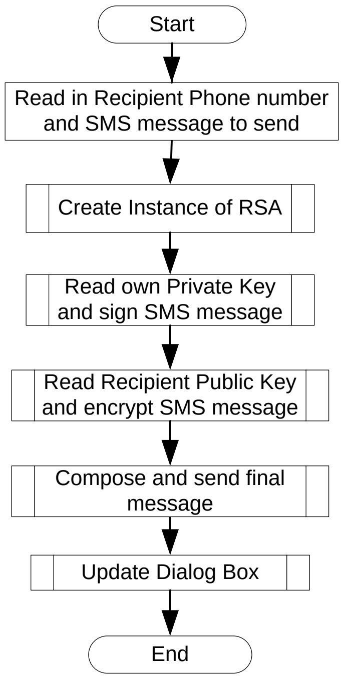

Start
Start SMS Interceptor Service and Listen for valid SMS
Successful SMS Interception?
Yes
No
Send button clicked?
No
Yes
SEND_MSG
MSG_RECEIVED
No
Exit?
Yes
End

Figure 20. Flow Diagram for Main Program

A key requirement in the application is to be able to trap the specially marked incoming SMS messages as it arrives at the cell phone. This service is provided by the SMS Message Interception Service provided by the SNAPI under Windows Mobile 5.0. This service allows developers to selectively intercept SMS messages programmatically. This is especially useful in a Secure Chat application because it allows encrypted messages to be processed and stored separately from normal SMS messages.

For sending encrypted SMS messages, the user selects the recipient phone number, types in the message in to the SMS Message box, and clicks the “Send Secure” button. The SEND_MSG procedure is executed. Figure 21 show the flow diagram of the SEND_MSG procedure.



Start
Read in Recipient Phone number and SMS message to send
Create Instance of RSA
Read own Private Key and sign SMS message
Read Recipient Public Key and encrypt SMS message
Compose and send final message
Update Dialog Box
End

Figure 21. Flow Diagram for SEND_MSG Process

All encrypted SMS are marked with “*” at the beginning. Once an SMS message meeting this criterion is met, the MSG_RECEIVED procedure is activated and the message is processed. The flow diagram for the MSG_RECEIVED procedure is shown in Figure 22.

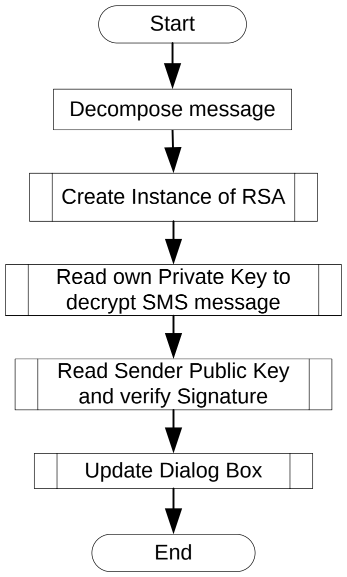

Start
Decompose message
Create Instance of RSA
Read own Private Key to decrypt SMS message
Read Sender Public Key and verify Signature
Update Dialog Box
End

Figure 22. Flow Diagram for MSG_RECEIVED Process

The encryption and decryption processes in the Microsoft .NET Framework make use of the *System.Security.Cryptography* namespace. The *CryptoStream* class is one of the many classes that is provided and is used as a buffer to encrypt and decrypt the content as it is streamed out to a *FileStream* or a *MemoryStream*. The following Section describes in detail the code used for encryption and signing in the SEND_MSG process. Similar steps are used in the MSG_RECEIVED process.

After the appropriate declarations, a new instance of the RSA *CryptoServiceProvider* with 1024 bit key length is created. An instance of the SHA1 hash algorithm was also created to facilitate the digital signing later.

```
RSACryptoServiceProvider TxRSA = new RSACryptoServiceProvider(1024);
SHA1CryptoServiceProvider TxSHA = new SHA1CryptoServiceProvider();
```

The Private Key is read from a key file that has been created earlier using the *Generate Key Pair* function. The Private Key is read as a *FileStream*, converted to a byte array and then imported into the RSA Instance.

```
FileStream TxReadPrivfs = File.OpenRead("Program Files\\SecureChat\\" +
MyPhoneNumber + ".prv");
BinaryReader TxReadPrivbr = new BinaryReader(TxReadPrivfs);
TxPrivKeyBlob = TxReadPrivbr.ReadBytes(596);
TxReadPrivbr.Close();
TxReadPrivfs.Close();
TxRSA.ImportCspBlob(TxPrivKeyBlob);
```

A hash is created using the SHA1 algorithm and the hashed data is encrypted with the RSA algorithm using the sender's Private Key.

```
Signature = TxRSA.SignData(dataToEncrypt, TxSHA);
```

The recipient's Public Key is read from the key file and imported into the RSA instance.

```
FileStream TxReadPubfs = File.OpenRead("Program Files\\SecureChat\\" +
ToPhoneNumber + ".pub");
BinaryReader TxReadPubbr = new BinaryReader(TxReadPubfs);
TxPubKeyBlob = TxReadPubbr.ReadBytes(148);
TxReadPubbr.Close();
TxReadPubfs.Close();
TxRSA.ImportCspBlob(TxPubKeyBlob);
```

The message is then encrypted by the RSA algorithm using the recipient's Public Key. The Optimal Asymmetric Encryption Padding (OAEP) parameter was set to false because it is not supported under Windows Mobile 5.0.

```
encryptedData = TxRSA.Encrypt(dataToEncrypt, false);
```

The message is finally completed by encoding the encrypted data stream using Base64 encoding and adding a marker in front of the data. The type of encoding used is crucial in ensuring that the encrypted data is accurately encoded as the SMS message undergoes different protocol translations across networks. The “**” is used as the marker to differentiate encrypted data from normal SMS messages. The choice of the marker character is purely arbitrary, as long as the characters are seldom used in normal SMS text exchanges.

```
FinalMsg = "**" +
Convert.ToBase64String(encryptedData)+Convert.ToBase64String(Signature)
```

The SMS sending service in Windows Mobile 5.0 is provided by the *Microsoft.WindowsMobile.PocketOutlook* namespace. A new instance of the *SmsMessage* class is created to send the SMS.

```
SmsMessage MsgToSend = new SmsMessage(ToPhoneNumber, FinalMsg);
MsgToSend.Send();
```

The last stage of the sending process is to update the display to provide feedback to the user as to the status of the sending process. The typed message is moved to the *Conversation Box* to indicate that the message has been sent successfully. The system status box indicates whether the SMS message has been successfully signed, encrypted and sent. The length of message is included as an additional check.

```
this.textBoxDialog.Text += "Me:" + this.textBoxMsgToSend.Text + "\r\n";

// Clear the "Message" edit box
this.textBoxMsgToSend.Text = "";
this.textBoxDump.Text += "sent.[" + FinalMsg.Length.ToString() +
"]\r\n";
```

C. OBSERVATIONS

It was observed that the signing and encryption process was fast from a usability perspective as compared to the time required to send the SMS. In this case, a 1024-bit key length was used for both encryption and signing. The encrypted data produced an output of 172 bytes. The signature is also 172 bytes. This resulted in a final message length of 346 bytes, if the two marker character markers are included. This means that three SMS messages are required to send the final message.

The waiting time for sending three SMS messages appeared very long as compared to the encryption processes, probably because the system status message is only updated when all three messages have been sent. During the process, nothing is seen to be happening. This may be unnerving for some users because the time taken to send a message is now significantly longer than sending a normal unencrypted message.

To improve the user interface, more feedbacks could be provided to the user with regard to the sending progress. The other way is to reduce the signature length by using a shorter key length. For example, an 840-bit key length will produce a 140 byte signature. This will reduce the total length of the message to 314 bytes, which can be sent with two SMS messages.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSION

In this thesis, a detailed vulnerability study was conducted on the GSM Network and the SMS protocol. It was concluded that application layer encryption is required to protect the confidentiality and integrity of SMS messages. A study of the different encryption schemes was conducted to understand their properties. Several considerations were drawn up with regard to the implementation of an encryption scheme on a mobile device. Of particular concern are the efficiency and power consumption requirements of encryption operations. Therefore, empirical measurements were taken to compare the performances of symmetric and asymmetric encryption on a modern mobile phone device. It was discovered that asymmetric encryption for SMS is no longer prohibitively expensive in terms of timing and resource consumption in a modern mobile device, due to advances in CPU performance for mobile devices.

A demonstration Secure Chat application was developed to validate the feasibility of implementing a pure asymmetric encryption solution for SMS. It was discovered that the overheads generated by asymmetric encryption is a key factor in deciding the suitability of asymmetric encryption. The long output lengths generated by an asymmetric encryption algorithm resulted in messages that are many times longer than the original message. For SMS, this is significant because the low bandwidth is further exacerbated.

B. RECOMMENDATIONS

The SMS and GSM technologies are matured after being in operation for more than ten years. Although they are not secure by design and implementation, their pervasiveness and low cost may be leveraged to improve other aspects of security. Following are several areas of potential exploration which may prove to increase the utility of the SMS protocol for sensitive communications.

1. Remote Device Termination by SMS

By using SMS interception, a remote device that is physically lost may be remotely locked and the contents encrypted to prevent loss of sensitive information such as address book and personal information. However, the address book and sensitive information such as emails are locked for access by the Pocket Outlook application. The challenge will be to explore ways in which the information can be accessed and encrypted.

2. One Time Pad (OTP) for SMS Encryption

The advantages and possibility of using OTP for SMS encryption was discussed in Chapter III. This possibility can be further explored. The key research area would be to design an architecture for the key management and key synchronization for a OTP encryption scheme.

3. SMS-based Two-factor Authentication

Some banks are already using SMS as an additional authentication mechanism for online banking. This idea could be further extended by using the cell phone as the second factor of authentication. The cell phone is connected to the laptop via Bluetooth and the laptop is connected to the server via Internet. A challenge and response authentication mechanism can be built such that either the challenge, or the response information is sent via SMS through the cell phone, and the information is relayed to the laptop. The sending of the challenge and response on different channels makes it virtually impossible for the attacker to conduct a man-in-the-middle attack. The attacker has to be able to monitor, correlate and respond on two channels in order to carry out the attack. For the user, it is a two-factor authentication. If the laptop is lost, access to the server is denied even if the attacker has the password. The key research question is the synchronization and the timing requirements for such a setup.

4. SMS Blue Force Tracking (Personnel)

Personnel tracking in an urban area is difficult because the “concrete jungle” is not a favorable environment for radio frequency propagation. GPS information may also be affected due to lack of line-of-sight to satellites. Research has shown that a pure client-based GSM localization system can achieve median localization accuracies of 5 and 75 meters for indoor and outdoor environments, respectively [37]. If the information can be further correlated with the Base Transceiver Stations (BTS), better accuracies may be achieved. Currently small portable GSM BTS are commercially available that can be set up quickly. If a few of such BTS can be set up near the area of operations, the mini-GSM network can be used to provide positional information for individual soldiers. The information can then be sent back to the Command Post to provide a situation of the Blue Force disposition. The key research question is the positional accuracy that can be achieved using portable GSM BTS.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. POWER CONSUMPTION EXPERIMENT

A. PRE-MEASUREMENT CHECKLIST

The following settings are checked prior to each measurement to ensure consistency of measurements:

Feature	Setting
Wifi	OFF
GSM	OFF
Bluetooth	OFF
Active Processes	File Explorer is the only process running. This is because File Explorer is required to start the application
LCD Backlight Setting: Battery Power	*"Turn off backlight if device is not used for "* – 30 sec
LCD Backlight Setting: Battery Power	*"Turn on backlight when a button is pressed or the screen is tapped"* – Enabled
LCD Backlight: Brightness Level	6 / 10
	"Auto adjust backlight level by battery level" – Disabled
	"Auto adjust backlight level by idle time" - Disabled
Power Management: Sleep Mode Settings: On Battery Power	*"Turn off device if not used for"* - Disabled
Power Management: Sleep Mode Settings: On External Power	*"Turn off device if not used for"* - Disabled

Table 9. Pre-Measurement Checklist

B. FAILED APPROACHES

Figure 23 illustrates the logic associated with the original intended program flow. However, the program flow below cannot be implemented due to certain software limitations, which will be discussed in the following Sections.

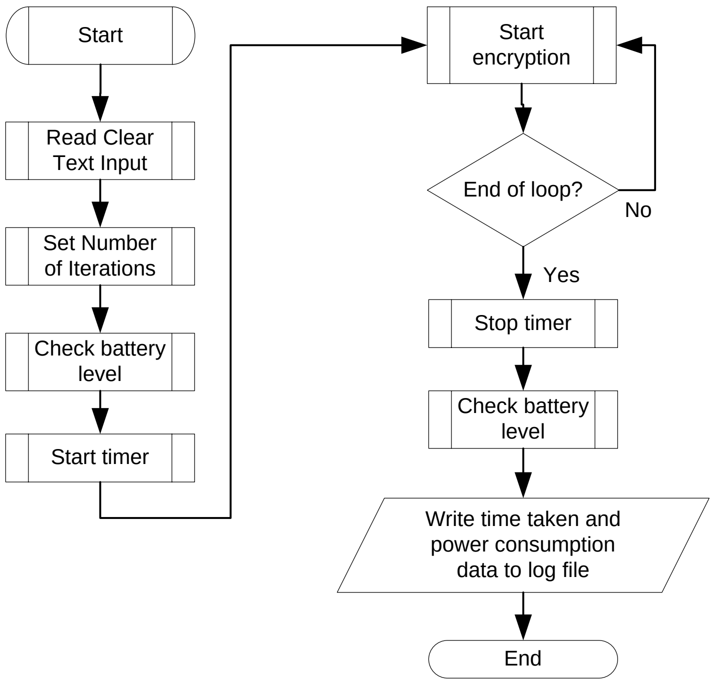


Figure 23. Flowchart for Performance Measurement (Original Approach)

1. Power Measurement Resolution

The battery power level was accessed through the SystemState.PowerBatteryStre*ngth* property using the State and Notification API (SNAPI). However, the returned value was expressed as power levels in 5 distinct bands: Very Low (0-20%), Low (21%-40%), Medium (41%-60%), High (61%-80%), Very High (81-100%). This resolution was clearly insufficient and another approach was required.

2. Failure in SNAPI Notification Service

Another approach was adopted to make use of the notification feature of the SNAPI to detect the changes as the battery levels as it changes from one band to another. By noting the number of iterations of encryption that causes battery level to change by a 20 per cent range, the consumption figure for each iteration of encryption can be approximated. The program was coded according to the flow chart in Figure 25.

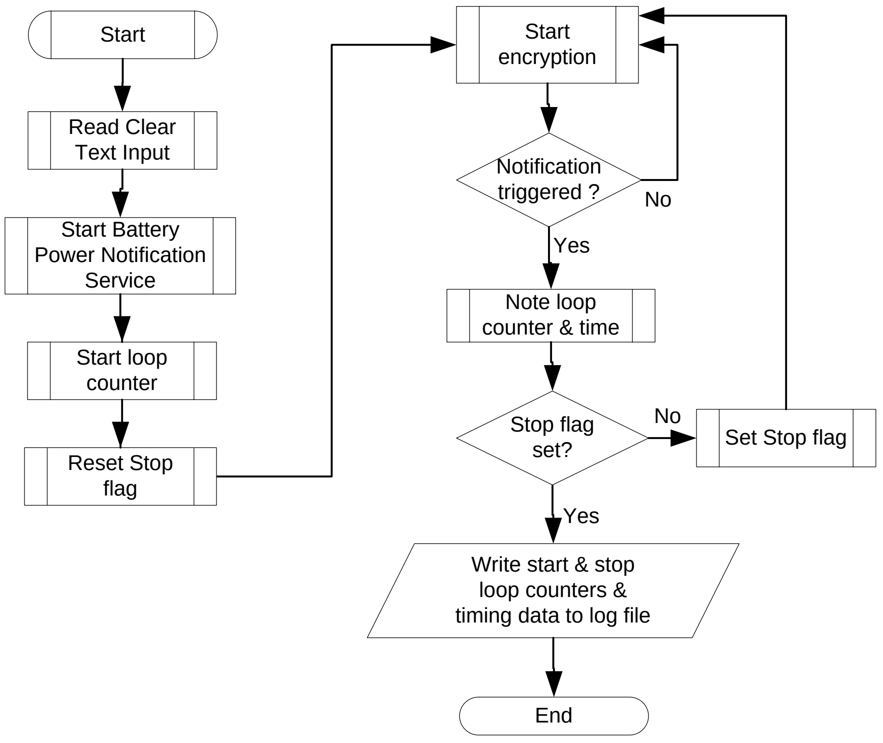


Figure 24. Flowchart for Performance Measurement (Second Approach)

However, this approached also failed because the Notification Service in the SNAPI failed to trap the power changes while the application was executing the loop. As the result, the program executed till the end of the loop without any interruptions and all the system notifications appear after the loop termination.

C. RESULTS

The results of the baseline measurement are shown in Table 10. The results of the RSA performance measurements are shown in Tables 11 and 12. The results of the AES performance measurements are shown in Tables 13 and 14.

S/No	Iteration Number		Time		Duration	Time/ 100kb	Consumption
	Start	**End**	**Start**	**End**	**(ms)**	**(ms)**	**(mAH)**
1	43682	83792	4931000	9591000	4660000	116.2	0.718
2	83792	118124	9591000	15994000	6403000	186.5	0.344
3	118124	175326	15994000	23692000	7698000	134.6	0.503
4	80408	107952	9242000	15603000	6361000	230.9	1.046
5	107952	136765	15603000	21262000	5659000	196.4	0.267
6	83340	124493	9241000	14062000	4821000	117.1	0.700
7	65970	105149	7938000	12779000	4841000	123.6	0.735
8	105149	137708	12779000	17158000	4379000	134.5	0.274
9	57900	90260	8116000	12757000	4641000	143.4	0.890
10	90260	129587	12757000	18477000	5720000	145.4	0.319
11	52458	89864	6305000	10924000	4619000	123.5	0.770
12	89864	122644	10924000	15085000	4161000	126.9	0.320
13	817	43773	87000	4727000	4640000	108.0	0.670
14	43773	82268	4727000	8967000	4240000	110.1	0.658
15	73978	114330	8073000	12714000	4641000	115.0	0.714
16	114330	151178	12714000	17053000	4339000	117.8	0.252
17	7242	49880	776000	5418000	4642000	108.9	0.675
18	49880	88666	5418000	9738000	4320000	111.4	0.577
19	76174	116348	8344000	12983000	4639000	115.5	0.717
20	69876	110292	7687000	12387000	4700000	116.3	0.713
21	110292	153096	12387000	17466000	5079000	118.7	0.261
22	3538	44852	387000	5008000	4621000	111.9	0.697
23	44852	80860	5008000	9147000	4139000	114.9	0.642
24	80860	144264	9147000	16768000	7621000	120.2	0.454
25	20004	61282	2189000	6829000	4640000	112.4	0.698
26	61282	99840	6829000	11310000	4481000	116.2	0.470
27	99840	164968	11310000	19170000	7860000	120.7	0.442
28	52493	89505	6432000	11073000	4641000	125.4	0.778
29	89505	123909	11073000	15453000	4380000	127.3	0.322
30	123909	190592	15453000	24159000	8706000	130.6	0.432
31	55518	94798	6260000	10900000	4640000	118.1	0.733
32	94798	132716	10900000	15499000	4599000	121.3	0.304
33	132716	197653	15499000	23579000	8080000	124.4	0.444
					Mean	**128.61**	**0.56**
					Std Dev	**26.65**	**0.21**

Table 10. Baseline Measurements

S/No	**Iteration Number**		**Time**		**Duration**	**Time/ 100kb**	**Consumption**
	Start	**End**	**Start**	**End**	**(ms)**	**(ms)**	**(mAH)**
1	152	447	2478000	7303000	4825000	16355.9	97.627
2	567	851	9249000	13873000	4624000	16281.7	101.408
3	851	1074	13873000	17501000	3628000	16269.1	129.148
4	59	334	974000	5593000	4619000	16796.4	104.727
5	334	567	5593000	9209000	3616000	15519.3	123.605
6	570	867	9265000	14086000	4821000	16232.3	96.970
7	867	1077	14086000	17493000	3407000	16223.8	137.143
8	162	459	2648000	7473000	4825000	16245.8	96.970
9	459	711	7473000	11567000	4094000	16246.0	114.286
10	583	881	9449000	14273000	4824000	16187.9	96.644
11	881	1107	14273000	17932000	3659000	16190.3	127.434
12	49	347	811000	5637000	4826000	16194.6	96.644
13	347	582	5637000	9444000	3807000	16200.0	122.553
14	108	406	1763000	6586000	4823000	16184.6	96.644
15	406	629	6586000	10195000	3609000	16183.9	129.148
16	505	803	8203000	13037000	4834000	16221.5	96.644
17	803	1024	13037000	16621000	3584000	16217.2	130.317
18	595	893	9651000	14477000	4826000	16194.6	96.644
19	893	1172	14477000	18997000	4520000	16200.7	103.226
20	300	599	4868000	9707000	4839000	16183.9	96.321
21	599	851	9707000	13788000	4081000	16194.4	114.286
22	659	956	10690000	15502000	4812000	16202.0	96.970
23	956	1292	15502000	20947000	5445000	16205.4	85.714
24	317	614	5165000	9989000	4824000	16242.4	96.970
25	614	825	9989000	13420000	3431000	16260.7	136.493
26	639	935	10372000	15167000	4795000	16199.3	97.297
27	935	1262	15167000	20464000	5297000	16198.8	88.073
28	250	549	4062000	8903000	4841000	16190.6	96.321
29	549	760	8903000	12320000	3417000	16194.3	136.493
30	285	582	4636000	9453000	4817000	16218.9	96.970
31	582	905	9453000	14693000	5240000	16222.9	89.164
32	681	979	11021000	15840000	4819000	16171.1	96.644
33	979	1252	15840000	20256000	4416000	16175.8	105.495
					Mean	**16212.31**	**107.00**
					Std Dev	**164.44**	**15.77**

Table 11. RSA 1024-bit key length (Block Size 117 bytes) performance data

S/No	Iteration Number		Time		Duration	Time/ 100kb	Consumption
	Start	End	Start	End	(ms)	(ms)	(mAH)
1	95	275	2468000	7088000	4620000	25666.7	160.000
2	275	451	7088000	11608000	4520000	25681.8	163.636
3	451	727	11608000	18704000	7096000	25710.1	104.348
4	284	464	7321000	11934000	4613000	25627.8	160.000
5	464	605	11934000	15548000	3614000	25631.2	204.255
6	605	829	15548000	21287000	5739000	25620.5	128.571
7	305	485	7855000	12472000	4617000	25650.0	160.000
8	485	626	12472000	16088000	3616000	25645.4	204.255
9	626	854	16088000	21943000	5855000	25679.8	126.316
10	132	314	3400000	8050000	4650000	25549.5	158.242
11	314	456	8050000	11678000	3628000	25549.3	202.817
12	456	721	11678000	18457000	6779000	25581.1	108.679
13	152	305	3919000	7840000	3921000	25627.5	188.235
14	305	546	7840000	14020000	6180000	25643.2	119.502
15	58	237	1523000	6121000	4598000	25687.2	160.894
16	237	388	6121000	10003000	3882000	25708.6	190.728
17	388	646	10003000	16639000	6636000	25720.9	111.628
18	367	548	9456000	14097000	4641000	25640.9	159.116
19	548	689	14097000	17714000	3617000	25652.5	204.255
20	689	909	17714000	23361000	5647000	25668.2	130.909
21	355	536	9129000	13772000	4643000	25651.9	159.116
22	536	676	13772000	17362000	3590000	25642.9	205.714
23	676	909	17362000	23341000	5979000	25660.9	123.605
24	158	299	4077000	7693000	3616000	25645.4	204.255
25	299	551	7693000	14159000	6466000	25658.7	114.286
26	349	530	8971000	13607000	4636000	25613.3	159.116
27	530	676	13607000	17346000	3739000	25609.6	197.260
28	676	936	17346000	24002000	6656000	25600.0	110.769
29	72	252	1870000	6483000	4613000	25627.8	160.000
30	252	400	6483000	10277000	3794000	25635.1	194.595
	400	655	10277000	16818000	6541000	25651.0	112.941
					Mean	**25643.18**	**157.68**
					Std Dev	**40.16**	**34.96**

Table 12. RSA 2048-bit key length (Block Size 117 bytes) performance data

S/No	**Iteration Number**		**Time**		**Duration**	**Time/ 100kb**	**Consumption**
	Start	**End**	**Start**	**End**	**(ms)**	**(ms)**	**(mAH)**
1	427	796	5354000	9969000	4615000	12506.8	78.049
2	796	1085	9969000	13583000	3614000	12505.2	99.654
3	1085	1332	13583000	16672000	3089000	12506.1	116.599
4	98	389	1239000	4860000	3621000	12443.3	98.969
5	389	747	4860000	9319000	4459000	12455.3	80.447
6	637	1007	7989000	12622000	4633000	12521.6	77.838
7	1007	1296	12622000	16241000	3619000	12522.5	99.654
8	1296	1565	16241000	19611000	3370000	12527.9	107.063
9	561	929	7052000	11662000	4610000	12527.2	78.261
10	929	1218	11662000	15283000	3621000	12529.4	99.654
11	1218	1427	15283000	17903000	2620000	12535.9	137.799
12	719	1089	9006000	13632000	4626000	12502.7	77.838
13	1089	1378	13632000	17244000	3612000	12498.3	99.654
14	1378	1609	17244000	20133000	2889000	12506.5	124.675
15	15	386	212000	4847000	4635000	12493.3	77.628
16	386	679	4847000	8510000	3663000	12501.7	98.294
17	679	1091	8510000	13662000	5152000	12504.9	69.903
18	594	1047	6076000	10700000	4624000	10207.5	63.576
19	1047	1421	10700000	14517000	3817000	10205.9	77.005
20	1421	2089	14517000	21335000	6818000	10206.6	43.114
21	898	1352	9156000	13778000	4622000	10180.6	63.436
22	1352	1767	13778000	18001000	4223000	10175.9	69.398
23	1767	2445	18001000	24904000	6903000	10181.4	42.478
24	885	1338	9045000	13664000	4619000	10196.5	63.576
25	1338	1697	13664000	17323000	3659000	10192.2	80.223
26	1697	2388	17323000	24367000	7044000	10193.9	41.679
27	406	761	4153000	7769000	3616000	10185.9	81.127
28	761	1429	7769000	14572000	6803000	10184.1	43.114
29	829	1283	8468000	13084000	4616000	10167.4	63.436
30	1283	1639	13084000	16704000	3620000	10168.5	80.899
31	1639	2272	16704000	23147000	6443000	10178.5	45.498
					Mean	**11458.50**	**80.02**
					Std Dev	**1172.65**	**24.34**

Table 13. RSA 2048-bit key length (Block Size 245 bytes) performance data

S/No	Iteration Number		Time		Duration	Time/ 100kb	Consumption
	Start	**End**	**Start**	**End**	**(ms)**	**(ms)**	**(mAH)**
1	18133	29156	9588000	15448000	5860000	531.6	2.613
2	29156	40045	15448000	21288000	5840000	536.3	2.645
3	40045	56872	21288000	30347000	9059000	538.4	1.712
4	16904	27734	8977000	14758000	5781000	533.8	2.659
5	27734	37644	14758000	20098000	5340000	538.8	2.906
6	37644	54905	20098000	29437000	9339000	541.0	1.669
7	11818	21695	6267000	11526000	5259000	532.4	2.916
8	21695	32419	11526000	17266000	5740000	535.2	2.686
9	32419	48912	17266000	26164000	8898000	539.5	1.746
10	19087	30221	10180000	16160000	5980000	537.1	2.587
11	30221	41359	16160000	22201000	6041000	542.4	2.586
12	41359	57622	22201000	31060000	8859000	544.7	1.771
13	19244	29995	10208000	15948000	5740000	533.9	2.679
14	29995	41360	15948000	22068000	6120000	538.5	2.534
15	11771	22799	6213000	12053000	5840000	529.6	2.612
16	22799	34265	12053000	18213000	6160000	537.2	2.512
17	34265	52263	18213000	27850000	9637000	535.4	1.600
18	19041	30461	10110000	16209000	6099000	534.1	2.522
19	30461	41140	16209000	21969000	5760000	539.4	2.697
20	41140	57323	21969000	30709000	8740000	540.1	1.780
21	4938	16042	2611000	8511000	5900000	531.3	2.594
22	16042	24714	8511000	13132000	4621000	532.9	3.321
23	24714	42192	13132000	22532000	9400000	537.8	1.648
24	19409	30121	10269000	15969000	5700000	532.1	2.689
25	30121	40626	15969000	21609000	5640000	536.9	2.742
26	40626	58036	21609000	30987000	9378000	538.7	1.654
27	77	7167	40000	3803000	3763000	530.7	4.062
28	7167	24327	3803000	12983000	9180000	535.0	1.678
29	18064	28927	9611000	15431000	5820000	535.8	2.651
30	28927	40416	15431000	21632000	6201000	539.7	2.507
31	40416	56692	21632000	30452000	8820000	541.9	1.769
					Mean	**536.53**	**2.41**
					Std Dev	**3.74**	**0.57**

Table 14. AES 128-bit key length performance data

S/No	**Iteration Number**		**Time**		**Duration**	**Time/ 100kb**	**Consumption**
	Start	**End**	**Start**	**End**	**(ms)**	**(ms)**	**(mAH)**
1	17281	27538	10188000	16287000	6099000	594.6	2.808
2	27538	36329	16287000	21527000	5240000	596.1	3.276
3	10136	19033	5949000	11189000	5240000	589.0	3.237
4	19033	26385	11189000	15529000	4340000	590.3	3.917
5	17530	28232	10243000	16524000	6281000	586.9	2.691
6	28232	37855	16524000	22183000	5659000	588.1	2.993
7	16135	26423	9376000	15375000	5999000	583.1	2.799
8	26423	38162	15375000	22235000	6860000	584.4	2.453
9	18174	28065	10589000	16389000	5800000	586.4	2.912
10	28065	39241	16389000	22969000	6580000	588.8	2.577
11	17655	28125	10285000	16406000	6121000	584.6	2.751
12	28125	37310	16406000	21786000	5380000	585.7	3.136
13	17395	30272	10125000	17672000	7547000	586.1	2.237
14	30272	37627	17672000	22003000	4331000	588.9	3.916
15	8713	17713	5056000	10295000	5239000	582.1	3.200
16	17713	26078	10295000	15175000	4880000	583.4	3.443
17	15222	26039	8944000	15325000	6381000	589.9	2.662
18	26039	34803	15325000	20503000	5178000	590.8	3.286
19	15192	26443	8824000	15383000	6559000	583.0	2.560
20	26443	36196	15383000	21084000	5701000	584.5	2.953
21	17779	27824	10359000	16259000	5900000	587.4	2.867
22	27824	38035	16259000	22280000	6021000	589.7	2.820
23	13456	23916	7818000	13917000	6099000	583.1	2.753
24	23916	34522	13917000	20117000	6200000	584.6	2.715
25	8027	16554	4654000	9613000	4959000	581.6	3.378
26	16554	28627	9613000	16653000	7040000	583.1	2.385
27	17659	28183	10274000	16435000	6161000	585.4	2.737
28	28183	38155	16435000	22296000	5861000	587.7	2.888
29	15800	26946	9177000	15678000	6501000	583.3	2.584
30	26946	37758	15678000	21997000	6319000	234.5	1.069
31	17558	27870	10215000	16256000	6041000	585.8	2.793
32	27870	38002	16256000	22215000	5959000	213.8	1.033
					Mean	**563.95**	**2.81**
					Std Dev	**89.24**	**0.60**

Table 15. AES 256-bit key length performance data

## D.	PROGRAM CODE

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Security.Cryptography;
using Microsoft.WindowsMobile.Status;

namespace PowerEncryption
{
    public partial class MainForm : Form
    {
        public MainForm()
        {
            InitializeComponent();
        }

        private void menuExit_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void MainForm_Load(object sender, EventArgs e)
        {
            this.menuBaseline.Checked = true;
            this.textBoxCurrentMode.Text = "RSA";
            comboBoxClearTextLen.SelectedItem = "100KB";
            textBoxIterations.Text = "3000";
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int n = Convert.ToInt32(textBoxIterations.Text);
            TimeSpan ts;  // Create a TimeSpan instance

            this.textBoxPowerStart.Text = "";
            this.textBoxDuration.Text = "";
            this.textBoxPowerEnd.Text = "";
            this.textBoxPowerDiff.Text = "";

            if (this.textBoxCurrentMode.Text == "RSA")
            {
                #region RSA

            this.textBoxPowerStart.Text = 
SystemState.GetValue(SystemProperty.PowerBatteryStrength).ToString();
            DateTime dtStart = DateTime.Now; //Capture Start time
            this.textBoxDump.Text = this.textBoxDump.Text + "Encrypt using RSA... ";

            //Create byte arrays to hold original, encrypted data.
            byte[] RSAClrData;
            byte[] RSAEncryptedData;
             //Create a new instance of RSACryptoServiceProvider to generate public and 
private key data.
            RSACryptoServiceProvider RSAKey = new RSACryptoServiceProvider(2048);
            this.textBoxKeyLength.Text = RSAKey.KeySize.ToString();

            for (int i = 0; i < n; i++)
            {
               //Read input file
                FileStream RSAinStream = File.Open("Temp\\" + 
comboBoxClearTextLen.SelectedItem.ToString() + ".txt", FileMode.Open);
                StreamReader RSAsReader = new StreamReader(RSAinStream);
                string RSAClrTxt = RSAsReader.ReadToEnd();
                System.Text.Encoding ByteEnc = System.Text.Encoding.ASCII;

                    //Create a new instance of RSACryptoServiceProvider.
                    RSACryptoServiceProvider RSA = new RSACryptoServiceProvider(2048);

                    //Import the RSA Key information.
                    RSA.ImportParameters(RSAKey.ExportParameters(false));

                    System.Text.Encoding enc = System.Text.Encoding.ASCII;

                    string RSAEncryptedTxt="";

                //Encrypt individual blocks of data.
                //Max input for 1024-bit key length is 117bytes.
                //Max input for 2048-bit key length is 245bytes

                    int j = 0;
                int MaxInput = 245;
                    while ( j+MaxInput < RSAClrTxt.Length)
                    {
                       RSAClrData = ByteEnc.GetBytes(RSAClrTxt.ToCharArray(),j,MaxInput);
                        RSAEncryptedData = RSA.Encrypt(RSAClrData, false);
                        RSAEncryptedTxt += Convert.ToBase64String(RSAEncryptedData);
                        j = j + MaxInput;
                    }

                //Encrypt last block
                    RSAClrData = ByteEnc.GetBytes(RSAClrTxt.ToCharArray(), j,
(RSAClrTxt.Length - j));

                RSAEncryptedData = RSA.Encrypt(RSAClrData, false);
                    string temptxt =  Convert.ToBase64String(RSAEncryptedData);
                    RSAEncryptedTxt += temptxt;

                //Write encrypted data to file
                    FileStream RSAoutStream = File.Open("Temp\\encrypted.txt",
FileMode.Create);
                    StreamWriter RSAWriter = new StreamWriter(RSAoutStream);
                    RSAWriter.Write(RSAEncryptedTxt);

                RSAWriter.Close();
                RSAoutStream.Close();
                RSAinStream.Close();

           //Get time and power settings after encryption and calculate difference

                DateTime dtEnd = DateTime.Now;
            textBoxPowerEnd.Text =
SystemState.GetValue(SystemProperty.PowerBatteryStrength).ToString();

            //Calculate Duration
            ts = dtEnd.Subtract(dtStart).Duration();
            textBoxDuration.Text = ts.TotalMilliseconds.ToString();

            //Calculate Power Consumption
            int PwrDiff = Convert.ToInt32(textBoxPowerStart.Text) -
Convert.ToInt32(textBoxPowerEnd.Text);
            textBoxPowerDiff.Text = Convert.ToString(PwrDiff) + "%";

            // Write readings to log file

            StreamWriter logStreamWriter = null;
            try
            {
                string time = DateTime.Now.ToString();
                // Create a StreamWriter using a static File class.
                logStreamWriter = File.AppendText("Temp\\RSAlog.txt");
                logStreamWriter.Write(i.ToString());
                logStreamWriter.Write("\t");
                logStreamWriter.Write(this.textBoxCurrentMode.Text);
                logStreamWriter.Write("\t");
                logStreamWriter.Write(this.textBoxKeyLength.Text);
                logStreamWriter.Write("\t");
                logStreamWriter.Write(this.comboBoxClearTextLen.Text);
                logStreamWriter.Write("\t");
                logStreamWriter.Write(this.textBoxIterations.Text);
                logStreamWriter.Write("\t\t");
                logStreamWriter.Write(this.textBoxDuration.Text);
                logStreamWriter.Write("\t\t");
                logStreamWriter.Write(this.textBoxPowerStart.Text);
                logStreamWriter.Write("\t\t");
                logStreamWriter.Write(this.textBoxPowerEnd.Text);
                logStreamWriter.Write("\t\t");
                logStreamWriter.Write(RSAEncryptedTxt.Length.ToString());
                logStreamWriter.Write("\r\n");

                logStreamWriter.Flush();
            }
            catch (Exception exc)
            {
                // Show the error to the user.
                MessageBox.Show("File could not be created or written to. Exception: " +
exc.Message);
            }
            finally
            {
                // Close the object if it has been created.
                if (logStreamWriter != null)
                {
                    logStreamWriter.Close();
                }
            }


            }
                #endregion
            this.textBoxDump.Text += "done.\r\n";
            }


            if (this.textBoxCurrentMode.Text == "Baseline")
            {

                //Capture Start time and Power Level
                this.textBoxPowerStart.Text =
SystemState.GetValue(SystemProperty.PowerBatteryStrength).ToString();
                DateTime dtStart = DateTime.Now;

                this.textBoxDump.Text = this.textBoxDump.Text + "Encrypt using AES... ";

                for (int i = 0; i < n; i++)
                {
                    FileStream AESinStream = File.Open("Temp\\" +
comboBoxClearTextLen.SelectedItem.ToString() + ".txt", FileMode.Open);
                    StreamReader AESsReader = new StreamReader(AESinStream);
                    string AESClrTxt = AESsReader.ReadToEnd();

                    // Create or open the output file.
                    FileStream AESoutStream = File.Open("Temp\\encrypted.txt",
FileMode.OpenOrCreate);

                    AESsReader.Close();
                    AESoutStream.Close();
                    AESinStream.Close();

                    // Get end time and power level
                    DateTime dtEnd = DateTime.Now;
                    textBoxPowerEnd.Text =
SystemState.GetValue(SystemProperty.PowerBatteryStrength).ToString();

                    //Calculate Duration
                    ts = dtEnd.Subtract(dtStart).Duration();
                    textBoxDuration.Text = ts.TotalMilliseconds.ToString();

                    //Calculate Power Consumption
                    int PwrDiff = Convert.ToInt32(textBoxPowerStart.Text) -
Convert.ToInt32(textBoxPowerEnd.Text);
                    textBoxPowerDiff.Text = Convert.ToString(PwrDiff) + "%";

                    // Write readings to AES log file

                    StreamWriter logStreamWriter = null;
                    try
                    {
                        logStreamWriter = File.AppendText("Temp\\Baselinelog.txt");
                        logStreamWriter.Write(i.ToString());
                        logStreamWriter.Write("\t");
                        logStreamWriter.Write(this.textBoxCurrentMode.Text);
                        logStreamWriter.Write("\t");
                        logStreamWriter.Write(this.textBoxKeyLength.Text);
                        logStreamWriter.Write("\t");
                        logStreamWriter.Write(this.comboBoxClearTextLen.Text);
                        logStreamWriter.Write("\t");
                        logStreamWriter.Write(this.textBoxIterations.Text);
                        logStreamWriter.Write("\t\t");
                        logStreamWriter.Write(this.textBoxDuration.Text);
                        logStreamWriter.Write("\t\t");
                        logStreamWriter.Write(this.textBoxPowerStart.Text);
                        logStreamWriter.Write("\t\t");
                        logStreamWriter.Write(this.textBoxPowerEnd.Text);
                        logStreamWriter.Write("\t\t");
                        //logStreamWriter.Write(AEScStream.Length.ToString());
                        logStreamWriter.Write("\r\n");
                        logStreamWriter.Flush();
                    }
                    catch (Exception exc)
                    {
                        // Show the error to the user.
                        MessageBox.Show("File could not be created or written to.
Exception: " + exc.Message);
                    }
                    finally
                    {
                        // Close the object if it has been created.
                        if (logStreamWriter != null)
                        {
                            logStreamWriter.Close();
                        }
                    }

                }
                this.textBoxDump.Text += "done.\r\n";
            }



            if (this.textBoxCurrentMode.Text == "AES")
            {

                //Capture Start time and Power Level
                this.textBoxPowerStart.Text =
SystemState.GetValue(SystemProperty.PowerBatteryStrength).ToString();
                DateTime dtStart = DateTime.Now;

                this.textBoxDump.Text = this.textBoxDump.Text + "Encrypt using AES... ";
                for (int i = 0; i < n; i++)
                {
                    FileStream AESinStream = File.Open("Temp\\" +
comboBoxClearTextLen.SelectedItem.ToString() + ".txt", FileMode.Open);
                    StreamReader AESsReader = new StreamReader(AESinStream);
                    string AESClrTxt = AESsReader.ReadToEnd();

                    // Create or open the output file.
                    FileStream AESoutStream = File.Open("Temp\\encrypted.txt",
FileMode.OpenOrCreate);

                    //Create a new instance to create keys
                    RijndaelManaged AESalg = new RijndaelManaged();
                    byte[] AESabytIV = AESalg.IV;
                    byte[] AESabytKey = AESalg.Key;
                    this.textBoxKeyLength.Text = AESalg.KeySize.ToString();

                    // Create a CryptoStream using the FileStream and the key and
initialization vector (IV).
                    CryptoStream AEScStream = new CryptoStream(AESoutStream,
                    new RijndaelManaged().CreateEncryptor(AESabytKey, AESabytIV),
                    CryptoStreamMode.Write);

                    // Create a StreamWriter using the CryptoStream.
                    StreamWriter AESsWriter = new StreamWriter(AEScStream);

                    // Write the data to the stream to encrypt it.
                    AESsWriter.WriteLine(AESClrTxt);


                    AESsReader.Close();
                    AESsWriter.Close();
                    AESoutStream.Close();
                    AESinStream.Close();
                    AEScStream.Close();

                    DateTime dtEnd = DateTime.Now;
                    textBoxPowerEnd.Text =
SystemState.GetValue(SystemProperty.PowerBatteryStrength).ToString();

                    //Calculate Duration
                    ts = dtEnd.Subtract(dtStart).Duration();
                    textBoxDuration.Text = ts.TotalMilliseconds.ToString();

                    //Calculate Power Consumption
                    int PwrDiff = Convert.ToInt32(textBoxPowerStart.Text) -
Convert.ToInt32(textBoxPowerEnd.Text);
                    textBoxPowerDiff.Text = Convert.ToString(PwrDiff) + "%";

                    // Write readings to AES log file

                    StreamWriter logStreamWriter = null;
                    try
                    {
                        logStreamWriter = File.AppendText("Temp\\AESlog.txt");
                        logStreamWriter.Write(i.ToString());
                        logStreamWriter.Write("\t");
                        logStreamWriter.Write(this.textBoxCurrentMode.Text);
                        logStreamWriter.Write("\t");
                        logStreamWriter.Write(this.textBoxKeyLength.Text);
                        logStreamWriter.Write("\t");
                        logStreamWriter.Write(this.comboBoxClearTextLen.Text);
                        logStreamWriter.Write("\t");
                        logStreamWriter.Write(this.textBoxIterations.Text);
                        logStreamWriter.Write("\t\t");
                        logStreamWriter.Write(this.textBoxDuration.Text);
                        logStreamWriter.Write("\t\t");
                        logStreamWriter.Write(this.textBoxPowerStart.Text);
                        logStreamWriter.Write("\t\t");
                        logStreamWriter.Write(this.textBoxPowerEnd.Text);
                        logStreamWriter.Write("\t\t");
                        logStreamWriter.Write("\r\n");
                        logStreamWriter.Flush();
                    }
                    catch (Exception exc)
                    {
                        // Show the error to the user.
                        MessageBox.Show("File could not be created or written to.
Exception: " + exc.Message);
                    }
                    finally
                    {
                        // Close the object if it has been created.
                        if (logStreamWriter != null)
                        {
                            logStreamWriter.Close();
                        }
                    }

                }

                this.textBoxDump.Text += "done.\r\n";

            }


        }


        private string RandomString(int size)
        {
            StringBuilder builder = new StringBuilder();
            Random random = new Random();
            char ch;
            for (int i = 0; i < size; i++)
            {
                ch = Convert.ToChar(Convert.ToInt32(Math.Floor(26 * random.NextDouble() +
65)));
                builder.Append(ch);
            }
            return builder.ToString(0,size);
        }

        public void CreateTxtFile(string FileName, string content)
        {
            StreamWriter myStreamWriter = null;

            try
            {
                // Create a StreamWriter using a static File class.
                myStreamWriter = File.CreateText(FileName);

                // Write the entire contents of the txtFileText text box
                //   to the StreamWriter in one shot.
                myStreamWriter.Write(content);
                myStreamWriter.Flush();
            }
            catch (Exception exc)
            {
                // Show the error to the user.
                MessageBox.Show("File could not be created or written to. Exception: " +
exc.Message);
            }
            finally
            {
                // Close the object if it has been created.
                if (myStreamWriter != null)
                {
                    myStreamWriter.Close();
                }
            }
        }

        private void menuItemGenerateFiles_Click(object sender, EventArgs e)
        {

            this.textBoxDump.Text = this.textBoxDump.Text + "Generating text files.... ";
            CreateTxtFile("Temp\\160B.txt", RandomString(160));
            CreateTxtFile("Temp\\1KB.txt", RandomString(1000));
            CreateTxtFile("Temp\\10KB.txt", RandomString(10000));
            CreateTxtFile("Temp\\100KB.txt", RandomString(100000));
            string LogFileHeader = "Loop\tMode\tKey\t
Input\tIterations\tDuration\tPowerStart\tPowerEnd\tOutput
Length\r\n==========================================================================
=============================== \r\n";
            CreateTxtFile("Temp\\Baselinelog.txt", LogFileHeader);
            CreateTxtFile("Temp\\RSAlog.txt", LogFileHeader);
            CreateTxtFile("Temp\\AESlog.txt", LogFileHeader);

            this.textBoxDump.Text = this.textBoxDump.Text + "done.\r\n ";
        }

        private void menuItem1_Click(object sender, EventArgs e)
        {
            this.menuItemRSA.Checked = true;
            this.menuBaseline.Checked = false;
            this.textBoxCurrentMode.Text = "RSA";
        }

        private void menuItemAES_Click(object sender, EventArgs e)
        {
            this.menuItemRSA.Checked = false;
            this.menuBaseline.Checked = false;
            this.menuItemAES.Checked = true;
            this.textBoxCurrentMode.Text = "AES";

        }

        private void menuBaseline_Click(object sender, EventArgs e)
        {
            this.menuItemRSA.Checked = false;
            this.menuBaseline.Checked = true;
            this.menuItemAES.Checked = false;
            this.textBoxCurrentMode.Text = "Baseline";

        }


    }
}
```

APPENDIX B.   SECURE CHAT

A.   PROGRAM CODE

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Security.Cryptography;
using Microsoft.WindowsMobile.Forms;
using Microsoft.WindowsMobile.PocketOutlook;
using Microsoft.WindowsMobile.PocketOutlook.MessageInterception;
using Microsoft.WindowsMobile.Status;
using System.IO;
using Microsoft.Win32;

namespace SecureChat
{
    public partial class MainForm : Form
    {
        private MessageInterceptor SMSInterceptor = new
MessageInterceptor(InterceptionAction.NotifyAndDelete, true);
        private MessageCondition msgCondition = new MessageCondition();
        string ToPhoneNumber = "";
        string FromPhoneNumber = "";
        string MyPhoneNumber = "+14250010001";
        string FinalMsg = "";


        public MainForm()
        {
            InitializeComponent();
        }

        private void SMSMessageReceived(object sender, MessageInterceptorEventArgs e)
        {
            byte[] RxencryptedData;
            byte[] RxdecryptedData;
            byte[] RxPrivKeyBlob;
            byte[] RxPubKeyBlob;
            byte[] RxSignature;
            byte[] RxPubKeyData;
            ASCIIEncoding ByteConverter = new ASCIIEncoding();

            SmsMessage msg = (SmsMessage)e.Message;

            this.textBoxDump.Text += "Msg Rx [" + msg.Body.Length.ToString()+"], ";
            FromPhoneNumber = msg.From.Address.ToString();

            //Create a new instance of RSA
            RSACryptoServiceProvider RxRSA = new RSACryptoServiceProvider(1024);
            SHA1CryptoServiceProvider RxSHA = new SHA1CryptoServiceProvider();

            if (msg.Body.Substring(0,2)=="**")
            {

            //Decompose Message
            RxencryptedData = Convert.FromBase64String(msg.Body.Substring(2, 172));

            string t = msg.Body.Substring(174, 172);
            RxSignature = Convert.FromBase64String(t);
            //Read private key for decryption
            FileStream RxReadPrivfs = File.OpenRead("Program Files\\SecureChat\\" +
MyPhoneNumber + ".prv");
            BinaryReader RxReadPrivbr = new BinaryReader(RxReadPrivfs);
            RxPrivKeyBlob = RxReadPrivbr.ReadBytes(596);
            RxReadPrivbr.Close();
            RxReadPrivfs.Close();
            RxRSA.ImportCspBlob(RxPrivKeyBlob);

           //Decrypt Message
            RxdecryptedData = RxRSA.Decrypt(RxencryptedData, false);
            this.textBoxDump.Text += "decrypted, ";

            //Read Public Key from Sender public key file
            FileStream RxReadPubfs = File.OpenRead("Program Files\\SecureChat\\" +
FromPhoneNumber+ ".pub");
            BinaryReader RxReadPubbr = new BinaryReader(RxReadPubfs);
            RxPubKeyBlob = RxReadPubbr.ReadBytes(148);
            RxReadPubbr.Close();
            RxReadPubfs.Close();
            RxRSA.ImportCspBlob(RxPubKeyBlob);

            //Verify signature
            string temp = ByteConverter.GetString(RxdecryptedData, 0,
RxdecryptedData.Length);
            byte[] tb = ByteConverter.GetBytes(temp);
            if (RxRSA.VerifyData(tb, RxSHA, RxSignature))
            {
                this.textBoxDump.Text += "verified.\r\n";
                this.textBoxDump.ScrollToCaret();
            }
            else
            { this.textBoxDump.Text += "NOT VERIFIED.\r\n"; }

            //Display Message

            this.textBoxDialog.Text += FromPhoneNumber.Substring(8,4) + ":" +
ByteConverter.GetString(RxdecryptedData, 0, RxdecryptedData.Length) + "\r\n";
            }


            if (msg.Body.Substring(0,2)=="*x")
            {
                //Extract public key
                RxPubKeyData = Convert.FromBase64String(msg.Body.Substring(2, 200));

                textBoxDump.Text += "Public Key from " + FromPhoneNumber + "\r\n";
                FileStream Pubfs = File.Create("Program Files\\SecureChat\\" +
FromPhoneNumber + ".pub");
                BinaryWriter Pubbw = new BinaryWriter(Pubfs);
                Pubbw.Write(RxPubKeyData);
                Pubbw.Close();
                Pubfs.Close();
            }

        }

        private void MainForm_Load(object sender, EventArgs e)
        {
            // Intercept SMS starting with *
            msgCondition.Property = MessageProperty.Body;
            msgCondition.ComparisonType = MessagePropertyComparisonType.StartsWith;
            msgCondition.ComparisonValue = "*";
            SMSInterceptor.MessageCondition = msgCondition;

            //set up event handler to process incoming messages
            SMSInterceptor.MessageReceived += new
MessageInterceptorEventHandler(SMSMessageReceived);
```

```
            this.comboBoxToPhoneNumber.SelectedItem = "+18319175596";
            this.textBoxMsgToSend.Focus();
            this.textBoxToPhoneNumber.Text = "+1";
        }

        private void menuItemExit_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void buttonSend_Click(object sender, EventArgs e)
        {
            byte[] dataToEncrypt;
            byte[] encryptedData;
            byte[] TxPrivKeyBlob;
            byte[] TxPubKeyBlob;

            byte[] Signature;
            ASCIIEncoding ByteConverter = new ASCIIEncoding();

            dataToEncrypt = ByteConverter.GetBytes(this.textBoxMsgToSend.Text);

            //Set Recipient phone number
            if (checkBoxToPhoneNumber.Checked)
            {
                ToPhoneNumber = this.textBoxToPhoneNumber.Text;
            }
            else
            {
            ToPhoneNumber=this.comboBoxToPhoneNumber.SelectedItem.ToString();
            }

            //Create a new instance of RSA
            RSACryptoServiceProvider TxRSA = new RSACryptoServiceProvider(1024);
            SHA1CryptoServiceProvider TxSHA = new SHA1CryptoServiceProvider();

            //Read private key from file
            FileStream TxReadPrivfs = File.OpenRead("Program Files\\SecureChat\\" +
MyPhoneNumber + ".prv");
            BinaryReader TxReadPrivbr = new BinaryReader(TxReadPrivfs);
            TxPrivKeyBlob = TxReadPrivbr.ReadBytes(596);
            TxReadPrivbr.Close();
            TxReadPrivfs.Close();
            TxRSA.ImportCspBlob(TxPrivKeyBlob);

            //Sign Data
            Signature = TxRSA.SignData(dataToEncrypt, TxSHA);
            this.textBoxDump.Text += "Msg signed, ";

            //Read Recipient public key
            FileStream TxReadPubfs = File.OpenRead("Program Files\\SecureChat\\" +
ToPhoneNumber + ".pub");
            BinaryReader TxReadPubbr = new BinaryReader(TxReadPubfs);
            TxPubKeyBlob = TxReadPubbr.ReadBytes(148);
            TxReadPubbr.Close();
            TxReadPubfs.Close();
            TxRSA.ImportCspBlob(TxPubKeyBlob);

            //Encrypt Message
            encryptedData = TxRSA.Encrypt(dataToEncrypt, false);
            this.textBoxDump.Text += "encrypted, ";

            //Compose final outbound message

            FinalMsg = "**" +
Convert.ToBase64String(encryptedData)+Convert.ToBase64String(Signature);

            // Send the SMS message
            SmsMessage MsgToSend = new SmsMessage(ToPhoneNumber, FinalMsg);
            MsgToSend.Send();

            // Update the dialog box
            this.textBoxDialog.Text += "Me:" + this.textBoxMsgToSend.Text + "\r\n";
            // Clear the "Message" edit box
            this.textBoxMsgToSend.Text = "";
            this.textBoxDump.Text += "sent.[" + FinalMsg.Length.ToString() + "]\r\n";

        }


        private void menuItem2_Click(object sender, EventArgs e)
        {

           RSACryptoServiceProvider RSAKey = new RSACryptoServiceProvider(1024);
            byte[] MyPubKeyBlob = RSAKey.ExportCspBlob(false);
            byte[] MyPrivKeyBlob = RSAKey.ExportCspBlob(true);

            this.textBoxDump.Text += "Generating Key Pair... ";

            //Write Keys to files

           FileStream  Pubfs = File.Create("Program 
Files\\SecureChat\\"+MyPhoneNumber+".pub");
            BinaryWriter Pubbw = new BinaryWriter(Pubfs);
             Pubbw.Write(MyPubKeyBlob);
            Pubbw.Close();
            Pubfs.Close();

            FileStream Privfs = File.Create("Program Files\\SecureChat\\"+ 
MyPhoneNumber+".prv");
            BinaryWriter Privbw = new BinaryWriter(Privfs);
            Privbw.Write(MyPrivKeyBlob);
            Privbw.Close();
            Privfs.Close();
            this.textBoxDump.Text += "done. \r\n";

        }

        private void menuItemSendPubKey_Click(object sender, EventArgs e)
        {

            //Read my public key from file
            FileStream ReadMyPubfs = File.OpenRead("Program Files\\SecureChat\\" + 
MyPhoneNumber + ".pub");
            BinaryReader ReadMyPubbr = new BinaryReader(ReadMyPubfs);
            byte[] MyPubKeyBlob = ReadMyPubbr.ReadBytes(148);
            ReadMyPubbr.Close();
            ReadMyPubfs.Close();
            string TxPubKeyString = "*x" + Convert.ToBase64String(MyPubKeyBlob);

            //Send my public key;
            //Set Recipient phone number
            if (checkBoxToPhoneNumber.Checked)
            {
                ToPhoneNumber = this.textBoxToPhoneNumber.Text;
            }
            else
            {
                ToPhoneNumber = this.comboBoxToPhoneNumber.SelectedItem.ToString();
            }

            SmsMessage KeyToSend = new SmsMessage(ToPhoneNumber, TxPubKeyString);
            KeyToSend.Send();

            this.textBoxDump.Text += "Public key sent to " + ToPhoneNumber + "\r\n";

        }
        private void comboBoxToPhoneNumber_SelectedIndexChanged(object sender, EventArgs 
e)
        {

        }

        private void buttonSendClear_Click(object sender, EventArgs e)
        {
            if (checkBoxToPhoneNumber.Checked)
            {
                ToPhoneNumber = this.textBoxToPhoneNumber.Text;
            }
            else
            {
                ToPhoneNumber = this.comboBoxToPhoneNumber.SelectedItem.ToString();
            }

            SmsMessage SendClear = new SmsMessage(ToPhoneNumber, 
this.textBoxMsgToSend.Text);
            SendClear.Send();

            this.textBoxDump.Text += "SMS sent to " + ToPhoneNumber + "in CLEAR!\r\n";

        }

    }

}
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Rafat, Ali, *The SMS Privacy Problem*, Textually.org website, http://www.textually.org/textually/archives/2004/04/003489.htm, last accessed 19 October 2006

[2] Breed, Allen G., *Ubiquitous message technology can be powerful tool for good or ill*, TMCNet Website, http://www.tmcnet.com/usubmit/2006/10/17/1985881.htm, last accessed 19 October 2006.

[3] Jones, Nick, *Don't Use SMS for Confidential Communication*, Gartner Website, 26 November 2002, http://www.gartner.com/DisplayDocument?doc_cd=111720, last accessed 19 October 2006.

[4] GSM World, *GSM Facts and Figures*, http://www.gsmworld.com/news/statistics/index.shtml, last accessed 17 October 2006.

[5] GSM World, GSM Services, http://www.gsmworld.com/services/index.shtml, last accessed 17 October 2006.

[6] European Telecommunications Standards Institute, *GSM 02.01, Digital cellular telecommunications system; Principles of telecommunication services supported by a GSM Public Land Mobile Network (PLMN)*.

[7] Vijay, K. Garg and Joseph E. Wilkes, Principles & Applications of GSM, Prentice Hall 1999.

[8] European Telecommunications Standards Institute, *GSM 02.09, Security aspects*.

[9] A. Mehrotra, *GSM System Engineering*. Norwood, MA: Artech House, 1997.

[10] European Telecommunications Standards Institute, *GSM 02.17, Subscriber Identity Module*

[11] European Telecommunications Standards Institute, *GSM 03.20, Security related network functions*

[12] Margrave, David, *GSM Security and Encryption*, http://www.hackcanada.com/blackcrawl/cell/gsm/gsm-secur/gsm-secur.html, last accessed on 17 October 2006.

[13] Pesonen, Lauri , GSM Interception, Helsinki University of Technology, 21.11.1999, http://www.dia.unisa.it/professori/ads/corso-security/www/CORSO-9900/a5/Netsec/netsec.html, last accessed 17 October 2006.

[14] GSM Cloning, http://www.isaac.cs.berkeley.edu/isaac/gsm.html, last accessed 17 October 2006.

[15] Josyula, R. Rao, Pankaj Rohatgi and Helmut Scherzer. IBM Watson Research Center, *Proceedings of the 2002 IEEE Symposium on Security and Privacy (S&P'02).*

[16] GSM Security, http://www.gsm-security.net/faq/gsm-encryption-algorithm-a5-cipher.shtml, last accessed 25 October 2006.

[17] Elad, Barkan, Eli Biham, Nathan Keller, *Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication*, CS-2003-05, Proceedings of Crypto 2003.

[18] Traynor, Patrick, William Enck, and Patrick McDaniel and Thomas La Porta, *Mitigating Open Functionality in SMS-Capable Cellular Networks*, Proceedings of the ACM Twelfth Annual International Conference on Mobile Computing and Networking (MobiCom), September 2006.

[19] Guthery, Scott and Cronin, Mary, *Mobile Application Development with SMS and the SIM Toolkit*, McGraw Hill, 2002.

[20] Mobile/Cellular Technology, *SMS (Short Message System) Mobile Technology, International,* http://www.mobilecomms-technology.com/projects/sms/, last accessed 23 October 2006.

[21] Starhub Mobile Service: Price Plans, Starhub Website, http://www.starhub.com/mobile/mobileservice/voiceplans.html#voice, last accessed 25 October 2006.

[22] ETSI Technical Specification TS 101 181 V8.9.0 (2005-06) Digital cellular telecommunications system (Phase 2+); Security mechanisms for SIM application toolkit; Stage 2 (3GPP TS 03.48 version 8.9.0 Release 1999).

[23] Ponemon Institute, LLC, *U.S. Survey: Confidential Data at Risk*, http://www.securitymanagement.com/library/Ponemon_laptoptheft1106.pdf, last accessed on 30 October 2006.

[24] GSM Association Press Release 2006, *Mobile Networks to Cover 90% of the World's Population by 2010,* http://www.gsmworld.com/news/press_2006/press06_51.shtml, last accessed 30 October 2006.

[25] Singh, Gurminder, *Mobile Alerts Project Slides*, May 2006, http://www.nps.navy.mil/cs/singh/CMDC/MobileAlerts.pdf, last accessed on 1 July 2006.

[26] Le, Phong and Hsu, Michael, *Web Portal for the Integrated Messaging System*. M.Sc. Dissertation, Naval Postgraduate School, Monterey, CA, 2006.

[27] General Dynamics C4 Systems, *General Dynamics Announces First Type 1 Secure Phone for GSM*, http://www.gdc4s.com/news/detail.cfm?prid=96, last accessed 27 October 2006.

[28] Schneier, Bruce, *Applied Cryptography*, Second Edition, John Wiley & Sons, 1996.

[29] Kaufmann, Charlie, Perlman Radia, Speciner Mike, *Network Security: Private Communications in a Public World*, Prentice Hall, 2002.

[30] Denning, Dorothy, *Information Warfare and Security*, Addison Wesley, 1999.

[31] Saltzer, Jerome H. and Schroeder, Michael D., *The Protection of Information in Computer Systems*. Proceedings of the IEEE, 63(9):1278–1308, 1975.

[32] Hagai, Bar-El, *Introduction to Side Channel Attacks*, http://www.discretix.com/PDF/Introduction%20to%20Side%20Channel%20Attacks.pdf, last accessed 13 November 2006.

[33] Handys-Mobile Website, http://www.handys-mobile.de/img/eten_m600.jpg, last accessed 7 November 2006.

[34] Microsoft Developer Network, *Learn Windows Mobile: Overview*, http://msdn.microsoft.com/windowsmobile/learning/#WMApps, last accessed 7 November 2006.

[35] Microsoft Developer Network, *Windows Mobile Version 5.0 SDK BatteryLevel numeration,* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/mobilesdk5/html/T_Microsoft_WindowsMobile_Status_BatteryLevel.asp, last accessed 8 November 2006.

[36] Bishop, Matt, *Computer Security: Art and Science*, Addison Wesley, 2005.

[37] Alex Varshavsky, Mike Y. Chen, Eyal de Lara, Jon Froehlich, Dirk Haehnel, Jeffrey Hightower, et al. , *Are GSM phones THE solution for localization?*, http://placelab.org/publications/pubs/GSM-hotmobile2006.pdf, last accessed 24 November 2006.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Yeo Tat Soon
   Director, Temasek Defence Systems Institute (TDSI)
   National University of Singapore
   Singapore