SMSSec: an end-to-end protocol for secure SMS

Johnny Li-Chang Lo, Judith Bishop, J.H.P Eloff
Computer Science Department
University of Pretoria, South Africa
jlo,jbishop,eloff@cs.up.ac.za

Abstract

Short Message Service is usually used to transport unclassified information, but with the rise of mobile commerce it has become an integral tool for conducting business. However SMS does not guarantee confidentiality and integrity of the message content. This paper proposes a protocol called SMSSec that can be used to secure a SMS communication sent by Java's Wireless Messaging API. The physical limitations of the intended devices such as mobile phones, made it necessary to develop a protocol which would make minimal use of computing resources. SMSSec has a two-phase protocol with the first handshake using asymmetric cryptography which occurs only once, and a more efficient symmetric *n*th handshake which is used more dominantly. What distinguishes this work from conventional protocols is the ability to perform the secure transmission with limited size messages. Performance analysis showed that the encryption speed on the mobile device is faster than the duration of the transmission. To achieve security in the mobile enterprise environment, this is deemed a very acceptable overhead. Furthermore, a simple mechanism handles fault tolerance without additional overhead is proposed.

Keywords: Small Message Service, Cryptography, Protocols, Mobile Devices, Wireless Messaging API.

1 Introduction

Short Message Service, better known as SMS is a service that enables the sending of text messages over a mobile cellular network. The messages can be stored in that network until they are collected by the recipient's terminal equipment (such as a mobile phone, or devices that can be connected to the network). SMS was originally designed as part of Global System for Mobile communications (GSM), but is now available on a wide range of network standards such as the Code Division Multiple Access (CDMA). By mid-2004, SMS messages were being sent at a rate of about 500 billion per annum [34]. Although SMS was originally

meant to notify users of their voicemail messages, it has now become a popular means of communication by individuals and businesses. Banks worldwide are using SMS to conduct some of their banking services. For example, clients are able to query their bank balances via SMS or conduct mobile payments [10]. People sometimes exchange confidential information such as passwords or sensitive data amongst each other.

SMS provides many conveniences in our everyday lives but is it really secure? When sensitive information is exchanged using SMS, it is crucial to protect the content from eavesdroppers as well as ensuring the origin of the message is from the legitimate sender. SMS messages are sent via a store-and-forward mechanism to a Short Message Service Centre (SMSC), which will attempt to send the message to the recipient and possibly retry if the user is not reachable at a given moment. Transmission of the short messages between SMSC and phone is via the Signalling System Number 7 (SS7) within the GSM MAP (Mobile Application Part) framework [4]. The problem with GSM MAP is that it is an unencrypted protocol allowing employees within the cellular provider's network that has access to SS7 network to eavesdrop or modify SMS messages. Further SMS vulnerabilities are presented in section 2.

Due to the popularity of SMS, Java 2 Micro Edition (J2ME) provides an optional package called Wireless Messaging API (WMA) [29] that enables mobile applications to send and receive wireless messages. J2ME is a Java platform that provides a standard environment for developing applications suitable for running on mobile devices such as mobile phones and PDAs. Unfortunately, WMA does not support any security for SMS communication. SMS does not use TCP as the transport protocol, so it cannot rely on HTTPS to secure the transmission.

There are two reasons why it is beneficial to implement SMS-based applications in Java. The first is the market potential. There are already many Java enabled handheld devices. Secondly, Java provides a richer way to present a SMS message. For example, instead of a plain text presentation, an individual can monitor personal stock exchange shares in the form of a graph constructed from SMS messages. Therefore it is imperative to investigate a way to provide a security mechanism for SMS communication using WMA. At the current time of this writing, there are two versions namely 1.0 and 1.1 for WMA implemented on most Java-enable phones [29].

In this paper, a protocol called SMSSec is proposed that can be used to secure SMS communication between a WMA client and a server using end-to-end encryption. With end-to-end encryption, the encryption process is carried out at the two end systems [25]. Similar work has been done by [4] where the writers make use of an approximated one-time pad scheme to encrypt SMS messages between two mobile phones. The limitation in this mechanism is that it does not ensure end-to-end encryption between the two mobile phones because there is a decryption occurring within the cellular network so that another one-time pad can be created for the receiving phone to decrypt the message.

1.1 Paper layout

This paper contains seven sections. Section 2 presents the security concerns of SMS messaging. Section 3 presents the proposed protocol in terms of the choices made on which cryptographic algorithms to use, design objectives and specification, fault tolerance and implementation considerations. Section 4 and 5 presents the security and efficiency analysis respectively. Section 6 mentions the future challenges facing SMSSec and section 7 concludes.

2 Security concerns

In this section an overview of the security issues within the GSM network infrastructure and the WMA package is presented. These security issues provide the pivotal motivation for this research.

2.1 SMS message path between the mobile client and server

The GSM network uses SMS and Unstructured Supplementary Service Data (USSD) as signalling technology. USSD is a capability built into the GSM standard for support of transmitting information over the signalling channels of the GSM network. USSD provides session-based communication, enabling a variety of applications suitable for mobile banking [10]. The primary benefit of USSD is that it allows for very fast communication between the user and an application. Some of the vulnerabilities within the GSM network infrastructure apply to USSD signals as well. We are not going to focus on USSD in this paper but rather have a pure focus on SMS.

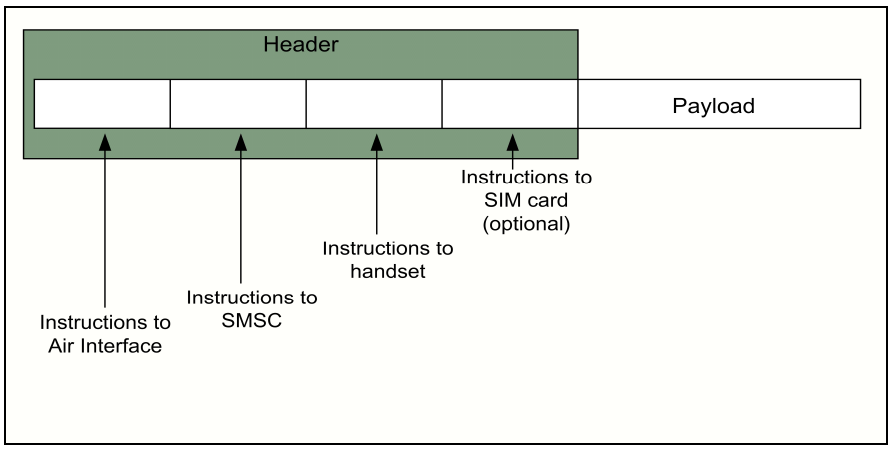

Figure 1: SMS packet structure (adapted from [3])

An SMS packet contains a header and a payload. The header contains information that enables the cellular network to route the SMS message to the correct recipient. The originating address (the mobile phone number of the sender) is also included in the header. The payload is the message content that is displayed on the mobile handset. The size of the payload is 140 bytes, consisting of 160 7-bit characters, or 140 8-bit characters, depending on the provider [4]. Those 140 / 160 characters can comprise of alphanumeric characters or binary bytes.

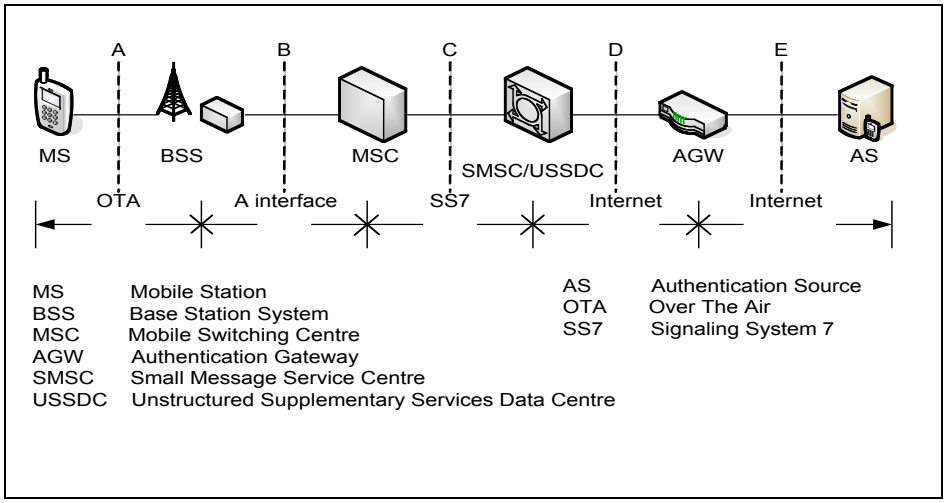

Figure 2: Underlying carrier network (adapted from [18])

The underlying carrier network illustrated in Figure 2 depicts the path of SMS messages being sent in a two-way direction between a mobile station[1] (MS) and the authentication source

[1] The mobile handset is often referred to as the mobile station, where the MS consists of the terminal equipment (TE) and a mobile terminal (MT). The TE is the device that hosts the applications and user interaction, while the MT is the part that proves networking capability. For the rest of this paper, we refer to mobile handsets as mobile station, with the abbreviation **MS**.

(AS) over a GSM network. The AS could be the backend server belonging to an issuing bank or a merchant where it verifies the authenticity of the person who is in possession of the MS. The location of the AS is located in the internet. Assuming that a user wishes to transfer money to some other account, he/she would send a SMS that will initiate this process. Therefore the SMS in this particular scenario would be sent from the MS first. The SMS message reaches the BSS via the air interface. The BSS composes of an base transceiver station (BTS) and a base station (BS) where a number of these base stations that form a coverage area can be connected to a base station controller (BSC). The BSS transfers the SMS message to the MSC over the A interface. The main purpose of the MSC is to control calls to and from other telephony networks. It plays a major role in subscriber roaming by providing all the necessary functionality involved in registering, authentication, location updating, SMS routing and call routing for a roaming subscriber. The MSC routes the SMS to the SMSC that the AS is connected to over the SS7 network. The connection between the AS and SMSC is facilitated by a GSM modem or a TCP/IP connection. It is possible to have more than one SMSC within the service provider to improve scalability. Once the AS has verified the MS, it will send an acknowledgement SMS message back to the MS. Once receiving the message at the SMSC, the contents of the incoming packets are examined and if necessary, converted into the SMS packet structure (see Figure 1). The SMSC queries a Home Location Register (HLR) to determine the location of the target MS. When the HLR locates the MS, it forwards the address of the MSC to the SMSC; otherwise the text is stored in the SMSC, until the target MS is located. The MSC then forwards the acknowledgement message to the BSS that serve the coverage area of the MS.

2.2 Network vulnerabilities identification and analysis

SMS messages are delivered in a best effort manner to other cellular telephone users asynchronously through the cellular network. These networks operate separately from the internet and are sometimes considered to be more secure, accessible and less open to misuse (such as spam). However the GSM network does not provide important security services such as mutual authentication, end-to-end security, non-repudiation or user anonymity [14]. In this section we present the vulnerabilities identified in [18] that occurs within a mobile payment environment.

2.2.1 Vulnerability at the MS

The main risk to a MS is theft or misplacement and as a result, the attacker could gain access to the data content stored on the MS. This threat applies to SMS messages. Even if the entire SMS communication path is encrypted, the decrypted SMS messages are stored within the SIM or persistent flash storage. In order to protect sensitive SMS messages, extra mechanisms should be in place within the application that only decrypts received SMS messages when the user enters a correct password. However, if the user chooses a guessable password, the intention of protecting the encrypted messages are lost.

2.2.2 Loss of messages that do not reach their destination (SMS or USSD)

Because SMS messages are delivered in a best effort manner, there are no guarantees that a message will actually be delivered to its recipient. One can consider that there may be possibilities where messages routed between MS and AS could be lost. The possibilities are that messages could simply be lost, accidentally routed to an incorrect end-user and deliberately misrouted to another end-user (possibly to an attacker). The impact of message loss could result in denial of service (DoS) as transactions cannot be completed. The impact of misrouting could result in loss of privacy. For example, if a client sends a request to transfer his money to another account the possibility is that the information entered within the SMS message could contain the account number and some authentication details. However the feasibility of deliberate misrouting seems low, except if an attacker has access to any of the internal interfaces of the message handling system. Although there are no guarantees of a reliable message delivery, the reliability of SMS messages being delivered is quite high. This is justified by services where it enables sending a SMS message from one user to another in case of critical emergency by pressing a specific speed dial button on the mobile phone.

2.2.3 SMS Spoofing to the MS or AS

The possibility exists that an attacker manages to inject SMS messages into the messaging network with a ‘spoofed’ originator IDs. The attack can be applied in both ways by impersonating the AS for a legitimate MS or impersonating the MS for a legitimate AS. With the former case, the possibility of spoofing is very high as it is possible to send SMS message

from the internet with the correct headers, without the recipient being able to detect that it comes from the internet. Also the mobile service provider is able to change the originator ID. In the latter case, the possibility of spoofing is also high, however the attacker is required to know the authenticating information of the user. This depends on how the SMS service is implemented. If the attacker can manage to spoof an SMS message, fraudulent transactions can be conducted.

2.2.4 Replay of messages

The possibility exists that an attacker arranges for authentication request and/or authentication response messages to be replayed. An attack on the reply of an authentication request message does not seem obvious, however replaying an authentication response could be a more serious vulnerability. If such a replay is possible, it can be used to impersonate a legitimate user and hence authenticate a false transaction. Note however that this attack will not work if there exists an authentication request number (anti-replay mechanisms) that must be included in the response. Such a replay attack may also involve obtaining a copy of this message, which would automatically reveal the authentication information of the client. The likelihood that it would be possible to replay a message without having access to the message seems very low. Moreover, even if such a replay were possible, its effects would be mitigated by secure use of authentication request numbers. If this is implemented, chances of replay attacks are very low.

2.2.5 Capturing and Modifying of data during OTA transmission

SMS messages are sent across the air interface between the MS and BSS (indicated by *A* in Figure 2). In most GSM networks all traffic (digitised voice and data) and signalling data sent across this air interface is encrypted. However, the choice of whether or not to encrypt the data is network specific (it is under the control of the BSS) – in a number of countries, encryption is not permitted. Moreover the choice of encryption algorithm is also network specific, although it must be one of two GSM-specific algorithms known as A5/1 or A5/2. The A5 algorithm is a symmetric cipher, with A5/1 opting for 64-bit key encryption and A5/2 for 16-bit key encryption. The A5/1 and A5/2 implementations of the A5 algorithm have been susceptible to cryptanalysis. Biryukov, Shamir and Wagner [1] demonstrated that the secret key could be cracked in minutes, rendering A5/1 only to counter casual eavesdropper and A5/2 completely insecure. The problem with A5 is that it was never under public scrutiny and

many flaws were exposed by cryptanalysis. If the network is unencrypted (A5/0), or the attacker has the means to decrypt the encrypted traffic, the attacker could read and possibly modify the message content. The result is loss of privacy, extraction of sensitive information and integrity of the message, which could lead to consequences such as fraud and DoS. The feasibility of this threat is very much country and operator specific. Intercepting messages is straightforward and assuming there is no encryption implemented in *A* the attacker would have the problem of finding the traffic of interest amongst all the other traffic and also linking the traffic to the transaction of interest.

2.2.6 Vulnerabilities of the BSS

GSM authentication is a one-way process i.e. the MS is authenticated to the BSS but not vice versa. As a result it is possible for the operator of a 'false' BSS equipment to 'capture' the MS. Of course, this false BSS will typically not possess the keys necessary to set up encrypted communications to the MS, but given that encryption is under BSS control, the false BSS can choose to simply disable encryption. The realization of this vulnerability is a loss of privacy, extraction of sensitive information and DoS. BSS equipment is (apparently) not difficult to obtain and is not expensive, however in order for this attack to be successful, the MS should be within the area of the false BSS. If the transaction is of a type where the legitimate user is likely to be in a particular location, then routine 'capture' of the user's MS using a false BSS is simple to arrange.

2.2.7 Lack of protection for message passing through SS7

The security of the SMS/USSD messages sent across the SS7 networks depends on the levels of physical security provided for these networks and also on the nature of the cryptographic security, if any, provided for these networks. The security provision of the SS7 network is operator dependent. The realization of this vulnerability is a loss of privacy, extraction of sensitive information and DoS. The SS7 network is physically protected but not under the control of a single entity. The protection of the SS7 network against the injection of messages is thus only as effective as the weakest protection offered by any of the operators since messages can be injected into the weakest point of access. As mentioned earlier, GSM MAP is an unencrypted protocol allowing employees within the cellular provider's network that has access to a SS7 network to eveasdrop or modify SMS messages. Also, the SS7 network is not a Federal Information Processing Standard approved for carrying sensitive information [19].

However, given these circumstances, the difficulty the attacker will experience is finding within the high volume of traffic, a particular message that contains sensitive data.

2.2.8 Vulnerabilities of the SMSC and USSDC

If the SMSC/USSDC is accessible in some way to an attacker, then it may be possible to read/manipulate the messages in transit. The potential impact of this vulnerability is a loss of privacy, extraction of sensitive information and a possibility of denial of service (DoS). As with the SS7 networks, the degree to which the SMSC (or USSDC) systems are protected is not known (and probably difficult to find out). The protection almost varies between countries and between operators. The operators have less of an incentive to protect these systems than they do for SS7, since there is no danger of compromise of GSM authentication triplets.

2.2.9 Lack of protection of the interface between SMSC or USSDC and AS

If the interface to the SMSC (or USSDC) is accessible in some way to an attacker, then it may be possible to read/manipulate SMS (respectively USSD) messages in transit. In such a case, several threats could arise such as possible extraction of sensitive information within the message and DoS attacks from a fraudulent merchant. As mentioned earlier, a typical connection between the AS and SMSC is facilitated by a GSM modem or a TCP/IP connection. Because the traffic is outside of the mobile operator, it is not the responsibility of the operator to ensure its protection. The connections between the internet and cellular network introduce open functionality that detrimentally affects the fidelity of a cellular provider's service [8]. To top it all, according to [19] the link between the SMSC and AS is unecrypted.

2.3 WMA package

As mentioned in Section 1, WMA is an optional package for J2ME that enables an application developer to develop an application that sends and receives an SMS. Figure 3, illustrates the components within the WMA package.

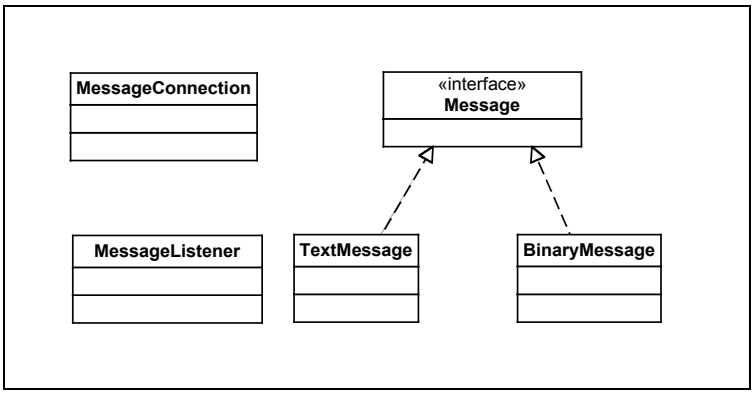

Figure 3: Components of the WMA package (adapted from [20])

None of these components includes any encryption mechanism to protect the confidentiality content of a SMS message during transit. Also, the WMA 1.1 specification [27] stipulates that for security purposes, an application must be granted permission to send and receive messages. Unfortunately, granting permissions for these operations does not ensure confidentiality. Therefore, developers must supply their own security information and accompanying mechanism if they wish to secure the content.

To handle SMS on server computers, one can develop a J2SE/J2EE WMA implementation that interacts with the SMSC via a GSM modem or TCP/IP connection [35]. In fact, the "Generic Connection Framework Optional Package for J2SE" (JSR 197) [26] provides a generic connection framework implementation for J2SE and hence allows the WMA to be ported to Java platforms beyond J2ME. Unfortunately, such an enterprise server-compatible WMA implementation is not available at the time of this writing.

3 SMSSec: the proposed protocol

In this section, various aspects concerning the design of the protocol are discussed. These aspects include the choice of cryptographic algorithms, fault tolerance and implementation.

3.1 Design Requirements

There are two main challenges that were encountered during the design of SMSSec. The first is the limited CPU strength and physical memory size on the MS. To ensure confidentiality and integrity of SMS messages, cryptographic algorithms are required. Cryptography can be computationally expensive to perform especially when public key cryptographic functions are used. We do not prefer to sacrifice security over efficiency so careful judgment needs to be

made on the choice and utilization of cryptographic algorithms on a MS. The choice of in cryptographic algorithms for SMSSec will be discussed in the next section. The second challenge is the SMS message structure and its length. In order to be compliant with all flavours of SMS, the encryption algorithm should possess three attributes, namely [4]:

1. The encrypted message should be in the form of ciphertext in order to meet the SMS message body standards.
2. The encryption algorithm cannot alter the size of the message, since that would cause initially large messages to exceed the maximum allowed size after encryption (no padding).
3. The encryption algorithm should be simple and computationally inexpensive.

Based on the security threats defined in section 2 and the limitations mentioned in the above two paragraphs, a security protocol of this nature should satisfy the following requirements outlined in [16]:

- secure,
- easy to implement,
- low computation needs,
- no storage of secret cryptographic keys,
- achieve entity authentication (integrity),
- achieve key exchange (confidentiality).

In addition to these requirements, we further set objectives to ensure that there is:

- no transporting of secret keys and user's personal identification number (PIN) on any computing environment or intermediate station in GSM network,
- sufficient speed in encryption and decryption,
- use of existing commercially available crypto algorithms,
- availability of end-to-end encryption,
- reliability,
- no additional security protocol (for example Kerberos) or network related hardware infrastructure required for implementation.

The result of applying SMSSec is illustrated in Figure 4, where the SMS message is protected for the entire duration of its transit through the network.

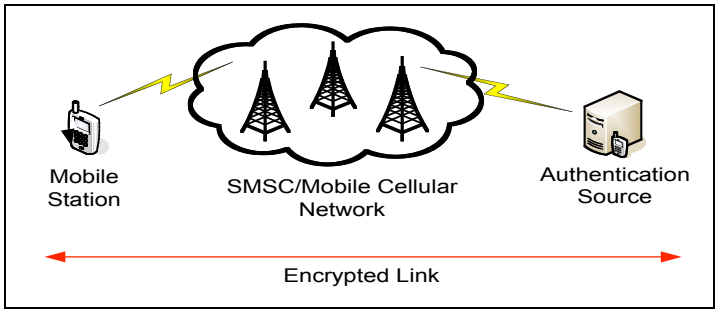

Figure 4: End-to-end encryption

3.2 Cryptographic algorithm choices

According to the objectives laid out in the previous section, it is evident that all cryptographic systems such as symmetric and asymmetric cryptosystems and message authentication code (MAC) are going to be used. The algorithms within the systems chosen should enhance the system's security and efficiency, with security being given at a higher priority than efficiency. It is no good to develop a security protocol and blame the insecurity on the poor choice of the crypto algorithm.

Algorithm	Key size	Description
AES_CTR (Rijndael)	256-bit	Symmetric key cryptography using the AES (Rijndael) cipher and Counter Mode.
RSAES-OAEP	2048-bit	RSA cryptosystem using the OAEP encoding scheme
HMAC_SHA256	256-bit secret value	Message authentication using HMAC and the underlying hash function SHA256

Table 1: Cryptographic algorithms required in SMSSec

In Table 1, the cryptographic algorithms required in SMSSec are listed. For symmetric cipher, the choice is obviously to go for AES for its speed, efficiency and standardization [5, 24]. The counter (CTR) mode is easy to implement and does not introduce padding and collisions (provided that the counter is unique for each key) [9]. In this way, both these combinations

are recommended for securing SMS applications with a small payload size. For a true 128-bit system, a 256 key is recommended due to the birthday paradox. Another reason for supporting a 256-bit key is that due to the cryptic nature of SMS messages, it facilitates easier cryptanalytic attempts based on word frequencies. Therefore by using a 256-bit key, it basically thwarts any cryptanalytic attempts.

For asymmetric cipher the choice is to use the RSAES-OAEP scheme [23]. The RSA cipher has been trusted for many years and is still the most currently used cipher for securing e-commerce applications. The Optimal Asymmetric Encryption Padding (OAEP) encoding function is included to destroy any mathematical structure within small messages that might be used in mobile applications [21]. As technology advances in factoring the RSA primes [22], the safest key size for the next 20 years is 2048-bit [9]. The reader might wonder why not choosing Elliptic Curve Cryptography (ECC) as the core benefit of ECC compared to RSA is that it appears to offer equal security for a far smaller key size, thereby reducing processing overhead [32]. The reason for not choosing ECC is because various aspects of ECC have been patented by a variety of people and companies around the world. Notably, the Canadian company, Certicom Inc. holds over 130 patents related to elliptic curves and public key cryptography in general [19].

HMAC with an underlying hash function called SHA256 (HMAC_SHA256) is chosen due to the recent attacks on collisions on SHA1 [33] and on recommendation by the National Institute of Standards and Technology that SHA1 is to be phased out by the year 2010. Therefore by proactively choosing a hash function that will last beyond the year 2010 is essential. HMAC is required to protect the digest against partial collision and length extension attacks [9].

3.3 Design specification

SMSSec is a protocol that uses public and symmetric key cryptography and two-factor authentication strategy. There are three criteria that authenticate a user namely: something the user knows (such as a password), something the user has (such as a credit card) and something the user is (such as is embodied in a fingerprint). Two-factor authentication is any authentication protocol that requires any two of these forms of authentication to access a system.

The following denotations are used for the explanation of the protocol design and they will be referred to throughout the paper:

- C: Mobile station (Client)
- S: Authentication source (Server)
- U: Cell phone number
- H: HMAC_SHA256(U || PIN || Q), where Q is given by the C
- K: Secret key generating parameters
- K_n: K generated on a new nth session
- Q: New session identifier
- Q_n: Q generated on a new nth session
- R_c: Fresh random challenge from C (64-bits)
- P_f: A certain private port number f
- M: Message
- SQ: Sequence number
- HU: HMAC_SHA256(U)
- HU_n: HU generated on a new nth session
- E_{PKpub}: RSA_OAEP encryption using the public key of the server (2048-bits)
- SK: Secret key (256-bits)
- SK_n: A SK generated from K_n
- E_{SK}: Symmetric key encryption using AES
- E_{SK_n}: Symmetric key encryption using the SK_n key
- PIN: Personal identification number that is known to both the C and S
- || : Append
- {} : Communication through a certain port number

At this point, the reader might question on the use of RSA cipher since one block of the encrypted RSA output exceeds 140 bytes when 2048-bits key is used, which means more than one SMS message are needed if the message content is going to be encrypted using a 2048-bit RSA key. In order to ensure a more efficient protocol, SMSSec is divided into two separate handshakes namely the *First* and the *n*th handshakes so that asymmetric and symmetric encryption techniques can be effectively utilized. These two handshakes are illustrated in Figure 5.

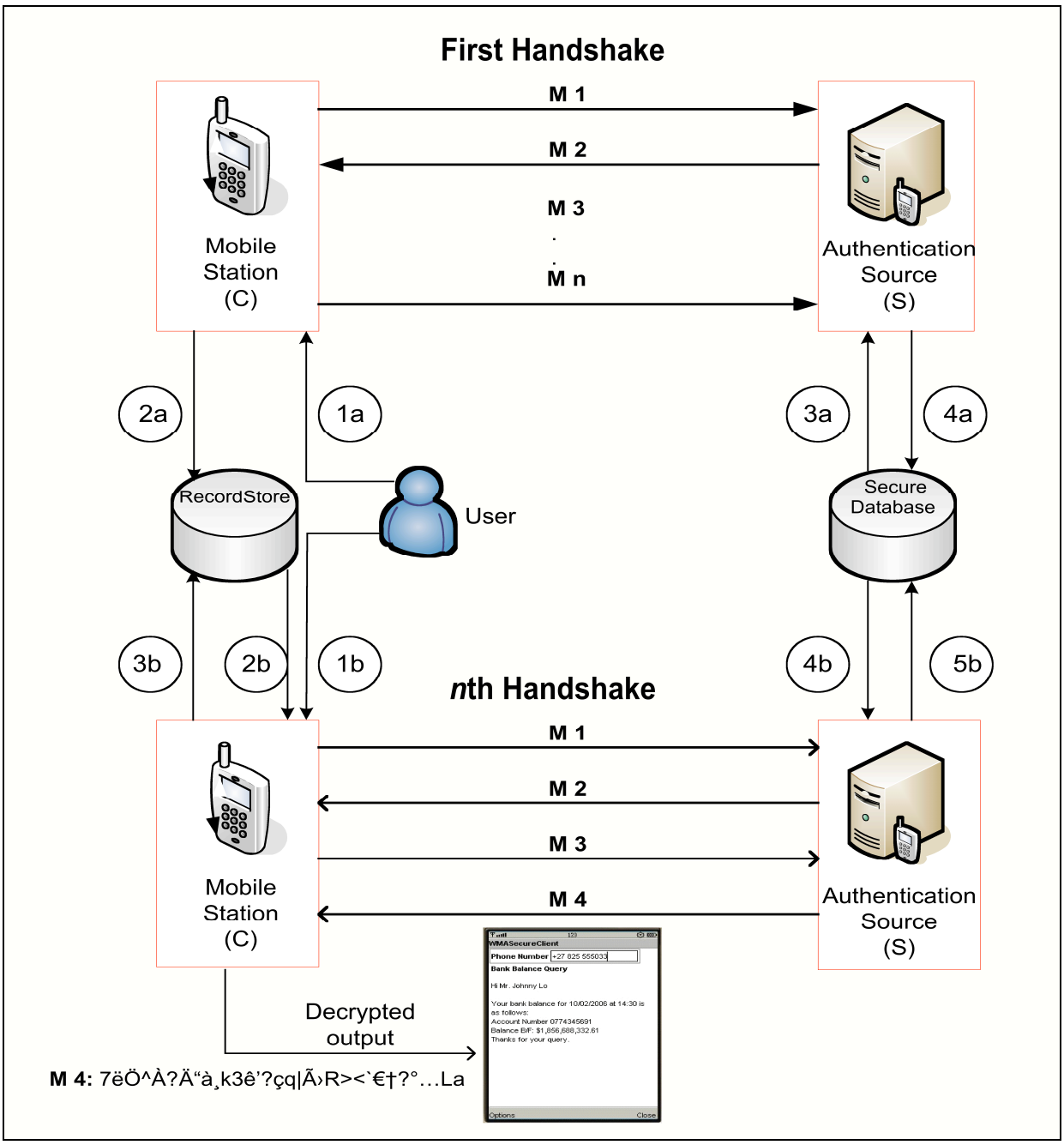

Figure 5: SMSSec protocol

3.3.1 First handshake

The first handshake is defined as follows:

M1: $C\ \{P_z\} \rightarrow S\ \{P_z\}: E_{PKpub}[U \parallel H \parallel K \parallel Q \parallel R_c]$

M2: $S\ \{P_z\} \rightarrow C\ \{P_z\}: E_{SK}[R_c \parallel P_j \parallel SQ]$

M3: $C\ \{P_z\} \rightarrow S\ \{P_j\}: E_{SK}[M \parallel SQ]$

Mn: $S\ \{P_j\} \rightarrow C\ \{P_z\}: E_{SK}[M \parallel SQ]$

The number denotations from Figure 5 are defined as follows:

1a) Inputs *PIN* and *U* before *M1*

2a) Saves (*K*, *Q*) once *M1* completes

3a) Retrieves user's *PIN* to use in *H* for *M1*

4a) Saves (*K*, *Q*, *HU*) once *M1* completes

M1: For the initial handshake message, the user inputs the *PIN* and *U*. The *PIN* is the password or passphrase the user chose for login. Therefore the *PIN* represents what the user knows. Both the *C* and the *S* know the *PIN*. The *U* represents what the user has. The *C* generates *K*, *Q*, *H* and a random challenge R_c. The purpose of the R_c is to assure the freshness of a protocol run [16]. The value *K* consists of values that are required to regenerate a symmetric key using the *PIN*. This is similar to password-based encryption [22], but with additional security provided by *K* in keeping the generated key random for each handshake. The *K* should not contain the *PIN* nor be the actual symmetric key. The value of *K* depends on the key generating algorithms used; therefore it does not have to contain an initialization vector. The simplest and probably the most efficient is that *K* should contain a 256-bit *salt*[2], possibly the number of rounds (*r*) used to stretch the *PIN* [9]. The password stretching technique mentioned in [9] is more efficient than [22] as it contains less computations and it works as follows: Let *p* be the password and *s* be the salt. Using any cryptographically strong hash function *h*, *C* can compute the *SK* as:

$$X_0 := 0$$
$$X_i := h(X_{i-1} \,||\, p \,||\, s) \qquad \text{for } i=1,\ldots,r$$
$$SK := X_r$$

The parameter *r* is the number of iterations in the computation, and should be as large as practical. It goes without saying that X_i and *SK* should be 256-bits long; therefore the recommended hash function is SHA256. The size of *r* is chosen such that computing *SK* takes 200-1000 milliseconds on a user's equipment [9] and is therefore a never a fixed value. This concept applies to the MS as computing speed of the MS will increase in the future; therefore by not having a fixed *r* value, *r* can increase accordingly to the computation speed. There are

[2] The *salt* is a random value of 256-bits generated by a cryptographically strong pseudorandom number generator.

other cryptographic libraries that have key generators [2, 17], which might require different input to generating a symmetric key, therefore the value *K* will be dependent on those input requirements, for example, an inclusion of a counter or an initialization vector.

It is important for the reader to note that *C* generates E_{SK} based on the *K* and that the *S* regenerates E_{SK} itself according to the received *K* from the *C*. E_{SK} is never transported within the handshake. The secret value for the HMAC used in *H* can be the salt value derived from *K*. The value *Q* is the session id for this particular SMS communication originated by the *C*. Because SMS messaging is stateless, *Q* can be used to keep track of the current communication. For the first time in handshake establishment, the value of *Q* in *C* is 1 and *S* is 0 because the value of *Q* in *C* should never be higher than the *Q* in *S* in order to prevent replay attacks during error correction (see section 3.4). The order in which *U*, *H*, *K*, *Q* and R_c are appended is not explicit as long as *S* can unmarshal these elements in the correct order. The size of these elements altogether should be less than 190 bytes instead of 256 bytes due to OAEP encoding. The entire *M1* is encrypted with E_{PKpub}[3] and before *M1* is sent to *S*, *K* and *Q* are saved on a persistent flash storage (record store) on the *C*. The assumption is that *M1* is sent via a fixed port number P_z that is pre-specified within the protocol. Once *S* receives *M1*, it decrypts *M1* using the private key of *S*. If the decryption is successful, *S* will first retrieve the user's *PIN* from the secure database by looking for *U*. *S* verifies the *H* and *Q* to make sure the message is authentic and not replayed. To verify *H*, it first computes *H* and matches the *H* received from *C*. If the *H* computed by *S* matches the one received from *C* then *C* is authenticated. The initial value of *Q* should be 0 to indicate that no handshake ever took place. If the verification of *H* and *Q* fails, then *S* discards *M1* and the protocol terminates. Before *S* sends *M2*, it saves *HU, K* and *Q* in a secure database associating it with the *U*. *HU* is computed by hashing the *U* with the 256-bit salt derived in *K* as the secret value.

M2: Once *C* is authenticated, *S* will generate a new private port number P_j in which *C* can use for further communication after *M2*. The reader should note that P_j is generated uniquely per user; therefore the protocol should manage which ports are used and not used at *S*. This is to ensure more privacy and scalability is given to each connection. *SQ* will start at 1[4] signalling that this is the first message sent using symmetric key encryption. *S* will encrypt *M2* by generating *SK* in the same way *C* did in *M1* using *K*. By using symmetric key cryptography, the size of the message can be sent within one SMS message and requires less computation

[3] The precondition to using the public key of the server is that it is obtained securely.
[4] There is nothing wrong with starting *SQ* at 0. The number 1 is easier to denote that this is the first encrypted message using symmetric key encryption in our opinion.

compared to asymmetric key cryptography. Once *C* receives *M2* it decrypts the message and verifies R_c. *S* is authentic to *C*, if *C* can decrypt and verify R_c, because only the legitimate *S* can generate the *SK*. Otherwise if *C* cannot verify R_c it discards *M2* and the protocol terminates. After *C* verifies *M2* the handshake completes.

M3 and Mn: *M3* and *Mn* are the subsequent SMS messages sent between the *C* and *S*. The *S* will still send SMS messages to the port P_z on *C*, but the *C* will send messages to the new port number P_j[5]. The entire duration of the communication is encrypted using *SK*. Both *C* and *S* must ensure the *SQ* in the current message is incremented since the previous message. This is to ensure that no replay attacks have occurred during the entire communication using the shared *SK* between *C* and *S*. If *SQ* is out of sequence in either *C* or *S*, the protocol should terminate with an appropriate timeout error message. The *C* does not need to send a terminate message to the *S* to close any communication. If there are no more SMS messages sent from *C*, a timeout would occur at *S*. Once the timeout occurs, the P_j assigned to that particular *C* would be freed.

3.3.2 *n*th handshake

The *n*th handshake is defined as follows:

M1: $C\ \{P_z\} \rightarrow S\ \{P_z\}: [HU \parallel E_{SK}[U \parallel H \parallel K_n \parallel Q_n \parallel R_c]]$

M2: $S\ \{P_z\} \rightarrow C\ \{P_z\} : E_{SK\_n}[R_c \parallel P_j \parallel SQ]$

M3: $C\ \{P_z\} \rightarrow S\ \{P_j\}: E_{SK\_n}[M \parallel SQ]$

M4: $S\ \{P_j\} \rightarrow C\ \{P_z\}: E_{SK\_n}[M \parallel SQ]$

The number denotations are defined as follows:

1b) Inputs *PIN* and *U* before *M1*

2b) Retrieves (*K*, *Q*) to produce E_{SK} and *HU* during *M1*

3b) Saves (K_n, Q_n) once *M1* completes

4b) Retrieves *HU* and (*K*, *Q*, *PIN*) to produce E_{SK}, and *H* during *M1*

5b) Saves (K_n,Q_n, HU_n) once *M1* completes

[5] The reader should note that there are no security risks of reusing P_z on *C* because *C* will discard any messages that it cannot verify. *S* already established a secure handshake with *C*; therefore it is still safe for *S* to communicate to *C* via P_z. By reusing P_z on *C*, it will result in a more memory efficient implementation for the *C*.

It is not efficient for SMSSec to always use the RSA cryptosystem to engage in the handshake because it will cost two SMS messages for *M1*. Therefore, to save computation, time and monetary cost on the number of SMS messages send during the handshake, the *n*th handshake is used. The *n*th handshake denotes the subsequent new handshakes conducted when the user wish to establish a new connection with *S*. The first handshake will never be used again, except for fault tolerance (see section 3.4).

M1: The formation of *M1* is similar to the *M1* in the first handshake, except using symmetric encryption. In order for the *S* to distinguish which user the *M1* is sent from, the *C* appends *HU* in front of *M1*. The *C* regenerates the *HU* by reusing the salt value it retrieves from the record store. The K_n and Q_n are the newly generated *K* and *Q* values for the current handshake. Q_n is the result of incrementing *Q* by 1. The entire *M1* is encrypted using the *SK* established during the previous handshake. Before *M1* is sent, *C* replaces the *K* and *Q* with K_n and Q_n. Once *S* receives the *M1* via port P_z, it should determine that the message is smaller than 256 bytes; therefore *HU* is split from *M1* and is used by *S* to look up within the secure database, the *HU* established from a previous session. If the matching *HU* is not found the protocol will terminate, otherwise *K* is retrieved to regenerate *SK* in order to decrypt *M1*. Once *M1* is decrypted, the values *H* and *Q* are evaluated in the same way as it is described in the first handshake, with an extra requirement for *Q* to be a higher value than what *S* has. *C* is authenticated to *S* because it encrypted *M1* using the same *SK* that *S* knew from a previous handshake after verifying *Q* and *H*. Before *S* sends *M1*, the values of *K*, *Q* and *HU* are replaced with K_n, Q_n and HU_n, so that these values will be reused again until the next protocol establishment.

M2: Once *C* is authenticated, *S* will encrypt the same message content as it is described in the first handshake, with the exception that the entire *M2* is encrypted with E_{SK_n} using *SK_n* that is generated in the same way as it is described in the first handshake by using the new K_n. Once C receives *M2*, it uses its own generated *SK_n* to decrypt *M2*. *S* is authentic to *C*, if *C* can decrypt and verify R_c, because only the legitimate *S* can generate the *SK_n.* Otherwise if *C* cannot verify R_c it will discard *M2* and the protocol terminates. After *C* verifies the *M2* the handshake completes.

M3 and Mn: Subsequent messages are commuicated in the same way as it is described in the first handshake, with the exception that these messages are encrypted using *SK_n*.

3.3.3 Security of persistent storage

An important security consideration is ensuring the values K and Q are stored securely on MS. The record store system for MIDP ensures that the record generated by a Java application is only accessible by itself [6, 15]. This means that no other Java applications on the MS may write or delete a record store.

Securing the database at the S is just as important. There are two customary types of database security mechanism that are used to prevent unauthorized access to databases namely discretionary and mandatory [7]. It is not the focus of this paper to provide specific instructions on securing databases, however it is important for the reader to note that SMSSec requires security mechanisms in place for the databases it accesses.

3.4 Fault tolerance

Fault tolerance means a system can still provide a service even in the presence of errors [31]. Here are some possible scenarios that one can encounter errors on a MS during SMS messaging:

- **Crash failure:** Before or after the messages are sent, the MS suddenly crashes. The cause for this crash could be memory fault, persistent storage space is full, or worse a software bug within the implementation of the operating system, and so on.
- **Omission failure:** The MS itself fails in receiving or sending messages as a result of a crash or loss of message within the communication path. The server might have a receive omission failure due to battery failure on the MS after an SMS message was sent.
- **Timing failures:** The MS can experience timing failures when the server times out due to an error in transmission. Sometimes timeouts can be caused by incorrect delivery of message sequences. For example in the case of M1 in the first handshake, the entire message is 256 bytes therefore the message will be split into two 140-byte messages and if the messages are not reassembled in the correct sequence, the server will not be able to decrypt it.

SMSSec should have a mechanism to recover from these errors because during the protocol run, elements such as *K*, *Q*, *HU* are saved and these values might be inconsistent if either the *C* or *S* experiences a failure.

If a failure occurs, the most feasible and efficient solution is to have a forward recovery by asking the user to engage in the first handshake again. The developer can create an option within the application menu to force the protocol to engage in the first handshake in case of an error within the communication. By using the first handshake, *C* recreates *K* and *Q* and sends them to *S* in *M1* exactly as it is described in section 3.3.1. Once *S* receives the message it verifies *H* and *Q* and determines *Q* is a higher value than a 0. Therefore *S* can safely replace the old *K* and *Q* with the current value and the protocol continues. A question might arise is what happens if a failure occurs during the correcting of the error in first handshake? As long as the *Q* generated by the *C* is a higher value than the *Q* stored in *S*, it doesn't matter. For example, during a second attempt at error recovery, *C* might have a *Q* value of 23 and the server 21, but because 23 is greater than 21, the *S* will accept the values as long as *H* is verified. A more difficult question is what happens if the *Q* in *C* is lower than the *Q* in *S*. There is a possibility this might happen because *Q* might not be written onto the flash storage within the MS. Therefore a situation occurs where the *Q* on both the *C* and *S* might equal or that the *Q* on *C* might be less than the *Q* on *S*. Generally speaking, the difference between the *Q* values on *C* and *S* is 1 within an error free communication and should hardly go beyond 10; otherwise there is something wrong with the hardware of the MS, network or the actual server and SMSSec should not be deployed. Therefore during fault handling, the *Q* generated by *C* should be incremented by 10 from the previous *Q* in order to be safe from all these problems.

The last problem is what happens when *Q* reaches 2^{31} - 1[6] at the *C*? The possiblity of this happening for SMS messaging is slim because it is quite impossible for the *C* to establish so many SMSSec connections. For example, assuming a user wishes to establish on average 10000 connections a day (an impossible target anyway), then this user would take ≈ 588 years for *Q* to reach $2^{31} - 1$!

[6] This value is the largest integer value that a Java integer variable can hold.

3.5 Implementation considerations

A suggested architecture placement of the SMSSec protocol within the system is illustrated in Figure 6.

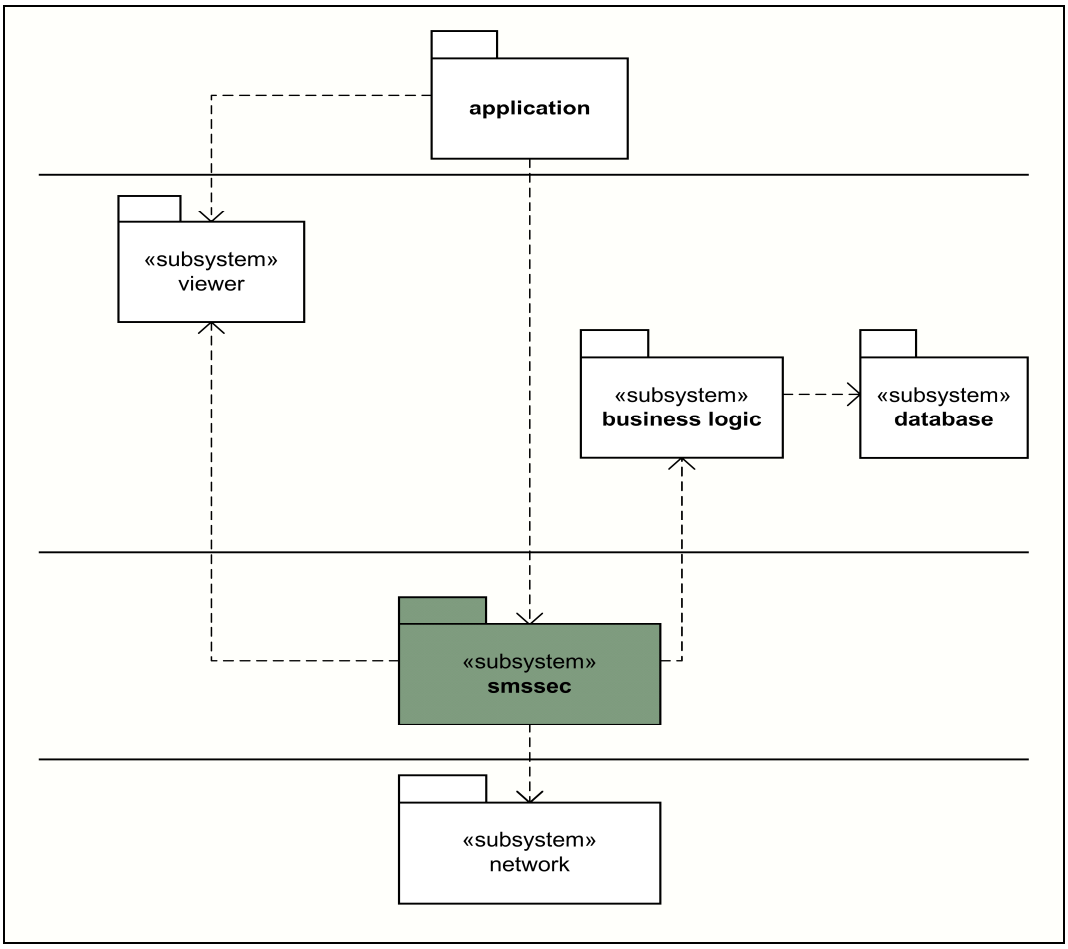

Figure 6: SMSSec package dependency structure

The packages are defined as follows:

- **application:** The application on the *C* and *S* that utilizes SMSSec
- **viewer:** The component that renders the interface according to the received SMS message
- **business logic:** The component that contains the code for writing to and querying from a particular database system
- **database:** The persistent storage that stores the security elements used in SMSSec
- **smssec:** The component that contains the code for the SMSSec protocol
- **network:** The network packages that enables SMS messaging (for example WMA)

It is recommended that SMSSec be implemented as a software component within the system, with the benefit that it can be reused in other applications that require the securing of SMS messages. The `application` utilizes the SMSSec protocol for encrypting the SMS messages to be sent across the cellular network and does not have to access the actual API required to send SMS messages. The actual protocol implementation of `smssec` should not contain the implemented code for accessing the persistent storage. Instead, this should be implemented in the `business logic` component. The implementation of the `business logic` is different between *C* and *S* because the persistent storage medium differs from each other as it is mentioned earlier. During the protocol run `smssec` invokes the interface defined in the `business logic` for accessing the persistent storage. Because SMS messaging is asynchronous, the decrypted message is sent from `smssec` to the `viewer` component by invoking the `viewer`'s interface. The `viewer` enables the display of the message within the application.

By referring to the protocol design in Figure 5 and the package dependency structure in Figure 6, SMSSec is easy to implement with no additional protocol or hardware infrastructure required for implementation.

4 Security analysis

In this section, a more detailed security analysis of SMSSec is presented.

4.1 Application of public key cryptography

The two main concerns of public key cryptography are the authenticity of the public key and private key management. Anyone can generate and forge a public key and falsely claim who they are. In order to prevent this, the public key is digitally signed by a certificate authority (CA). All Java applications are deployed on mobile phones in the form of a compressed Java Archive (jar) file. Ideally this `jar` file must be signed by the originating party and should be downloaded securely onto the mobile phone via HTTPS or Wireless Transport Layer Security (WTLS). It is recommend to follow this practice in order to support SMSSec since the public key will be protected within the `jar` file along with the other class files. This prevents tampering of the authentic public key by an attacker. The public and private key pair is generated at the server and the private key is kept securely in a secure key storage mechanism (for example the `keytool` facility provided by the Java 2 Standard Edition platform). The

`jar` file can also be cryptographically signed prior to deployment. A further security enhancement provided by MIDP 2.0[7] enabled phones is the verifying of the cryptographically signed `jar` file on the MS, which gives users some security about executing downloaded code [13]. In this way, the certificate encapsulated within the `jar` file is also verified so that man-in-the-middle attacks can be prevented.

If the server is compromised and the private key is retrieved by an attacker, the whole protocol will fail because the attacker can act as the legitimate party. Therefore an effective key management on *S* is crucial.

4.2 Authentication between two entities

One of the objectives of SMSSec is to ensure authentication between two parties (*C* and *S*). SMSSec is analysed according to the three rules that Tanenbaum [30] mentions for an authentication protocol:

1. Have the initiator prove who he/she is before the responder has to.
2. Have the initiator and responder use different keys for proof.
3. Have the initiator and responder draw their challenges from different sets.

In point 1, *S* does not have to respond to *C* if it fails to verify the *H* and *Q*. Verification of *H* would fail if *C* provides the wrong *PIN* and *Q* provided by *C* is smaller than the *Q* stored on *S* as the MAC result would not be the same. Before *C* will respond any further after *M2*, the *SK* generated by *S* should be the same as the *SK* generated by *C* and further verification should be done on the R_c to prove to *C* that *S* sent the message for the current protocol run using the session key *SK*. In point 2, the *C* uses the public key of *S* in *M1* and in *M2*, *S* responds by proving it can generates the *SK*. In point 3, the *Q* both is higher for *C* and lower for *S*. Once the handshake establishment completes, the *SQ* keeps an odd and even challenge value between the *C* and *S* for further communication.

[7] MIDP stands for Mobile Information Device Profile and contains APIs that targets cellular phones and simple pagers within the J2ME platform. The current version of MIDP as of this writing is 2.0.

4.3 Attacking the protocol

As already mentioned in section 2, not all vulnerabilities share the same feasibility for an attack. This section presents attacks to SMSSec by assuming all of the vulnerabilities are feasible and it is just as easy to launch attacks as it would be on a public network. In order to attack the SMSSec, an assumed role is made so that either *C* or *S* could be the attacker. For the first attack, *S* is the attacker and *C* is legitimate. *S* captures *M1* but cannot decrypt it because it does not have the private key. What *S* can do is that it can try to fool *C* by sending a previously captured *M2*. When *C* receives this *M2*, it will discard it because the *SK* used is different to the previous handshake and the protocol terminates.

For the second attack, assume *C* is the attacker and further assume *C* has the *U*. *C* can generate *M1* and send it to *S*. However the problem is that *C* does not know the *PIN,* therefore it cannot generate the correct *H*. Even if the *U* is known, it is difficult to compute *HU* because of the 256-bit secret value used for the HMAC. The only way to send a legitimate *M1* is by guessing the correct *PIN* and computing the correct *HU*, which is quite impossible to do if the *PIN* is stretched with a 256-bit salt.

4.4 Security advantages

There are four main security advantages SMSSec provides. The first advantage is that the *PIN* and the *SK* are not included in *M1* because key exchange through network transactions degrades the reliability of a security system [12]. The main idea is to get both *C* and *S* to generate *SK* by using the *PIN* they share between them. An attack on the encrypted *M1* would not be fruitful if the opponent does not know the *PIN*. However this might lead to dictionary or brute-force attacks on the *PIN*. But as already mentioned earlier, a brute-force attack on the *PIN* is fruitless. Due to the fact that *SK* is always generated when used, it is not stored on the MS and in this way *SK* is not compromised if the MS is stolen. The second advantage is achieving mutual authentication (section 4.2). The third advantage is using strong cryptography. Lastly, end-to-end encryption is achieved.

5 Efficiency analysis

Although SMSSec uses strong cryptography, it is still efficient to use on a MS. In this section, an efficiency analysis on the number of SMS messages sent, scalability and handshake duration are presented.

5.1 Number of SMS messages sent

For the first handshake, the size of the elements *U*, *H*, *K*, *Q* and R_c added together should be less than 190-bytes due to OAEP encoding. Fortunately, *U* = 10 bytes, *H* = 32 bytes, *K* = 32 bytes, Q = 4 bytes and R_c = 8 bytes adds up to 86 bytes. Recall earlier that *K* could be variable in length due to certain requirements for generating *K* on different cryptographic APIs. Therefore a 32-byte value for *K* is the minimum, which leaves extra 104 bytes in case *K* is bigger than 32 bytes. The total size of *M1* after encryption is 256 bytes, which requires two SMS messages to be sent. The minimum total space required for persistent storage is 36 bytes on the MS and 68 bytes on the *S* (per connection basis). The total size of *M2* is 16 bytes and because the encryption scheme used is AES_CTR, it requires no padding; therefore after encrypting *M2* the size remains the same and *M2* can be sent within one SMS message.

For *n*th handshake, the size of the *M1* is the same as the first handshake but with an extra 32 bytes for the *HU* to be appended in front. The size of *K* would be limited to 22 bytes for sending *M1* within one SMS message. In general the *n*th handshake is far more efficient than the first and if there are no communication errors, the entire handshake could be done using two SMS messages instead of three.

Communication after both handshakes could be done using one SMS message, depending on the size of *M*. However, by appending SQ the size if *M* is reduced to 136 bytes. After successful first handshake establishment and with the continuation of using the *n*th handshake, SMSSec complies with the three requirements mentioned in the second paragraph of section 3.1.

5.2 Scalability

The issue of using the port numbers could give rise to a question on scalability. The port number is a 16-byte value therefore it can only handle port numbers up to 65536 connections

concurrently (assuming SMS connections don't have reserved port numbers). If SMSSec is applied in a mobile payment environment where the clientele is around 6000000, scalability can be improved if these customers are given different cellular numbers for the *S*, because one number can host 65536 connections concurrently.

5.3 Handshake duration

In this section, the handshake duration is measured according to the time taken (in milliseconds) for the both the first and *n*th handshakes to complete on the *C* and *S*.

Due to the unavailability of a WMA implementation for a desktop server, the *S* will be another MS. The specifications of the *C* and *S* are summarised in Table 2.

Attributes	Client	Server (Using another phone)
Manufacturer	Nokia 6600	Nokia 6680
CPU Architecture	ARM4T	ARM5
CPU Speed	104 MHz	220 MHz
Flash Size	6139 KB	9928 KB
Ram Size	379 KB	1883KB
Memory Card	31066 KB	65536 KB

Table 2: Hardware specification

The goal behind this performance analysis is to give the reader an approximation on the duration of the handshakes especially experienced at the *C*. The duration for *M1* and *M2* denotes the time taken for the devices to compute *M1* and *M2* according to what is described in sections 3.3.1 and 3.3.2. The time taken for the entire protocol to complete successfully is registered at the *C* and not at the *S* because *C* receives the last handshake message. The network used to conduct this experiment is GSM.

The actual SMSSec protocol is implemented using the Linca API [17] and Tables 3 and 4 present the first and *n*th handshake durations respectively.

Device	Duration for M1	Duration for M2	Time taken for the handshake to complete	Tries
C	8906ms	250ms	56797ms	Try 1
S	5657ms	5703ms	-	Try 1
C	9047ms	312ms	49391ms	Try 2
S	5656ms	5516ms	-	Try 2

Table 3: First handshake durations

Device	Duration for M1	Duration for M2	Time taken for the handshake to complete	Tries
C	2250ms	297ms	46313ms	Try 1
S	812ms	1125ms	-	Try 1
C	2016ms	312ms	27438ms	Try 2
S	625ms	1000ms	-	Try 2

Table 4: *n*th handshake durations

As expected, the performance for the *n*th handshake is quicker. The experiment was conducted in two tries for each handshake, with each try taken at a random time during the day. What is interesting to note is the time taken for the SMS messages to be send across the network takes longer than computing *M1* and *M2* because the delays observed for the time taken for the entire handshake to complete includes the durations of *M1* and *M2* for both the *C* and *S*. The time for *S* can be faster if an actual server is used instead of another smart phone.

6 Future challenges

The current work shows the feasibility of executing the protocol between two mobile phone peers. Future work is to implement SMSSec on a WMA enabled J2SE/J2EE server and doing further research into the scalability of the protocol. Concurrently, work is underway to determine the feasibility of implementing SMSSec for other mobile device operating systems such as Symbian OS, Pocket PC, Windows CE, Qtopia and Palm OS. When implementing on

these operating systems, security analysis on accessing the persistent storage will be conducted.

Due to the flexibility of SMSSec, a study will be conducted to determine the applicability of using SMSSec to secure other mobile services such as Multimedia Messaging Service (MMS) and HTTP connections on MIDP 1.0 enabled phones because not all of these phones have HTTPS implemented.

With MMS, mobile clients are able to send graphics, text, audio and video clips instead of plain text such as SMS. MMS relies on packet switched cellular network infrastructure such as the General Packet Radio Service (GPRS) or 3G in forwarding the messages. The security implications of GPRS and 3G [11] apply to MMS messaging as well; therefore it is worthwhile to look into the possibility securing MMS messages.

Currently, MIDP 1.0 phones are still dominating the market [28], but with MIDP 2.0 being released on newer phones, there will still be a mixture of the two MIDP versions in the market for a few years. Although HTTPS is implemented for MIDP 2.0 phones, [6] found that the Reference Implementation of SSL for MIDP 2.0 uses only the system time (`System.currentTimeMillis()`) for its seed update. Hence in order to obtain the random values generated by the client, all what the attacker has to do is to guess the precise system time (in milliseconds) at the moment of the random value computation. Based on this weakness, it would seem ideal to improve HTTPS in MIDP 2.0 using SMSSec where it could be applied on top of HTTPS within the application; however further research is required to determine whether the two protocols can function together, without changing SMSSec if needed.

7 Conclusion

SMS communication is not totally secure and therefore it should not always be trusted. To place trust on SMS messaging especially if it is used in an enterprise environment, a protocol should be in place to ensure confidentiality, integrity and authentication between the two communicating parties. Due to the lack of security in WMA and SMS messaging in general, a protocol called SMSSec is presented to ensure an end-to-end secure SMS communication. Throughout this paper, SMSSec is found to be secure, reliable and efficient. Further work is required to apply SMSSec on other mobile operating systems and services.

8 Acknowledgement

This work was funded the National Research Foundation of South Africa, under the ICT Focus Grant 2053401. We also would like to acknowledge Morkel Theunissen for his assistance with the performance analysis.

9 Glossary

3G	Third Generation Network
AGW	Authentication Gateway
API	Application programming interface
AS	Authentication Source
BSS	Base Station System
CTR	Counter mode
DoS	Denial of Service
ECC	Elliptic Curve Cryptography
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HLR	Home location register
HTTP	Hyper Text Transport Protocol
HTTPS	Secure Hyper Text Transport Protocol
J2ME	Java 2 Micro Edition
MIDP	Mobile Information Device Profile
MS	Mobile Station
MSC	Mobile Switching Centre
USSDC	Unstructured Supplementary Services Data Centre
OAEP	Optimal Asymmetric Encryption Padding
OTA	Over The Air
RSAES	Encryption scheme using the RSA cryptosystem
SMS	Small Messaging Service
SMSC	Small Message Service Centre
SS7	Signaling System 7
WMA	Wireless Messaging API
WTLS	Wireless Transport Layer Security

10 References

[1] A Biryukov, A Shamir, and D Wagner 2000. Real Time Cryptanalysis of A5/1 on a PC. [Online]. Available: http://cryptome.org/a51-bsw.htm. Accessed on: 10 October 2005.

[2] The Legion of BouncyCastle 2005. BouncyCastle API. [Online]. Available: www.bouncycastle.org. Accessed on: 10 November 2005.

[3] Tom Clements 2003. SMS -- Short but Sweet. [Online]. Available: http://developers.sun.com/techtopics/mobility/midp/articles/sms/. Accessed on: 15 November 2005.

[4] N.J Croft and M.S Olivier 2005. Using an approximated One-Time Pad to Secure Short Messaging Service (SMS). In *Proceedings of the Southern African Telecommunication Networks and Applications Conference (SATNAC) 2005*, 71-76.

[5] Joan Daemen and Vincent Rijmen 1999. AES Proposal: Rijndael. [Online]. Available: http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf. Accessed on: 12 July 2004.

[6] Mourad Debbabi, Mohamed Saleh, Chamseddine Talhi, and Sami Zhioua 2005. Security Analysis of Mobile Java. In *Proceedings of the Sixteenth International Workshop on Database and Expert Systems Applications*, 231 - 235.

[7] Ramez A. Elmasri and Shamkant B. Navathe 1994. *Fundamentals of Database Systems*. 2nd ed. Redwood City, California: The Benjamin/Cummings Publishing Company, Inc,

[8] William Enck, Patrick Traynor, Patrick McDaniel, and Thomas La Porta 2005. Exploiting Open Functionality in SMS-Capable Cellular Networks. In *12th ACM Conference on Computer and Communications Security*, Alexandria, VA, USA, To appear.

[9] Niels Ferguson and Bruce Schneier 2003. *Practical Cryptography.* Indianapolis, Indiana: Wiley Publishing, 75-82, 89-91, 233, 350-352.

[10] First National Bank 2005. Cellphone Banking. [Online]. Available: https://www.fnb.co.za/personal/transact/accessyouraccounts/cellphonebanking.html. Accessed on: 11 October 2005.

[11] S. Gindraux 2002. From 2G to 3G: A guide to mobile security. In *Proceedings of the Third International Conference on 3G Mobile Communication Technologies*, 308-311.

[12] Constatinos F. Grecas, Sotirios I. Maniatis, and Iakovos S. Venieris 2003. Introduction of the Asymmetric Cryptography in GSM, GPRS,UMTS, and Its Public Key Infrastructure Integration. *Mobile Networks and Applications*, vol. 8, 145-150.

[13] Jonathan Knudsen 2003. *Wireless Java: Deceloping with J2ME.* New York: Apress, 10.

[14] Otto Kolsi and Teemupekka Virtanen 2004. MIDP 2.0 Security Enhancements. In *Proceedings of the 37th Hawaii International Conference on System Sciences*, Hawaii, 287 - 294.

[15] Micheal Kroll and Stefan Haustein 2002. *Java 2 Micro Edition Application Development.* Indianapolis, Indiana: SAMS, 162-163.

[16] K.Y Lam, S. Chung, M. Gu, and J. Sun 2003. Lightweight security for mobile commerce transactions. In *IEEE Computer Communications*, vol. 26, 2052 -2060.

[17] Johnny Li-Chang Lo and Judith Bishop 2003. Component-based Interchangable Cryptographic Architecture for Securing Wireless Connectivity in Java Applications. In *Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists*, Johannesburg, South Africa, 301-307.

[18] Mobile Payment Forum 2003. Risks and Threats Analysis and Security Best Practices for Mobile 2-Way Messaging Systems. [Online]. Available: http://www.mobilepaymentforum.org/pdfs/MPF_Security_Best_Practices.pdf. Accessed on: 10 November 2005.

[19] National Communications System 2003. SMS over SS7. *Technical Information Bulletin 03-2*, 39,41.

[20] E. Ortiz 2002. The wireless Messaging API. [Online]. Available: http://developers.sun.com/techtopics/mobility/midp/articles/wma/index.html. Accessed on: 16 June 2005.

[21] David Pointcheval 2002. How to Encrypt Properly with RSA. *RSA Laboratories Cryptobytes*, vol. 5, 10-19.

[22] RSA Laboratories 1999. PKCS#5 v2.0: Password-Based Cryptography Standards. *PCKS Standards*.

[23] RSA Laboratories 2002. PKCS#1 v2.1: RSA Cryptography Standard. *PCKS Standards*.

[24] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson 1999. Performance Comparison of the AES Submissions. In *Proceedings of Second AES Candidate Conference*, NIST, 15-34.

[25] William Stallings 2000. *Network Security Essentials, application and standards.* New Jersey: Prentice Hall, 39-42.

[26] Sun Microsystems 2003. JSR 197: Generic Connection Framework Optional Package for the J2SE Platform.

[27] Sun Microsystems 2004. The WMA 2.0 Specification. [Online]. Available: http://java.sun.com/products/wma/. Accessed on: 18 October 2005.

[28] Sun Microsystems 2005. J2ME Devices. [Online]. Available: http://developers.sun.com/techtopics/mobility/device/device. Accessed on: 14 November 2005.

[29] Sun Microsystems 2005. Wireless Messaging API (WMA) JSR 120 and JSR 205.

[30] Andrew S. Tanenbaum 1996. *Computer Networks.* 3rd ed. New Jersey: Prentice Hall, 602-604.

[31] Andrew S. Tanenbaum 2002. *Distributed Systems, Principles and Paradigms.* Upper Saddle River, New Jersey: Prentice Hall, 363-366.

[32] S.A. Vanstone 2003. Next generation security for wireless: elliptic curve cryptography. *Computer and Security*, vol. 22, 12-44.

[33] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu 2005. Finding Collisions in the Full SHA-1. In *Crypto 2005*, Santa Barbara, California, USA, to appear.

[34] Wikipedia 2005. Short message service. [Online]. Available: http://en.wikipedia.org/wiki/Short_Message_Service. Accessed on: 10 November 2005.

[35] Micheal Juntao Yuan 2004. *Enterprise J2ME - Developing Mobile Java Applications.* Upper Saddle River, New Jersey: Prentice Hall, 174-179.