SECURITY ON MOBILE PHONES WITH LIGHTWEIGHT CRYPTOGRAPHIC MESSAGE SYNTAX

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MURAT YASİN KUBİLAY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2007

Approval of the thesis:

SECURITY ON MOBILE PHONES WITH LIGHTWEIGHT CRYPTOGRAPHIC MESSAGE SYNTAX

submitted by **MURAT YASİN KUBİLAY** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen Dean, Graduate School of **Natural and Applied Sciences**	_______________
Prof. Dr. Volkan Atalay Head of Department, **Computer Engineering**	_______________
Dr. Atilla Özgit Supervisor, **Computer Engineering Dept., METU**	_______________

Examining Committee Members

Prof. Dr. Ersan Akyıldız Mathematics Dept., METU	_______________
Dr. Atilla Özgit Computer Engineering Dept., METU	_______________
Asst. Prof. Dr. Albert Levi Faculty of Engineering and Natural Sciences, Sabancı U.	_______________
Dr. Cevat Şener Computer Engineering Dept., METU	_______________
Dr. Onur Tolga Şehitoğlu Computer Engineering Dept., METU	_______________

Date: 24.08.2007

ABSTRACT

SECURITY ON MOBILE PHONES WITH LIGHTWEIGHT CRYPTOGRAPHIC MESSAGE SYNTAX

Kubilay, Murat Yasin
MS, Department of Computer Engineering
Supervisor : Dr. Atilla Özgit
Co-Supervisor: Asst. Prof. Dr. Albert Levi

August 2007, 83 pages

Cryptographic Message Syntax (CMS) is a standard for protecting messages cryptographically. Using CMS, messages can be protected in different content types such as signed-data, enveloped-data, digested-data and authenticated-data. CMS is architected around certificate based key management and symmetric cryptography. In this thesis, a lightweight CMS envelope is proposed for the mobile phones which have limited memory and processing power, in order to provide the privacy of the data either stored on them or exchanged by means of multimedia messaging (MMS) and e-mail. A sample prototype is implemented on mobile phone which makes use of the proposed lightweight CMS. The prototype is compared with standard CMS in terms of application size and performance. The application size decreases approximately by 35% and the envelope preparation duration is much shorter with lightweight CMS in comparison to standard CMS.

Keywords: CMS, PKI, Mobile Phone

ÖZ

BASİTLEŞTİRİLMİŞ KRİPTOGRAFİK MESAJ SÖZDİZİMİ İLE
MOBİL TELEFONLARDA GÜVENLİK

Kubilay, Murat Yasin
Y. Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Dr. Atilla Özgit
Ortak Tez Yöneticisi: Asst. Prof. Dr. Albert Levi

Ağustos 2007, 83 sayfa

Kriptogtafik Mesaj Sözdizimi (KMS), mesajları kriptografik olarak korumak için tasarlanmış bir standarttır. KMS ile mesajlar imzalı veri, zarflanmış veri, özetlenmiş veri ve doğrulanmış veri içerik tipi olarak korunabilmektedir. KMS'nin mimarı yapısı sertifika tabanlı anahtar dağıtımı ve simetrik kriptografi etrafında oluşturulmuştur. Bu tezde, hafıza ve işlem gücü sınırlı olan cep telefonlarında saklanan ve MMS, e-posta vasıtasıyla paylaşılan verilerin güvenliğini sağlamak için basitleştirilmiş kriptografik mesaj sözdizimi önerilmektedir. Basitleştirilmiş KMS'yi kullanan ve cep telefonunda koşan örnek bir prototip geliştirilmiş, uygulama boyutu ve performans kriterleri açısından standart KMS ile karşılaştırılmıştır. Basitleştirilmiş KMS ile standart KMS'ye göre prototip boyutu %35 azalmış olup, zarf hazırlama süresi de oldukça kısalmıştır.

Anahtar Kelimeler : Kriptografik Mesaj Sözdizimi, Açık Anahtar Altyapısı, Mobil Telefon

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

For the last decade mobile phones have become an indispensable part of our life and being carried as an accessory. As the time passes, like every other technological device, the features and capabilities of the mobile phones are also evolved. Nowadays, they are used as music player, camera, camcorder, radio, voice recorder, document reader/writer, game box, internet browser, e-mail client, messenger (SMS/MMS) etc. in addition to their traditional roles of voice communication. With the increasing features of mobile phones, the security of the information stored on them has become an important concern. In case of lost or stolen mobile phones, the stored information can be easily compromised. Actually a greater security deficiency is not the data on a lost or stolen mobile phone. Everyday, millions of information is shared between mobile phone users by means of MMS, e-mail. Photos, videos, office documents, voice records are sent with almost no security. These data are encrypted on the air with a very weak encryption algorithm and can be easily decrypted with a PC processing power by a person who is sniffing them. A better known scenario is, either authorized or unauthorized employees of the GSM [28] operators can easily break the privacy of the communication.

In this thesis, a Public Key Infrastructure (PKI) [31] [32] [33] [34] based solution will be proposed to the problem explained in the previous paragraph. In the solution, an application running on the mobile phone prepares a cryptographic envelope for a file with the help of another application running on the subscriber identity module (SIM) [30] card. The details of two possible sample implementations will be explained and compared in terms of feasibility and performance.

The rest of this thesis is organized as follows. In Chapter 2, public key cryptography will be shortly explained and compared against to symmetric key cryptography. And then, the structure of X.509 Certificates [2] which binds public key's to distinguished names will be discussed. Then, CMS (Cryptographic Message Syntax) [3] will be explained which can be used for preparing encrypted data envelope for different recipients. In Chapter 3, the design and implementation of a sample application, which makes use of the technologies discussed in Chapter 2, will be explained. In Chapter 4, different sample implementations will be compared, and benefits and drawbacks of them will be further argued. And finally in Chapter 5, there will be a summary and conclusion for secure communication with mobile phones.

CHAPTER 2

LITERATURE SURVEY

Security has always been a very sensitive issue for the human being. Two modern cryptographic methodologies, the symmetric cryptography and the asymmetric cryptography will be briefly described and compared in the first section.

X.509 certificates combine the public key of an asymmetric key pair to a Distinguished Name (DN) [7] X.509 certificates will be used for identification of people through out this study. Their structure will be described shortly in the second section.

CMS is a standard used for preparation of a secure data envelope. To have a better understanding for our lightweight CMS proposal, standard CMS will be described in the third section.

Both X.509 Certificates and CMS types are DER encoded in ASN.1 notation. The encoding and decoding of these types is necessary through out this study. For this reason, a short guide for ASN.1 notation and encoding rules can be found in Appendix A. The ASN.1 DER encoding of a sample X.509 Certificate can be found in Appendix B.

2.1 Public Key Cryptography

2.1.1 Symmetric versus Asymmetric Ciphers

The desire of keeping communication confidential from unintended recipients exists since the human being started to communicate [4] Countless methods for hiding data have been developed for thousands of years. These methods attempt to transform the words, letters, and bits to meaningless messages. The intended recipient must be able to transform the meaningless message to its original form in order to read the sender's message

There are two mechanisms to make such transformation, symmetric (secret key) ciphers and asymmetric (public key) ciphers.

2.1.1.1 Symmetric Ciphers

In symmetric cryptography, the same key is used for encryption and decryption. In symmetric cryptography, this key must be kept secret. In this communication both sides uses the same encryption algorithm, the same key. The secret key encryption is depicted in Figure 1.

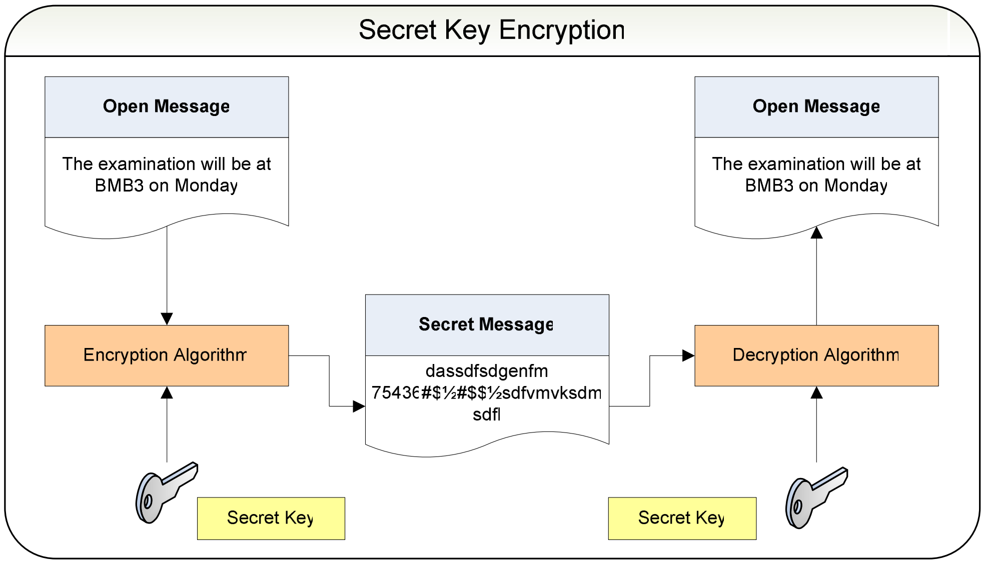

Secret Key Encryption
Open Message
The examination will be at BMB3 on Monday
Encryption Algorithm
Secret Message
dassdfsdgenfm 75436#$½#$$½sdfvmvksdm sdfl
Decryption Algorithm
Open Message
The examination will be at BMB3 on Monday
Secret Key
Secret Key

Figure 1 Secret Key Encryption

Although symmetric ciphers can posses very desirable features such as

- Algorithms are fast
- Implementation of algorithms on hardware is simple
- Gives the service of confidentiality

have the drawbacks

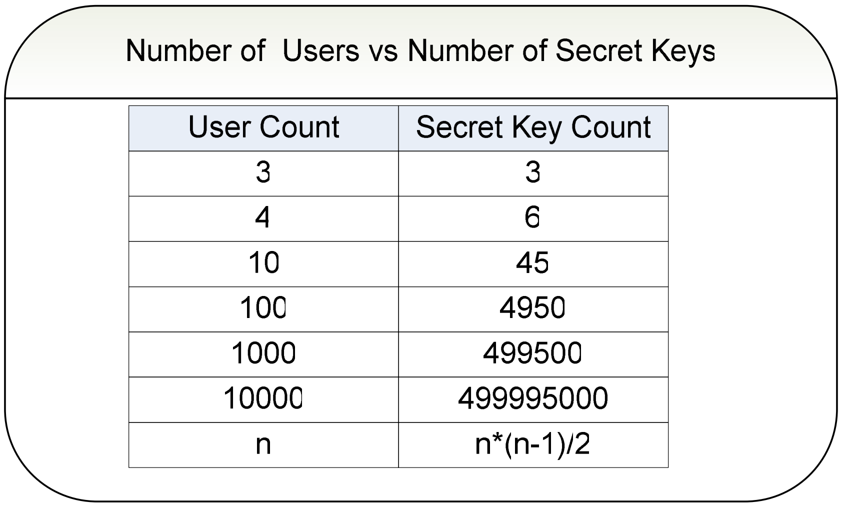
- Secure key exchange is difficult
- It is not scalable
- Communication is difficult between unknown sides
- Integrity and authentication service is difficult

These drawbacks are explained briefly in the following paragraphs.

The sender and the intended recipient must share the same secret key prior to the transmission of the message for the security of the symmetric ciphers. Transmission of this key requires a separate, out of band, secure communication which must occur prior to the intended communication. This additional step can be very difficult in some cases.

The secret key shared between a message sender and recipient *A* must be different from the secret key shared between the message sender and recipient *B*; otherwise, the confidentiality of messages sent to recipient *A* is compromised. If the sender wants to send different messages to 1000 recipients and keeps them secret between the recipients, he has to distribute 1000 different secret keys. The increase in the number of secret key related to the increase of the user is described in Table 1.

Table 1 Number of Users vs Number of Secret Keys



Number of Users vs Number of Secret Keys

User Count	Secret Key Count
3	3
4	6
10	45
100	4950
1000	499500
10000	499995000
n	n*(n-1)/2

If the parties do not know each other, out of band secure key exchange is much more difficult. For instance if an entity wants to share some secret information with another entity, but does not have any prior conversation with it, being certain if the secret key is shared with the correct entity will be quite difficult.

Message Authentication Codes (MAC) [35] can be used to provide the integrity of the data to check if it is altered. The most common way to build a MAC is a shared secret key and cryptographic hash function between the parties. But sharing this information again brings the problem of secure key distribution.

An implicit authentication can be done with symmetric ciphers accepting that the shared secret key is owned only by the authenticated entity. There must be a separate secret key for each of the entity which will be authenticated. Key distribution problem again arises in authentication.

2.1.1.2 Asymmetric Ciphers

In asymmetric cryptography, encryption and decryption is done with two different keys. The keys forming this pair are called public and private keys. In this method of cryptography the private key must be kept secret, but the public key can be shared with relevant people. Because of this feature, asymmetric cryptography is also called public key cryptography.

The sides communicating with this cryptography uses the same encryption algorithm, can reach the relevant keys. The public key encryption is depicted in Figure 2.

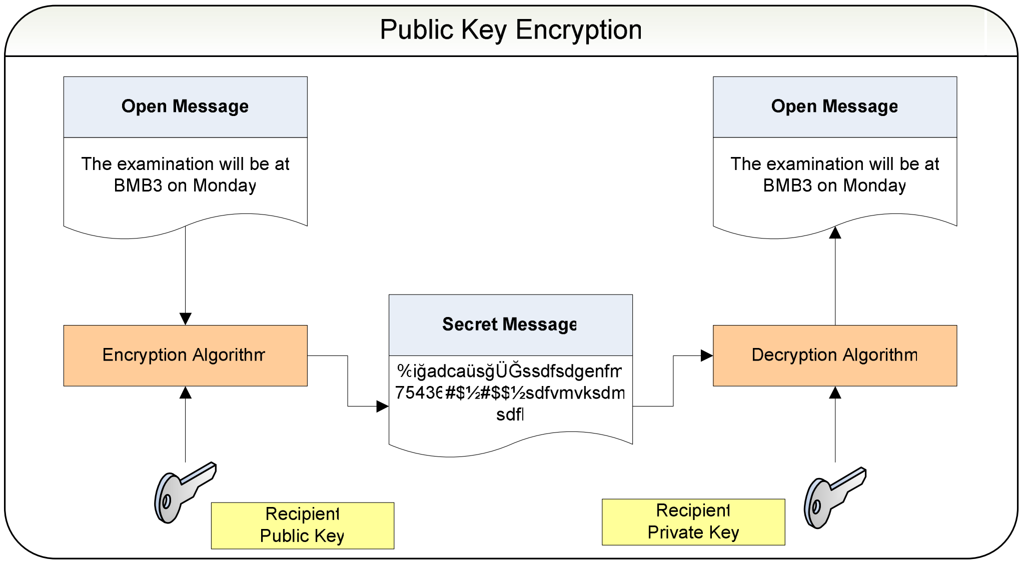


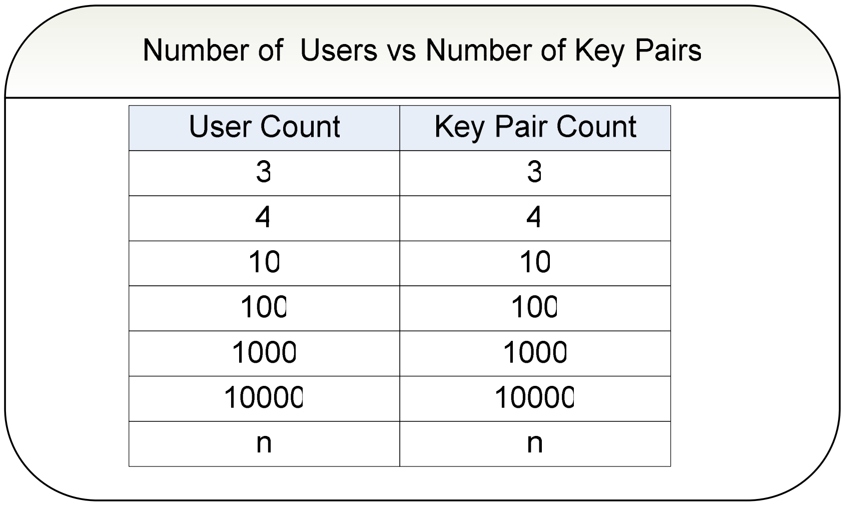
Figure 2 PublicKey Encryption

For asymmetric cryptography, key distribution method is very important. The following points must be taken into consideration for the key management.

- The public keys must be published by an authority, and the modification in these keys must be prevented.
- Key pairs can be produced individually or by a central authority.
- For encryption and signature there must be different key pairs.
- Revocation of keys must be carried out and published under control.

In asymmetric cryptography, key distribution is easier than symmetric cryptography since if an entity who wants to communicate securely needs the public key of the recipient. Since the public key is published by an authority, for a person who wants to communicate securely in a community needs only to produce a key pair. The number of users versus the number of key pairs in asymmetric cryptography is depicted in Table 2.

Table 2 Number of Users vs Number of Key Pairs



Number of Users vs Number of Key Pairs

User Count	Key Pair Count
3	3
4	4
10	10
100	100
1000	1000
10000	10000
n	n

The desirable features of asymmetric cryptography are,

- Key management is scalable
- Resistant against cryptanalysis
- Provides the service of integrity, authentication and non repudiation

with the drawback

- Algorithms are very slow in comparison to symmetric algorithms

2.1.2 Services of Public Key Cryptography

Some of the unknown or difficult services with symmetric ciphers are available with public key cryptography. Some of the important and interesting services are highlighted below.

2.1.2.1 Security between Strangers

Secure secret key exchange among strangers is one of the biggest drawbacks of symmetric cryptography. Since it is very difficult to compute the private key from a given public key, extensive distribution of the public key is possible. The public key can be stored in a public repository. Even though if a message sender has no prior communication with the recipient, he can look up the recipient's public key from the repository and encrypt data for him. But the message sender needs to trust the repository, so that the key he receives really belongs to the recipient. However in the electronic world, unauthorized data modification is a problematic issue and in general public repositories can not be trusted to return correct information. In this point, public key certificates are introduced to provide the trust mechanism.

2.1.2.2 Encryption

Not all but some of the public key algorithms provide the feature of encryption. Data is encrypted with the public key and can only be decrypted with the corresponding private key. However, public key algorithms are very slow and in many situations it is impractical to encrypt bulk data. Instead a two way encryption process is held,

- A randomly generated symmetric key is used to encrypt the data.
- The symmetric key is then encrypted using the public key of the intended recipient of the data

When the recipient receives the encrypted data, he follows a similar process

- The recipient decrypts the symmetric key with his private key
- And than the actual data is obtained by using the symmetric key

2.1.2.3 Digital Signature

Digital signature is a very important service enabled by public key cryptography which can not be easily achievable by symmetric ciphers.

A digital signature operation can be thought as a private key operation on data in which the resulting value is the signature. If only one entity knows its private key, it can be the only entity who could have signed the data. On the other hand anyone who can retrieve the public key of the entity can verify the signature by doing a public key operation on the signature and checking whether the result corresponds to the original data.

Size of the data which will be signed may vary, but a private key operation takes a fixed size input and computes a fixed size output. In order to achieve this, a cryptographic hash function is used. This function can map an arbitrary size of input to a fixed size of output, and it is computationally infeasible to find two different inputs with the same hash. Signing process is held as below,

- Signer takes the hash of the data which will be signed.
- And than a private key operation is done on the hash of the data.

The verification process is similar,

- The verifier takes the hash of the data which will be signed.
- Then the verifier checks this value, the transmitted signature and the signer's public key. If the signature matches the key and the hash value, the signature verifies, else it fails.

The digital signature process can be depicted as in Figure 3.

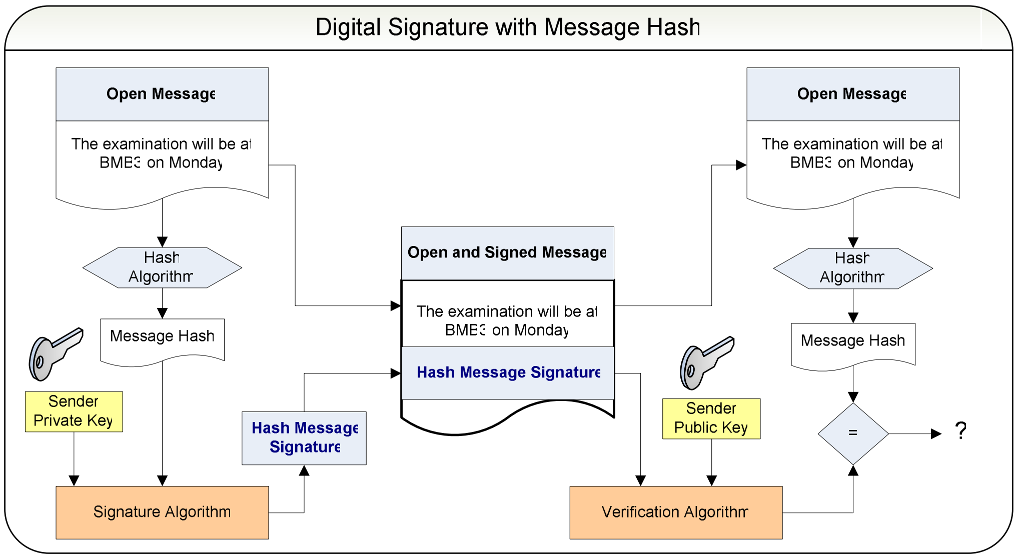


Figure 3 Digital Signature with Message Hash

2.1.2.4 Data Integrity

A digital signature provides both data origin authentication and the data integrity. Since it is very difficult to find two different inputs which have the same hash value, the alteration of original data will lead big changes in the hash and the verification process will fail.

2.1.2.5 Key Establishment

Public key cryptography can also be used for key establishment. A protocol can use public, private key pairs to share some secret symmetric key. It can occur in two ways:

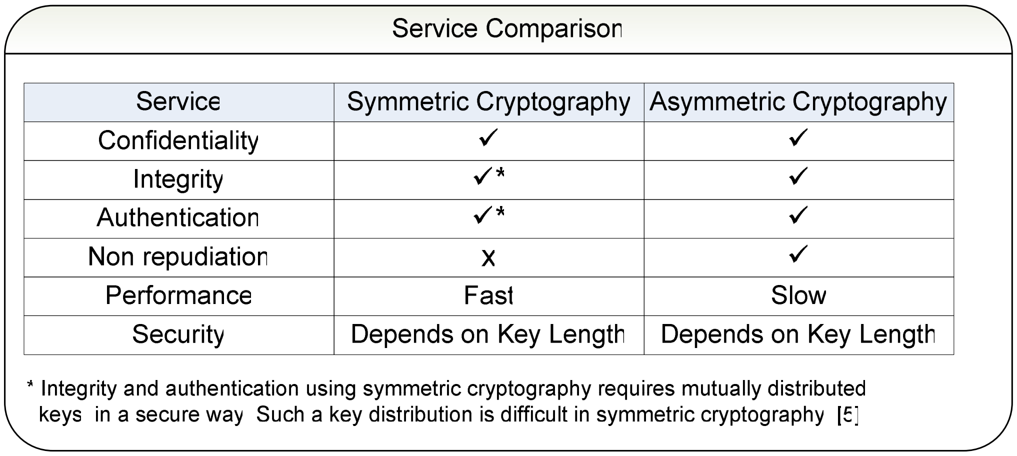
- In key transfer, one entity creates a symmetric key, encrypts it with public key of the recipient and sends.

- In key agreement, both entities can contribute to the generation of a symmetric key.

2.1.3 Comparison of the Symmetric and Asymmetric Cryptography

The comparison of symmetric and asymmetric cryptography is summarized in Table 3.

Table 3 Service Comparison



Service Comparison

Service	Symmetric Cryptography	Asymmetric Cryptography
Confidentiality	✓	✓
Integrity	✓*	✓
Authentication	✓*	✓
Non repudiation	x	✓
Performance	Fast	Slow
Security	Depends on Key Length	Depends on Key Length

* Integrity and authentication using symmetric cryptography requires mutually distributed keys in a secure way Such a key distribution is difficult in symmetric cryptography [5]

2.2 X.509 Public Key Certificates

Users of a public key require confidence that the associated private key is owned by the correct remote subject (person or system) with which an encryption or digital signature mechanism will be used. This confidence is obtained through the use of public key certificates, which are data structures that bind public key values to subjects [1] . The goal is to provide single mechanism by which a relying party is assured that

- The integrity of the public key is provided

- The public key has been bound to the claimed owner in a trusted way

This section explains how X509 Certificates [2] accomplishes these goals.

X.509 Certificates are published by ITU-T [36] In the X.509 The Directory: Public-Key and Attribute Certificate Framework [2] standard, X.509 certificates with versions v1, v2, v3 and X.509 Certificate Revocation List with versions v1 and v2 is defined.

X.509 certificates may be considered as three nested components. The first component is the protective envelope. The digital signature provides the protection. Inside the envelope, there is a basic certificate content. The basic certificate content includes the information that must be present in every certificate. The basic certificate content may include an optional set of certificate extensions.

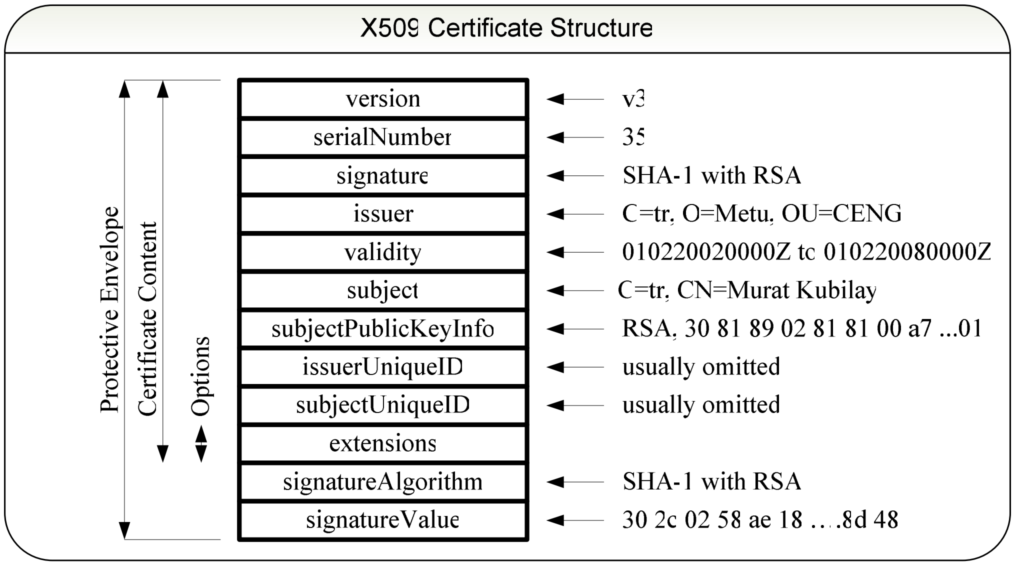


Figure 4 X.509 Certificate Structure

2.2.1 Protective Envelope

There are 3 fields in the outermost level of the certificate. These fields are the to-be-signed certificate, the signature algorithm identifier and the signature itself.

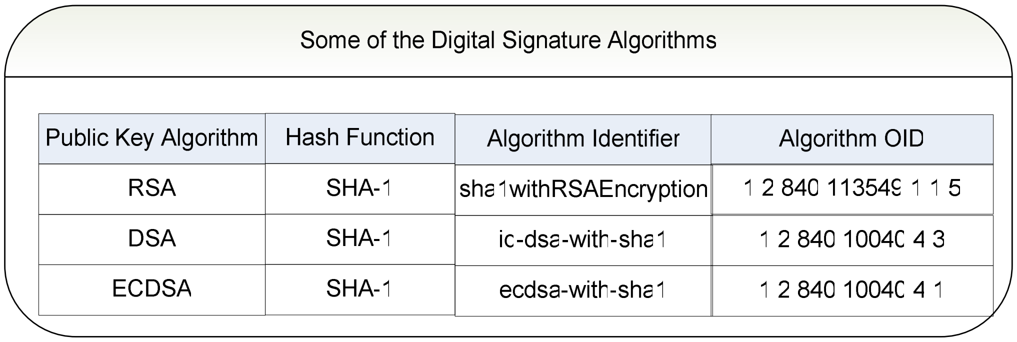
The ASN.1 type definition of the certificate is defined as below:

```
Certificate ::= SEQUENCE {
    tbsCertificate       TBSCertificate,
    signatureAlgorithm   AlgorithmIdentifier,
    signature            BIT STRING }
```

tbsCertificate contains the signed certificate. It will be explained in detail.

signatureAlgorithm contains an algorithm identifier and it identifies the digital signature algorithm identifier used by the certificate issuer to sign the certificate. Some of the digital signature algorithms are listed in Table 4.

Table 4 Some of the Digital Signature Algorithms



Some of the Digital Signature Algorithms

Public Key Algorithm	Hash Function	Algorithm Identifier	Algorithm OID
RSA	SHA-1	sha1withRSAEncryption	1 2 840 113549 1 1 5
DSA	SHA-1	ic-dsa-with-sha1	1 2 840 10040 4 3
ECDSA	SHA-1	ecdsa-with-sha1	1 2 840 10040 4 1

signatureValue contains the digital signature. The digital signature is computed using the ASN.1 DER encoded value of the tbsCertificate field.

2.2.2 Certificate Content

Certificate content contains all the basic certificate information. At minimum, it contains six fields: the serial number, the certificate signature algorithm identifier, the certificate issuer name, the certificate validity period, the public key and the subject name. The version number, two unique identifier and the extensions are optional.

The ASN.1 type definition of the basic certificate is defined as below:

```
TBSCertificate  ::=  SEQUENCE  {
    version         [0]  Version DEFAULT v1,
    serialNumber         CertificateSerialNumber,
    signature            AlgorithmIdentifier,
    issuer               Name,
    validity             Validity,
    subject              Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                    -- If present, version MUST be v2 or v3
    subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                    -- If present, version MUST be v2 or v3
    extensions      [3]  Extensions OPTIONAL
                    -- If present, version MUST be v3 --  }

Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }

CertificateSerialNumber  ::=  INTEGER

Validity ::= SEQUENCE {
    notBefore      Time,
    notAfter       Time  }
```

```
Time ::= CHOICE {
    utcTime        UTCTime,
    generalTime    GeneralizedTime }

UniqueIdentifier  ::=  BIT STRING
SubjectPublicKeyInfo  ::=  SEQUENCE  {
    algorithm            AlgorithmIdentifier,
    subjectPublicKey     BIT STRING  }

Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension

Extension  ::=  SEQUENCE  {
    extnID      OBJECT IDENTIFIER,
    critical    BOOLEAN DEFAULT FALSE,
    extnValue   OCTET STRING }
```

version is an optional field which describes the syntax of certificate. The default value of it is v1, but most of the modern certificates have the value v3.

serialNumber is an integer assigned by the certificate issuer to each certificate. The serial number must be unique for each certificate generated by an issuer. The combination of serialNumber and the issuerName uniquely identify any certificate.

signature is an algorithm identifier. Signature is the same as the signatureAlgorithm.

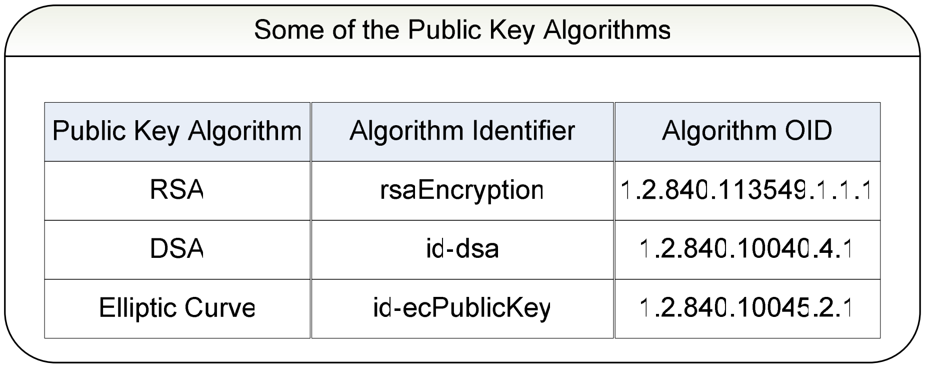
issuer field contains the X.520 [10] distinguished name of the certificate issuer.

validity contains two dates on which the certificate becomes valid, and the date on which the certificate expires.

subject contains the X520 [10] distinguished name of the holder of the private key corresponding to the public key in this certificate.

subjectPublicKeyInfo contains the subject's public key and the algorithm identifier. The public key in this field is used to verify a signature, to encrypt information for the subject, and if the certificate is a CA certificate than, to verify the signature on a certificate. Some of the public key algorithms are listed in Table 5.

Table 5 Some of the Public Key Algorithms



Some of the Public Key Algorithms

Public Key Algorithm	Algorithm Identifier	Algorithm OID
RSA	rsaEncryption	1.2.840.113549.1.1.1
DSA	id-dsa	1.2.840.10040.4.1
Elliptic Curve	id-ecPublicKey	1.2.840.10045.2.1

2.3 Cryptographic Message Syntax

The Cryptographic Message Syntax (CMS) [3] is a standard published by IETF [29] , which describes a general syntax for the data that may have cryptography applied to it, such as digital signatures, digital envelopes, digested-data, encrypted-data and authenticated-data. Recursion can be used in the syntax, so that, for example one can sign a previously enveloped data or one envelope can be nested inside another. It allows different attributes such as the signing time, association of a countersignature with a signature. In addition, the signing or encrypting certificates and the related revocation lists can also be inserted in the syntax. Any form of digital data can be digitally signed, digested, enveloped, authenticated or encrypted by using CMS. CMS is architected around certificate-based key management.

CMS is also compatible with Privacy-Enhanced Mail (PEM) [21] If the signed-data and signed-and-enveloped-data content are constructed in a PEM-compatible mode, then it can be converted into PEM messages without any cryptographic operations. PEM messages can similarly be converted into the signed-data and signed-and-enveloped data content types.

CMS is also employed in many other widely used cryptographic standards, such as S/MIME [17] [18] , PKCS#12 [19] and digital time stamping protocol RFC 3161 [11] .

The enveloped-data content type of CMS will be the point of interest for the rest of this section. A content type identifier is associated with content in CMS. The ASN.1 syntax of CMS is defined as follows.

```
ContentInfo ::= SEQUENCE {
     contentType ContentType,
     content [0] EXPLICIT ANY DEFINED BY contentType }

ContentType ::= OBJECT IDENTIFIER
```

The fields of ContentInfo have the following meanings:

contentType denotes the type of the related content. The content type is associated with an object identifier. An authority defines the object identifiers which is a unique string of integers and assigns the object identifier to a contentType.

content is the data identified by the contentType. For CMS, the content types can be signed-data, enveloped-data, digested-data, encrypted-data and authenticated-data. Here only the enveloped-data content type will be focused on.

In enveloped-data content type, any type of encrypted content and encrypted content-encryption keys for one or more recipients can be stored. A digital envelope for a recipient is comprised of the encrypted content and one encrypted content-encryption key. There are several key management techniques and a data can be enveloped for any number of recipients with one of these techniques.

The construction of enveloped-data is defined by the following steps:

1. Randomly a content encryption key for a particular content-encryption algorithm is generated.

2. For each recipient the content encryption key is encrypted. The encryption may differ according to the key management technique employed. In *key transport* technique the content encryption key is encrypted with the recipient's public key, in *key agreement* technique the recipients public key and the sender's private key is used to generate a symmetric key which is used for encrypting content encryption key, in *symmetric key-encryption keys* technique, content encryption key is encrypted with a previously distributed symmetric key and finally in *passwords* technique, a password is used to generate a key to encrypt content encryption key.

3. A RecipientInfo value is created which contains the encrypted content-encryption key and other recipient-specific information for each recipient. The structure of RecipientInfo is described in more detail in Section 2.3.2.

4. The content is encrypted with the content-encryption key.

5. The encrypted content and all of the recipientInfo values are collected into an EnvelopedData. The structure of EnvelopedData is described in Section 2.3.1.

When the recipient receives the envelope, first of all, he decrypts the content encryption key which is encrypted for him, and then decrypts the content with the decrypted content encryption key.

The top-level data structure EnvelopedData, per-recipient information type RecipientInfo, content-encryption and key-encryption process is described in the following sections.

2.3.1 EnvelopedData Type

EnvelopedData type is identified with the following object identifier.

id-envelopedData OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 3 }

The enveloped-data content is described in a structured ASN.1 type EnvelopedData:

```
EnvelopedData ::= SEQUENCE {
        version CMSVersion,
        originatorInfo [0] IMPLICIT OriginatorInfo OPTIONAL,
        recipientInfos RecipientInfos,
        encryptedContentInfo EncryptedContentInfo
        unprotectedAttrs [1] IMPLICIT UnprotectedAttributes OPTIONAL }

OriginatorInfo ::= SEQUENCE {
        certs [0] IMPLICIT CertificateSet OPTIONAL,
        crls [1] IMPLICIT RevocationInfoChoices OPTIONAL }
```

RecipientInfos ::= SET SIZE (1..MAX) OF RecipientInfo

EncryptedContentInfo ::= SEQUENCE {

contentType ContentType,

contentEncryptionAlgorithm ContentEncryptionAlgorithmIdentifier,

encryptedContent [0] IMPLICIT EncryptedContent OPTIONAL }

EncryptedContent ::= OCTET STRING

UnprotectedAttributes ::= SET SIZE (1..MAX) OF Attribute

The fields of type EnvelopedData is described in more detail below.

Version is the syntax version number. *OriginatorInfo* provides information about the sender. *RecipientInfos* is a collection of per-recipient information. There must be at least one element in the collection. *EncryptedContentInfo* is the encrypted content information. *UnprotectedAttributes* are a collection of attributes appended to the encrypted content. They are not encrypted and may be used for adding the content hash, signing time etc. information to the envelope

The fields of type *EncryptedContentInfo* have the following meanings:

ContentType indicates the type of content. *ContentEncryptionAlgorithm* identifies the content-encryption algorithm, and any associated parameters, used to encrypt the content. The same content-encryption algorithm and content-encryption key are used for all recipients. *EncryptedContent* is the result of encrypting the content.

2.3.2 RecipientInfo Type

Per-recipient information is represented in the type RecipientInfo. The keys used to encrypt content encryption keys may be encrypted with several key management techniques. All these techniques are supported in recipientInfo type. But, in this study only the key transport technique will be focused on.

The ASN.1 structure of the RecipientInfo and the EncryptedKey is defined as follows:

RecipientInfo ::= CHOICE {

ktri KeyTransRecipientInfo,

kari KeyAgreeRecipientInfo,

kekri KEKRecipientInfo,

pwri PasswordRecipientinfo,

ori OtherRecipientInfo}

EncryptedKey ::= OCTET STRING

KeyTransRecipientInfo structure stores the per recipient information. Each instance of KeyTransRecipientInfo transfers the content-encryption key to one recipient.

The ASN.1 structure of the KeyTransRecipientInfo, RecipientIdentifier and IssuerAndSerialNumber is defined as follows:

```
KeyTransRecipientInfo ::= SEQUENCE {
        version CMSVersion,  set to 3
        rid RecipientIdentifier,
        keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
        encryptedKey EncryptedKey }

RecipientIdentifier ::= CHOICE {
        issuerAndSerialNumber IssuerAndSerialNumber
        subjectKeyIdentifier [0] SubjectKeyIdentifier}

IssuerAndSerialNumber ::= SEQUENCE {
        issuer Name,
        serialNumber CertificateSerialNumber }
```

The fields of type KeyTransRecipientInfo is described as follows:

version is the syntax version number.

RecipientIdentifier specifies the recipient's certificate or key that was used by the sender to protect the content-encryption key. The content-encryption key is encrypted with the recipient's public key. RecipientIdentifier can be identified either by the certificate issuer distinguished name and the certificate serial number or by subjectKeyIdentifier which uniquely identifies the certificate by a key identifier. This is actually the subjectKeyIdentifier extension of the certificate *keyEncryptionAlgorithm* identifies the key-encryption algorithm used to encrypt the content-encryption key for the recipient.

encryptedKey is the result of encrypting the content-encryption key for the recipient.

CHAPTER 3

A PROPOSAL FOR MOBILE PHONES SECURITY WITH KORUGAN

As explained in the first Chapter, in today's world mobiles phones are used for many purposes such as listening music, taking pictures, recording videos, watching TV, reading, generating documents, sending e-mail etc. in addition to voice communication. In the mobile phones, a wide range of digital files are stored and shared by means of MMS and e-mail. The privacy and security of the stored and shared files in the mobile phones is a major concern for many people since these files can easily be compromised by either unauthorized or authorized person due to the insufficient security precautions. In this chapter, a PKI based solution will be proposed in order to fix these concerns.

Today applications can be developed and installed on the mobile phones and as well as on the SIM cards. The proposed solution, named *Korugan*, is mainly composed of two applications. One of the applications will run on the SIM card which will handle the cryptographic operations and the other application will run on the mobile phone which will prepare and resolve a lightweight CMS envelope for the encrypted data.

The steps for preparation of lightweight CMS envelope in Korugan is described below and depicted in Figure 5.

- A random symmetric encryption key is generated.
- The plain file is encrypted with the symmetric encryption key.
- The symmetric encryption key is encrypted for each recipient with their asymmetric public keys which are extracted from their X.509 certificates.
- The encrypted file and the encrypted symmetric keys are packed into the CMS envelope.

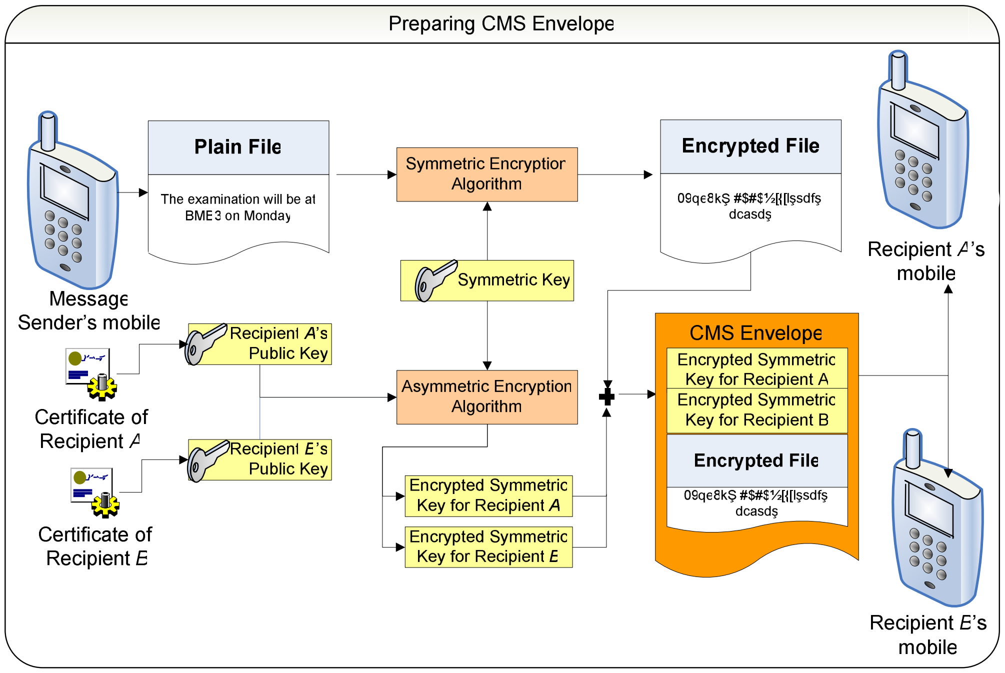
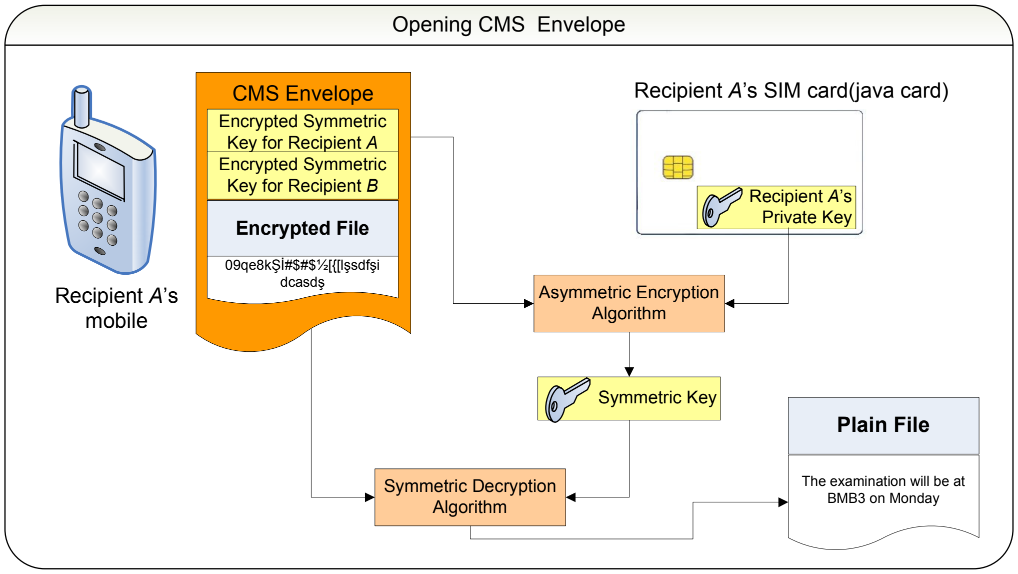

Preparing CMS Envelope
Plain File
The examination will be at BME3 on Monday
Symmetric Encryption Algorithm
Encrypted File
09qe8k$ #$#$½[{[lşsdfş dcasdş
Recipient A's mobile
Message Sender's mobile
Symmetric Key
Recipient A's Public Key
Certificate of Recipient A
Asymmetric Encryption Algorithm
CMS Envelope
Encrypted Symmetric Key for Recipient A
Encrypted Symmetric Key for Recipient B
Encrypted File
09qe8k$ #$#$½[{[lşsdfş dcasdş
Recipient B's Public Key
Certificate of Recipient B
Encrypted Symmetric Key for Recipient A
Encrypted Symmetric Key for Recipient B
Recipient B's mobile

Figure 5 Preparing CMS Envelope

Opening the steps of the lightweight CMS envelope in Korugan is described below and depicted in Figure 6.

- The lightweight CMS envelope is parsed, the recipient list and the encrypted content is identified.
- The recipient list is searched if the envelope is prepared for the message recipient.
- If the envelope is prepared for the message recipient, the encrypted symmetric encryption key is decrypted with the private key of the recipient which is stored in the SIM card.
- The encrypted message is decrypted with the symmetric key.



Opening CMS Envelope
Recipient A's mobile
CMS Envelope
Encrypted Symmetric Key for Recipient A
Encrypted Symmetric Key for Recipient B
Encrypted File
09qe8kŞl#$#$½[{[lşsdfşi dcasdş
Recipient A's SIM card(java card)
Recipient A's Private Key
Asymmetric Encryption Algorithm
Symmetric Key
Symmetric Decryption Algorithm
Plain File
The examination will be at BMB3 on Monday

Figure 6 Opening CMS Envelope

Since lightweight CMS envelopes are prepared using the X.509 certificates of the mobile phone users, the secure distribution of the X.509 certificates is very important. How can one know if a certificate belongs to an intended recipient? There may be two solutions two this problem. Either a certificate can be shared by some means like MMS, e-mail etc. between users, or can be downloaded from a trusted repository.

In the following sections, firstly the need for a lightweight CMS and a proposal for lightweight CMS will be explained, and then shortly the technologies employed for the design and the implementation of Korugan will be discussed and the design of the applications will be explained in detail.

3.1 Lightweight CMS

CMS is very ideal for enveloping encrypted data for any number of recipients. Since it is used not only for enveloped data content type but also, signed data, encrypted data, authenticated data content types, its ASN.1 structure is quite complex. But in mobile phones we are only interested in the enveloped data content type and do not need the rest. The required memory and processing power for DER encoding and decoding of the standard CMS structure is excessive for the mobile phones which have limited memory and processing power. For this reason a lightweight CMS structure is proposed in the following sections.

3.1.1 ContentInfo Type

For lightweight CMS, only the enveloped data content type will be supported.

3.1.2 EnvelopedData Type

In the EnvelopedData type the optional fields originatorInfo and unprotectedAttributes will be omitted. The ASN.1 structure of the EnvelopedData will be

EnvelopedData ::= SEQUENCE {

version CMSVersion,

recipientInfos RecipientInfos,

encryptedContentInfo EncryptedContentInfo}

3.1.3 RecipientInfo Type

As mentioned before there are several key management techniques for encrypting content encryption keys. In lightweight CMS, only the key transfer is enough as a key management technique, and the other choices will be omitted. In order to keep the conformance to the standard, the ASN.1 choice structure of RecipientInfo is preserved.

RecipientInfo ::= CHOICE {

ktri KeyTransRecipientInfo}

3.1.4 RecipientIdentifier Type

In the standard CMS, recipient can be identified either by certificate issuer distinguished name and the certificate serial number or by subjectKeyIdentifier which uniquely identifies the certificate by a key identifier. This is actually the subjectKeyIdentifier extension of the certificate. In lightweight CMS, both choices are omitted. Since, in the first choice issuerAndSerialNumber, issuer is represented in the ASN.1 name structure. Encoding, decoding and making comparison with this structure is quite complex and time consuming. Also in order to get the subjectKeyIdentifier of a certificate, the extensions of a certificate have to be decoded and identified. In addition, this extension may not be present in some of the certificates at all. In order to get rid of the unnecessary and heavy ASN.1 encoding, decoding, comparison process, a new alternative, the choice issuerHashAndSerialNumber is introduced. The ASN.1 structure of the type IssuerHashAndSerialNumber is:

IssuerHashAndSerialNumber ::= SEQUENCE {

issuerHash OCTET String,

issuerHashAlgorithm AlgorithmIdentifier

serialNumber INTEGER}

The fields of type IssuerHashAndSerialNumber have the following meanings:

issuerHash is the result of hashing the certificate issuer with the specified hash algorithm. *issuerHashAlgorithm* is the object id of the hash algorithm used for hashing the certificate issuer. *serialNumber* is the certificate serial number which is unique for a certificate authority.

3.2 Java Card Application

3.2.1 Cryptography with Java Card

Java Card technology enables smart cards with very limited memory to run small applications, called applets that employ Java technology. It provides smart card manufacturers with a secure and interoperable execution platform that can store and update multiple applications on a single device. Java Card technology is compatible with existing smart card standards.

Java Cards can be used as Subscriber Identity Module (SIM) cards. Currently millions of mobile phones use java cards as SIM cards on most wireless networks.

The public, private key pair for asymmetric cryptography can also be created on java cards. Once the key pair is created on the java card, it is impossible to extract the private key from the card, and the private key operations are only done within the card. Except from the java card owner, nobody else can use the private key for PKI based operations.

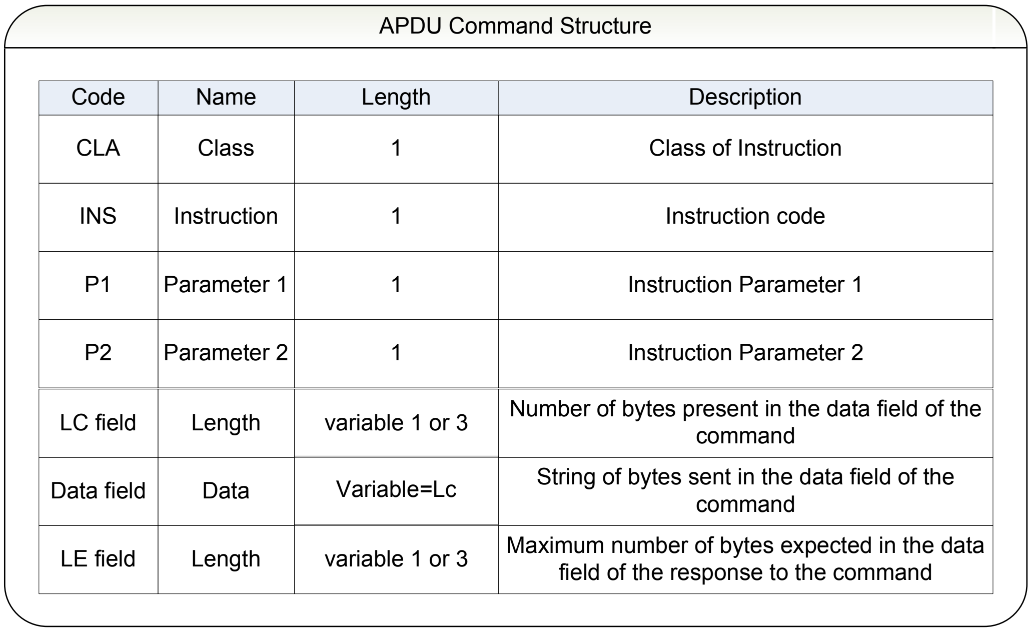
Not only the asymmetric cryptographic operations, but also symmetric cryptographic operations such as encryption, decryption and random number generation, hashing can also be done on java cards.

Java card will be used as a means of handling cryptographic operations for our project. In order to achieve this, an applet is developed which will run on java card VM. In the initialization phase of the applet an asymmetric key pair is created on the card. The certification of the public key in the card is not in the scope of the project, but it can be managed by creating a certification request in the form PKCS#10 [20] syntax file.

3.2.2 Communication with Java Card

There are two means of communication with the java card applet. The first way is communicating with APDU (Application Protocol Data Unit) commands. The message structure of APDU commands are defined in ISO/IEC 7816-4 [12] . In the APDU protocol a command is sent, it is processed and a response is sent to back. Therefore a specific response corresponds to a specific command, referred to as a command-response pair. An APDU contains either a command message or a response message, sent from the interface device to the card or conversely.

The structure of the APDU command is described in Figure 7:

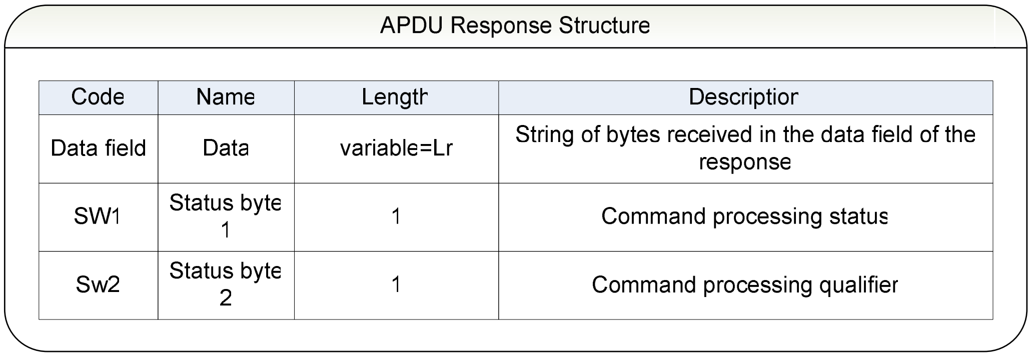


APDU Command Structure

Code	Name	Length	Description
CLA	Class	1	Class of Instruction
INS	Instruction	1	Instruction code
P1	Parameter 1	1	Instruction Parameter 1
P2	Parameter 2	1	Instruction Parameter 2
LC field	Length	variable 1 or 3	Number of bytes present in the data field of the command
Data field	Data	Variable=Lc	String of bytes sent in the data field of the command
LE field	Length	variable 1 or 3	Maximum number of bytes expected in the data field of the response to the command

Figure 7 APDU Command Structure

And the structure of the response APDU is described in Figure 8:



APDU Response Structure

Code	Name	Length	Description
Data field	Data	variable=Lr	String of bytes received in the data field of the response
SW1	Status byte 1	1	Command processing status
Sw2	Status byte 2	1	Command processing qualifier

Figure 8 APDU Response Structure

The communication between the mobile phone and java card is depicted in Figure 9:

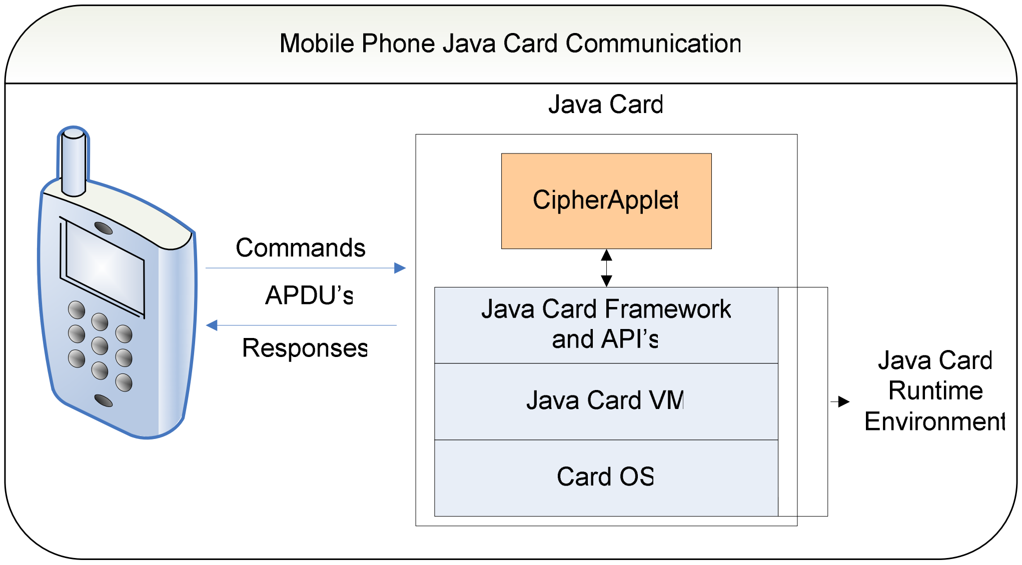

Mobile Phone Java Card Communication
Java Card
CipherApplet
Commands
APDU's
Responses
Java Card Framework and API's
Java Card VM
Card OS
Java Card Runtime Environment

Figure 9 Mobile Phone Java Card Communication

The second way for the communication with java card applet is Remote Method Invocation (RMI). This method is simpler than APDU communication, since the application running on the mobile phone can call objects on the java card applet remotely with no need to know about the details of the java card, the low level APDU communications, and it is easier to design, maintain the code.

For sure the second way of communication with RMI is simpler, and more prone to be selected. But, it is a new technology and only supported by the java cards compatible with java card API 2.2.2, which are not widespread in the market. For this reason, APDU's will be used as a means of communication between the java card and the application running on mobile phone.

3.2.3 Application Design

The application running on java card is mainly composed of two classes. The main class which is communicating with the outer world is the *CipherApplet*. The second class is named as *PKCS5Padding* which pads and unpads data according to PKCS5 [1] padding algorithm.

The UML class diagram of the java card application is depicted in Figure 10.

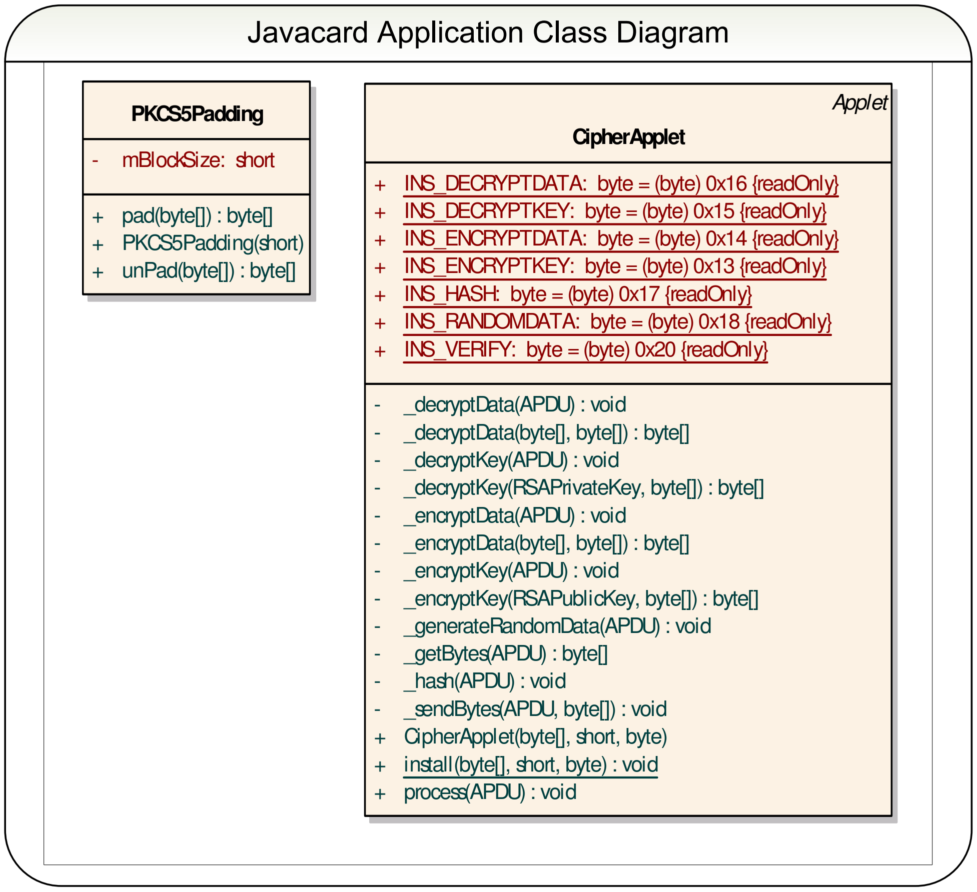

Javacard Application Class Diagram
PKCS5Padding
- mBlockSize: short
+ pad(byte[]) : byte[]
+ PKCS5Padding(short)
+ unPad(byte[]) : byte[]
Applet
CipherApplet
+ INS_DECRYPTDATA: byte = (byte) 0x16 {readOnly}
+ INS_DECRYPTKEY: byte = (byte) 0x15 {readOnly}
+ INS_ENCRYPTDATA: byte = (byte) 0x14 {readOnly}
+ INS_ENCRYPTKEY: byte = (byte) 0x13 {readOnly}
+ INS_HASH: byte = (byte) 0x17 {readOnly}
+ INS_RANDOMDATA: byte = (byte) 0x18 {readOnly}
+ INS_VERIFY: byte = (byte) 0x20 {readOnly}
- _decryptData(APDU) : void
- _decryptData(byte[], byte[]) : byte[]
- _decryptKey(APDU) : void
- _decryptKey(RSAPrivateKey, byte[]) : byte[]
- _encryptData(APDU) : void
- _encryptData(byte[], byte[]) : byte[]
- _encryptKey(APDU) : void
- _encryptKey(RSAPublicKey, byte[]) : byte[]
- _generateRandomData(APDU) : void
- _getBytes(APDU) : byte[]
- _hash(APDU) : void
- _sendBytes(APDU, byte[]) : void
+ CipherApplet(byte[], short, byte)
+ install(byte[], short, byte) : void
+ process(APDU) : void

Figure 10 Javacard Application Class Diagram

The classes will be explained in more detail in the following sections.

3.2.3.1 CipherApplet

CipherApplet is an applet which extends from javacard.framework.Applet class. It has the following methods

CipherApplet(byte[]bArray, short bOffset, byte bLength) : It is the constructor for the CipherApplet. It is executed only once in the installation phase of the applet to the java card. It has three arguments which are keeping the applet id as a byte array, the offset of the applet id in the array, and the length of the applet id respectively. In the constructor, a 1024 bit RSA key pair is also generated and stored in the java card.

process(APDU apdu): It is called by the java card runtime environment to process an incoming APDU command. According to the instruction byte in the apdu, it calls the related method. The activity diagram for this method is as in Figure 11.

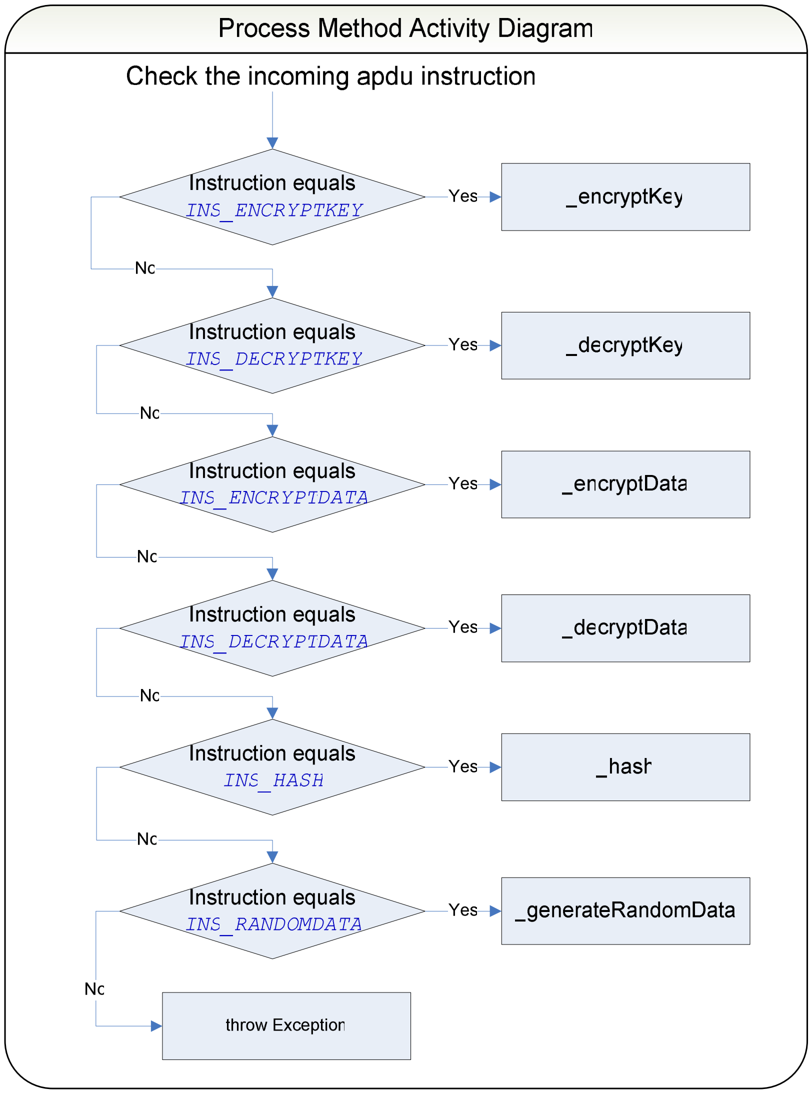

Process Method Activity Diagram
Check the incoming apdu instruction
Instruction equals INS_ENCRYPTKEY
Yes
_encryptKey
No
Instruction equals INS_DECRYPTKEY
Yes
_decryptKey
No
Instruction equals INS_ENCRYPTDATA
Yes
_encryptData
No
Instruction equals INS_DECRYPTDATA
Yes
_decryptData
No
Instruction equals INS_HASH
Yes
_hash
No
Instruction equals INS_RANDOMDATA
Yes
_generateRandomData
No
throw Exception

Figure 11 Process Method Activity Diagram

_encryptKey(APDU apdu): This method encrypts the input data with the asymmetric public key. It is used for the encryption of the symmetric key for either a given public key or the public key stored in the java card.

_decryptKey(APDU apdu): This method decrypts the encrypted input data with the asymmetric private key stored in the java card.

_encryptData(APDU apdu): This method encrypts the input data with the input symmetric key with encryption algorithm AES 128. It assumes the first 16 bytes of the input as the key and the rest as the data to be encrypted. Before encryption, the data is padded according to PKCS5 padding algorithm.

_decryptData(APDU apdu): This method decrypts the input data with the input symmetric key with AES 128. It assumes the first 16 bytes of the input as the key and the rest as the encrypted data. After decryption, the data is unpadded according to PKCS5 padding algorithm.

_generateRandomData(APDU apdu): This method generates a random data for a given length. It is used for random symmetric key generation.

_hash(APDU apdu): This method takes hash of the input data with SHA-1 hash algorithm.

3.2.3.2 PKCS5Padding

This class pads and unpads an input data for a given block size according to PKCS5 Padding algorithm. The details of the algorithm are defined in the methods below.

PKCS5Padding(short aBlockSize): This is the constructor for the PKCS5Padding class. It takes the data block size as an argument and sets this value to a member variable.

pad(byte[] aToBePadded): This method pads the input data's trailing end m-(n mod m) octets with all having value m-(n mod m), where n is the length of the input and m is the block size. In other words, the input is padded at the trailing end with one of the following strings:

01 -- if n mod m = m-1

02 02 -- if n mod m = m-2

….

m m ... m m -- if n mod m = 0

unpad(byte[] aToBeUnPadded): This method removes the padding text and returns back the original data. Padding can be removed unambiguously since all input is padded, including input values that are already a multiple of the block size, and no padding string is a suffix of another.

3.3 Midlet Application on Mobile Phone

The second part of Korugan is a java midlet application running on the mobile phone. It is an interactive application which the user can traverse through the files stored in the phone. Any file can be selected for encryption for one or more recipients. The encrypted files are stored in the mobile phone and can be sent to the intended recipients by means of MMS. The files encrypted for the mobile phone user can be decrypted, and their contents can be viewed.

The Midlet application is composed of three main components. The first component is the java midlet which browses the files, directories stored in the phone, receives the commands of the user, communicates with the applet running on the java card, prepares and resolves the CMS envelope of the selected file for the intended recipients. Second component is X.509 certificate parser. Third and the last component contain the core ASN.1 classes and lightweight CMS ASN.1 classes.

There are two versions of the java midlet. Except the asymmetric decryption operation which requires a private key residing in the java card, the other cryptographic operations such as asymmetric encryption, symmetric encryption, and decryption can be implemented also in the java midlet as well as in the java card. In the first version of the midlet, except the asymmetric decryption process all of the cryptographic operations are implemented in the java midlet. In the second version all of the cryptographic operations are implemented in the java card.

These components are explained in more detail in the following paragraphs.

3.3.1 X.509 Certificate Parser

The first component of the java midlet application is X.509 certificate parser. As explained in the Section 2.2, X.509 Certificates are ASN.1 DER encoded. In order to prepare CMS envelope for an intended recipient, the issuer, serial and public key of the recipient is necessary. And this information has to be extracted from the X.509 certificate of the recipient. There are several open source and commercial ASN.1 DER decoders available, but all of these decoders are designed for computers, and not suitable for mobile devices. Since the applications running on mobile phones can only use a limited amount of memory and processing power, a lightweight X.509 certificate parser is implemented which gets the binary form of the certificate and parse it according to the ASN.1 DER structure of X.509 Certificate.

Parsing X.509 certificates is necessary but insufficient. If somehow the private key of a user is compromised since his mobile phone is stolen, his certificate has to be canceled by the issuing CA. While preparing a CMS envelope for this user the status of his certificate has to be checked and validated. The X.509 certificate validation is really a rough process. Either Certificate Revocation Lists (CRL) [10] has to be downloaded or the status of the certificate has to be checked with the related Online Certificate Status Protocol (OCSP) [22] server. Of course, in every envelope preparation doing this process for all the recipients can bring an overhead and extra cost to the mobile phone users. Since being online with the net either for downloading a CRL or making an OCSP request will be charged to him. Nevertheless downloading the CRL or making an OCSP request is not enough, a detailed algorithm for certificate validation is described in RFC 3280 [10] . In future versions of Korugan, this parser may also include the lightweight X.509 certificate validation functionality.

The UML class diagram of the X.509 Certificate parser is depicted in Figure 12 and further details of these classes are described in the following sections.

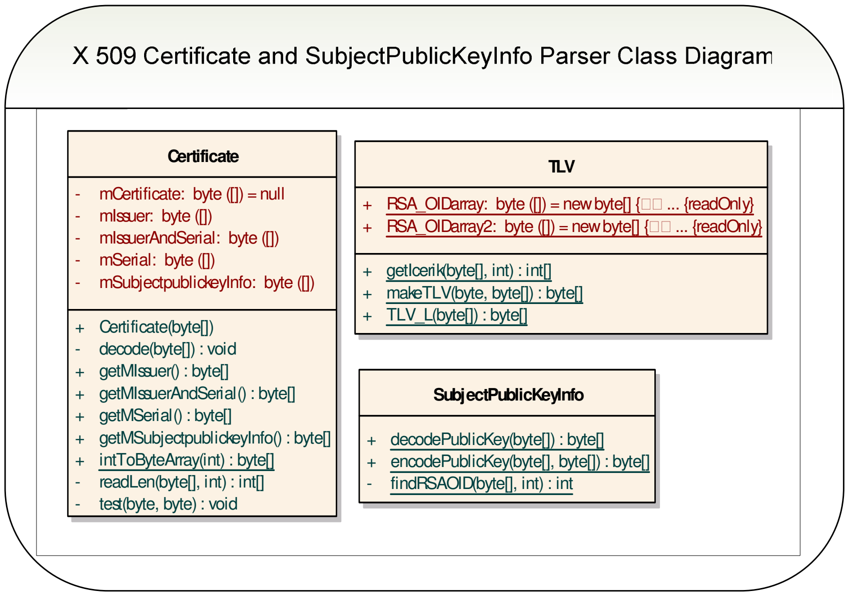

X 509 Certificate and SubjectPublicKeyInfo Parser Class Diagram
Certificate
- mCertificate: byte ([]) = null
- mIssuer: byte ([])
- mIssuerAndSerial: byte ([])
- mSerial: byte ([])
- mSubjectpublickeyInfo: byte ([])
+ Certificate(byte[])
- decode(byte[]) : void
+ getMIssuer() : byte[]
+ getMIssuerAndSerial() : byte[]
+ getMSerial() : byte[]
+ getMSubjectpublickeyInfo() : byte[]
+ intToByteArray(int) : byte[]
- readLen(byte[], int) : int[]
- test(byte, byte) : void
TLV
+ RSA_OIDarray: byte ([]) = new byte[] {□□ ... {readOnly}
+ RSA_OIDarray2: byte ([]) = new byte[] {□□ ... {readOnly}
+ getIcerik(byte[], int) : int[]
+ makeTLV(byte, byte[]) : byte[]
+ TLV_L(byte[]) : byte[]
SubjectPublicKeyInfo
+ decodePublicKey(byte[]) : byte[]
+ encodePublicKey(byte[], byte[]) : byte[]
- findRSAOID(byte[], int) : int

Figure 12 X.509 Certificate and SubjectPublicKeyInfo Parser Class Diagram

3.3.1.1 Certificate

This class is used to decode the X.509 certificate to its fields.

Certificate(byte[] aCertificate): This is the constructor method for the Certificate class. It takes the X.509 certificate as an argument in octet array and assigns it to the member variable mCertificate and calls the private method *decode*.

decode(byte[] certBytes): The X.509 certificate ASN.1 structure is defined in Section 2.2.1.1. This method parses this structure according to the DER encoding specifications described in Appendix A.

getMIssuer(): This is the getter method for the X.509 certificate issuer. The issuer is returned as an octet array.

getMSerial(): This is the getter method for the X.509 certificate serial number. The serial is returned as an octet array.

getMIssuerAndSerial(): This method encodes issuer and the serial number as an issuerAndSerial structure and returns it as an octet array.

getMSubjectpublickeyInfol(): This is the getter method for the X.509 certificate subjectpublickeyinfo field. The subjectpublickeyinfo is returned as an octet array.

3.3.1.2 SubjectPublicKeyInfo

The ASN.1 structure of subjectPublicKeyInfo is

SubjectPublicKeyInfo ::= SEQUENCE {

```
algorithm            AlgorithmIdentifier,
subjectPublicKey     BIT STRING  }
```

The RSA public key (subjectPublicKey) is comprised of the modulus (n) and the exponent (e) which is encoded as bit string. This class is used to encode SubjectPublicKeyInfo structure for a given modulus and exponent, and decode the modulus and exponent for a given subjectPublicKeyInfo.

encodePublicKey(byte[] aModulus, aExponent): This static method encodes the subjectPublicKeyInfo for a given modulus and exponent according to DER.

decodePublicKey(byte[] aSubjectPublicKeyInfo): This static method decodes the DER encoded subjectPublicKeyInfo and returns the modulus array and exponent array.

3.3.1.3 TLV

This is a utility class for encoding structures in DER format, in other words putting them in a structure comprised of tag, length and value.

makeTLV(byte tag, byte[] x): This method encodes the given input for a given tag.

TLV_L(byte[] x): This method prepares the length encoding of the input.

getIcerik(byte[] x,int bas): This method returns back the value which is DER encoded, removes the tag and the length octets.

3.3.2 Lightweight CMS ASN.1 Infrastructure

The second component of the java midlet application includes classes for basic ASN.1 types and lightweight CMS ASN.1 structures. The design of lightweight CMS ASN.1 infrastructure classes will be described in the following sections.

For the simple and constructed ASN.1 types described in Appendix A which are relevant for lightweight CMS, classes are implemented. These are core ASN.1 classes which encode and decode their corresponding types according to DER rules. Further details for the implementation of these classes are not necessary and not in the scope of this thesis.

The lightweight CMS classes all extend from abstract class ASN1Encodable. With utilizing the core ASN.1 type classes, the lightweight CMS classes can be DER encoded and decoded. ASN.1 encoding of a lightweight CMS example can be found in Appendix C.

The UML class diagram of the lightweight CMS infrastructure and further details of the classes are described in the following sections.

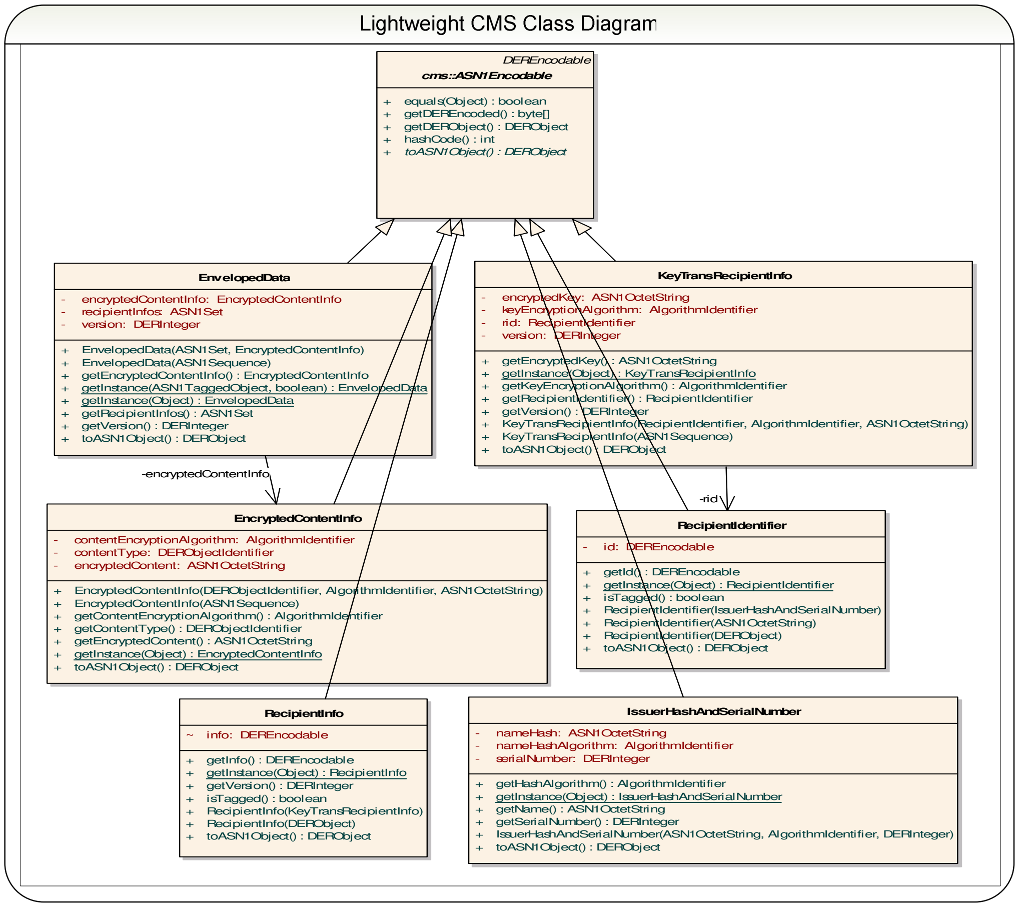

Lightweight CMS Class Diagram
DEREncodable
cms::ASN1Encodable
+ equals(Object) : boolean
+ getDEREncoded() : byte[]
+ getDERObject() : DERObject
+ hashCode() : int
+ toASN1Object() : DERObject
EnvelopedData
- encryptedContentInfo: EncryptedContentInfo
- recipientInfos: ASN1Set
- version: DERInteger
+ EnvelopedData(ASN1Set, EncryptedContentInfo)
+ EnvelopedData(ASN1Sequence)
+ getEncryptedContentInfo() : EncryptedContentInfo
+ getInstance(ASN1TaggedObject, boolean) : EnvelopedData
+ getInstance(Object) : EnvelopedData
+ getRecipientInfos() : ASN1Set
+ getVersion() : DERInteger
+ toASN1Object() : DERObject
KeyTransRecipientInfo
- encryptedKey: ASN1OctetString
- keyEncryptionAlgorithm: AlgorithmIdentifier
- rid: RecipientIdentifier
- version: DERInteger
+ getEncryptedKey() : ASN1OctetString
+ getInstance(Object) : KeyTransRecipientInfo
+ getKeyEncryptionAlgorithm() : AlgorithmIdentifier
+ getRecipientIdentifier() : RecipientIdentifier
+ getVersion() : DERInteger
+ KeyTransRecipientInfo(RecipientIdentifier, AlgorithmIdentifier, ASN1OctetString)
+ KeyTransRecipientInfo(ASN1Sequence)
+ toASN1Object() : DERObject
-encryptedContentInfo
-rid
EncryptedContentInfo
- contentEncryptionAlgorithm: AlgorithmIdentifier
- contentType: DERObjectIdentifier
- encryptedContent: ASN1OctetString
+ EncryptedContentInfo(DERObjectIdentifier, AlgorithmIdentifier, ASN1OctetString)
+ EncryptedContentInfo(ASN1Sequence)
+ getContentEncryptionAlgorithm() : AlgorithmIdentifier
+ getContentType() : DERObjectIdentifier
+ getEncryptedContent() : ASN1OctetString
+ getInstance(Object) : EncryptedContentInfo
+ toASN1Object() : DERObject
RecipientIdentifier
- id: DEREncodable
+ getId() : DEREncodable
+ getInstance(Object) : RecipientIdentifier
+ isTagged() : boolean
+ RecipientIdentifier(IssuerHashAndSerialNumber)
+ RecipientIdentifier(ASN1OctetString)
+ RecipientIdentifier(DERObject)
+ toASN1Object() : DERObject
RecipientInfo
~ info: DEREncodable
+ getInfo() : DEREncodable
+ getInstance(Object) : RecipientInfo
+ getVersion() : DERInteger
+ isTagged() : boolean
+ RecipientInfo(KeyTransRecipientInfo)
+ RecipientInfo(DERObject)
+ toASN1Object() : DERObject
IssuerHashAndSerialNumber
- nameHash: ASN1OctetString
- nameHashAlgorithm: AlgorithmIdentifier
- serialNumber: DERInteger
+ getHashAlgorithm() : AlgorithmIdentifier
+ getInstance(Object) : IssuerHashAndSerialNumber
+ getName() : ASN1OctetString
+ getSerialNumber() : DERInteger
+ IssuerHashAndSerialNumber(ASN1OctetString, AlgorithmIdentifier, DERInteger)
+ toASN1Object() : DERObject

Figure 13 Lightweight CMS Class Diagram

3.3.2.1 Asn1Encodable

All lightweight CMS classes extend from this abstract class. It has the following methods.

getDEREncoded() : This method DER encodes the object and return it as an octet array.

getDERObject() : From a given DER encoded octet array, it prepares the DER object.

toASN1Object : It is an abstract method which all the extender classes have to implement. In this method the extender classes implement how to prepare their DER object from its fields.

The rest of the classed are described below shortly. All of them implement the toASN1Object abstract method and has constructor methods comprised of their fields and getter methods for these fields. And since all of them extend from to *ASN1Encodable* class they can encoded as DER by calling *getDEREncoded* method.

EnvelopedData class is comprised of the encrypted content info and a set of recipient info. *EncryptedContentInfo* class is comprised of the encrypted content info, and content encryption algorithm identifier. *RecipientInfo* is a container class for the choice key transfer recipient info. *KeyTransRecipientInfo* class is comprised of the encrypted key, key encryption algorithm and the recipient identifier. *RecipientIdentifier* is a container class for issuer hash and serial number. Finally, *IssuerHashAndSerialNumber* class is comprised of the recipients X.509 certificate issuer hash, issuer hash algorithm and the serial number

3.3.3 Java Midlet

Upon execution of the Korugan in the mobile phone, the UI (user interface) of the application is created by the java midlet. In the UI, a user can traverse over the files stored in the mobile phone, select a lightweight CMS enveloped file for decryption, select a plain file to prepare CMS envelope.

If the user selects an enveloped file for decryption, the recipients in the enveloped are checked and if it is enveloped for the user than the decryption process starts. The decrypted file is saved in the same directory of the envelope.

If a plain file is selected for lightweight CMS envelope preparation, than the folder containing the user certificates is displayed. From this folder the intended recipients of the envelope can be selected. After the selection process is completed, the envelope preparation starts and upon completion the envelope is saved in a predefined folder. The activity diagram for the UI is depicted in Figure 14.

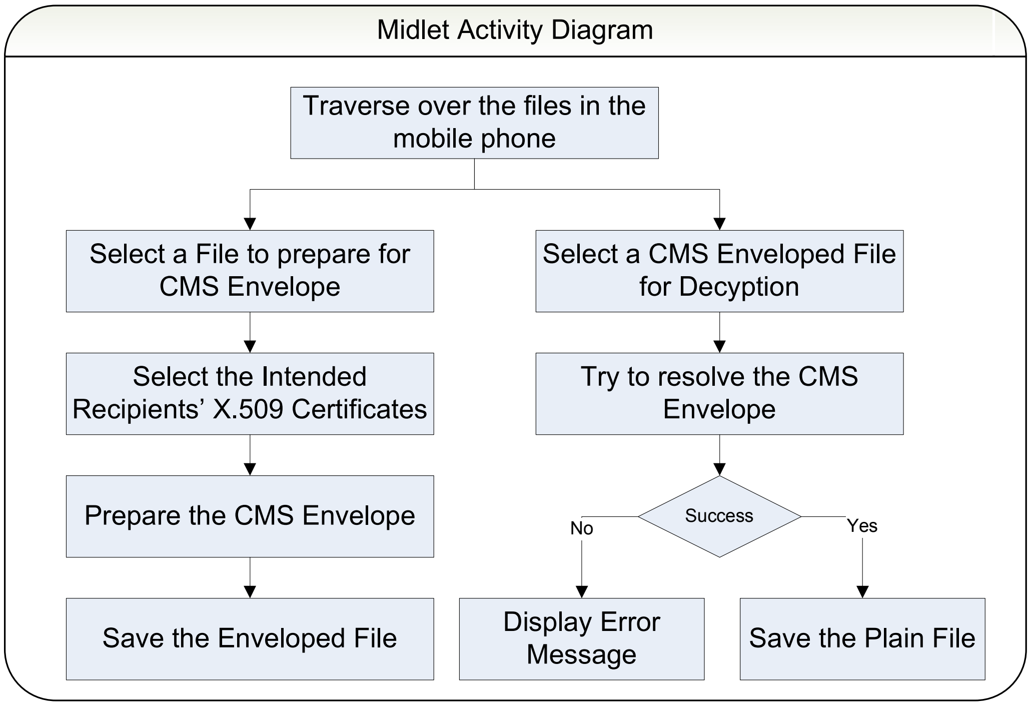

Midlet Activity Diagram
Traverse over the files in the mobile phone
Select a File to prepare for CMS Envelope
Select a CMS Enveloped File for Decyption
Select the Intended Recipients' X.509 Certificates
Try to resolve the CMS Envelope
Prepare the CMS Envelope
Success
No
Yes
Save the Enveloped File
Display Error Message
Save the Plain File

Figure 14 Midlet Activity Diagram

In this last component, in addition to UI business, the java midlet also prepares and resolves the lightweight CMS envelope which contains the encrypted content and encrypted content encryption key for each recipient. DER encoding and decoding of the lightweight CMS ASN.1 structures are implemented by utilizing the core ASN.1 classes described in Section 3.3.2. Content-encryption and key-encryption process is described in more detail in the following paragraphs.

The content-encryption key for the symmetric content-encryption algorithm (AES with 128 bits key) is randomly generated. The data to be protected is padded according to PKCS5 padding algorithm as described in Section 3.2.3.2, and then the padded data is encrypted using the content-encryption key. The encryption operation maps an arbitrary string of octets (the data) to another string of octets (the cipher text) under control of a content-encryption key. The encrypted data is included in the EnvelopedData encryptedContentInfo encryptedContent OCTET STRING.

The input to the key-encryption process, the value supplied to the recipient's key-encryption algorithm, is just the value of the content-encryption key. The key transport recipient info mentioned in Section 2.3.1.2 is used as a key management technique for each recipient of the same encrypted content. But of course, the new recipientIdentifier defined in Section 3.1.4 is used in The KeyTransRecipientInfo structure.

In the first version of the java midlet there is a class for user interface, a class for lightweight CMS envelope preparation and several classes for cryptographic operations. Except the asymmetric decryption operation which is done by the applet running in the java card, all the other necessary cryptographic operations are implemented in the midlet.

The UML class diagram of the first version of the java midlet is and their details are described in Figure 15.

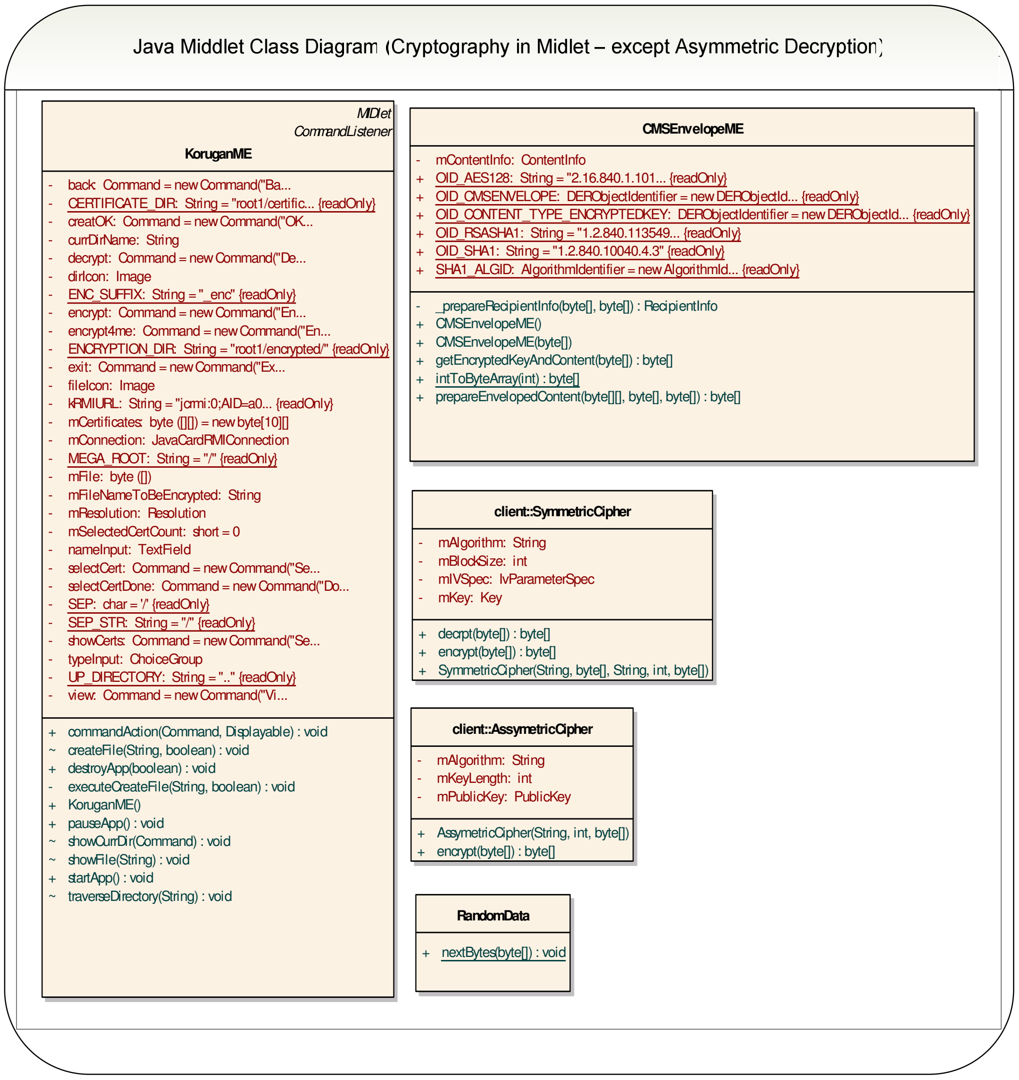
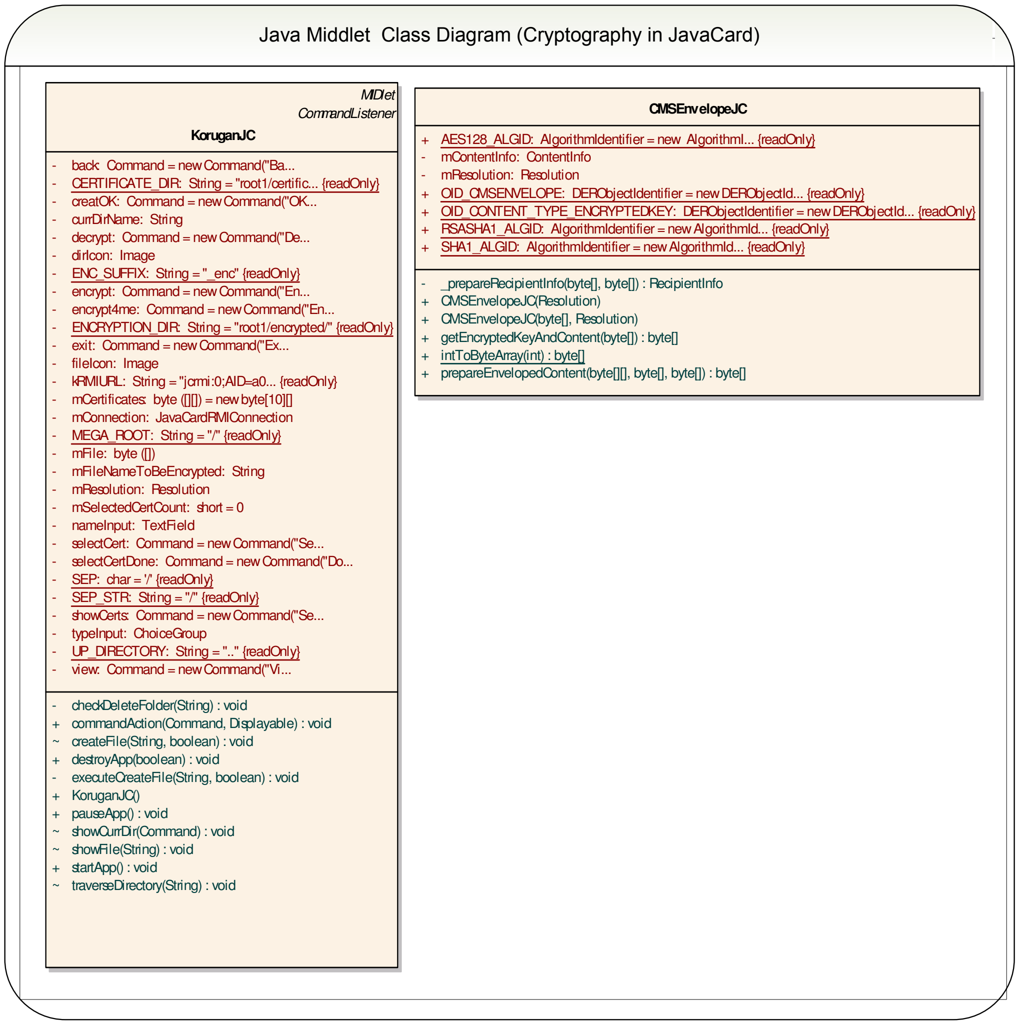

Java Middlet Class Diagram (Cryptography in Midlet – except Asymmetric Decryption)
MIDlet
CommandListener
KoruganME
- back: Command = new Command("Ba...
- CERTIFICATE_DIR: String = "root1/certific... {readOnly}
- creatOK: Command = new Command("OK...
- currDirName: String
- decrypt: Command = new Command("De...
- dirIcon: Image
- ENC_SUFFIX: String = "_enc" {readOnly}
- encrypt: Command = new Command("En...
- encrypt4me: Command = new Command("En...
- ENCRYPTION_DIR: String = "root1/encrypted/" {readOnly}
- exit: Command = new Command("Ex...
- fileIcon: Image
- KRMIURL: String = "jcrmi:0;AID=a0... {readOnly}
- mCertificates: byte ([][]) = new byte[10][]
- mConnection: JavaCardRMIConnection
- MEGA_ROOT: String = "/" {readOnly}
- mFile: byte ([])
- mFileNameToBeEncrypted: String
- mResolution: Resolution
- mSelectedCertCount: short = 0
- nameInput: TextField
- selectCert: Command = new Command("Se...
- selectCertDone: Command = new Command("Do...
- SEP: char = '/' {readOnly}
- SEP_STR: String = "/" {readOnly}
- showCerts: Command = new Command("Se...
- typeInput: ChoiceGroup
- UP_DIRECTORY: String = ".." {readOnly}
- view: Command = new Command("Vi...
+ commandAction(Command, Displayable) : void
~ createFile(String, boolean) : void
+ destroyApp(boolean) : void
- executeCreateFile(String, boolean) : void
+ KoruganME()
+ pauseApp() : void
~ showCurrDir(Command) : void
~ showFile(String) : void
+ startApp() : void
~ traverseDirectory(String) : void
CMSEnvelopeME
- mContentInfo: ContentInfo
+ OID_AES128: String = "2.16.840.1.101... {readOnly}
+ OID_CMSENVELOPE: DERObjectIdentifier = new DERObjectId... {readOnly}
+ OID_CONTENT_TYPE_ENCRYPTEDKEY: DERObjectIdentifier = new DERObjectId... {readOnly}
+ OID_RSASHA1: String = "1.2.840.113549... {readOnly}
+ OID_SHA1: String = "1.2.840.10040.4.3" {readOnly}
+ SHA1_ALGID: AlgorithmIdentifier = new AlgorithmId... {readOnly}
- _prepareRecipientInfo(byte[], byte[]) : RecipientInfo
+ CMSEnvelopeME()
+ CMSEnvelopeME(byte[])
+ getEncryptedKeyAndContent(byte[]) : byte[]
+ intToByteArray(int) : byte[]
+ prepareEnvelopedContent(byte[][], byte[], byte[]) : byte[]
client::SymmetricCipher
- mAlgorithm: String
- mBlockSize: int
- mIVSpec: IvParameterSpec
- mKey: Key
+ decrpt(byte[]) : byte[]
+ encrypt(byte[]) : byte[]
+ SymmetricCipher(String, byte[], String, int, byte[])
client::AssymetricCipher
- mAlgorithm: String
- mKeyLength: int
- mPublicKey: PublicKey
+ AssymetricCipher(String, int, byte[])
+ encrypt(byte[]) : byte[]
RandomData
+ nextBytes(byte[]) : void

Figure 15 Java Midlet Class Diagram
(Cryptography in Midlet -except Asymmetric Decryption)

In the second version of the midlet there are no classes for cryptographic operations. There are only two classes *KoruganJC* and *CMSEnvelopeJC* which have the same functionality with the classes *KoruganME* and *CMSEnvelopeME*, but all the cryptographic operations in these classes are done by the application running on java card. The UML class diagram of the second version of the java midlet is depicted in Figure 16.



```
MIDlet
CommandListener
KoruganJC

-  back: Command = new Command("Ba...
-  CERTIFICATE_DIR: String = "root1/certific... {readOnly}
-  creatOK: Command = new Command("OK...
-  currDirName: String
-  decrypt: Command = new Command("De...
-  dirIcon: Image
-  ENC_SUFFIX: String = "_enc" {readOnly}
-  encrypt: Command = new Command("En...
-  encrypt4me: Command = new Command("En...
-  ENCRYPTION_DIR: String = "root1/encrypted/" {readOnly}
-  exit: Command = new Command("Ex...
-  fileIcon: Image
-  KRMIURL: String = "jcrmi:0;AID=a0... {readOnly}
-  mCertificates: byte ([][]) = new byte[10][]
-  mConnection: JavaCardRMIConnection
-  MEGA_ROOT: String = "/" {readOnly}
-  mFile: byte ([])
-  mFileNameToBeEncrypted: String
-  mResolution: Resolution
-  mSelectedCertCount: short = 0
-  nameInput: TextField
-  selectCert: Command = new Command("Se...
-  selectCertDone: Command = new Command("Do...
-  SEP: char = '/' {readOnly}
-  SEP_STR: String = "/" {readOnly}
-  showCerts: Command = new Command("Se...
-  typeInput: ChoiceGroup
-  UP_DIRECTORY: String = ".." {readOnly}
-  view: Command = new Command("Vi...

-  checkDeleteFolder(String) : void
+  commandAction(Command, Displayable) : void
~  createFile(String, boolean) : void
+  destroyApp(boolean) : void
-  executeCreateFile(String, boolean) : void
+  KoruganJC()
+  pauseApp() : void
~  showCurrDir(Command) : void
~  showFile(String) : void
+  startApp() : void
~  traverseDirectory(String) : void
```

```
CMSEnvelopeJC

+  AES128_ALGID: AlgorithmIdentifier = new AlgorithmI... {readOnly}
-  mContentInfo: ContentInfo
-  mResolution: Resolution
+  OID_CMSENVELOPE: DERObjectIdentifier = new DERObjectId... {readOnly}
+  OID_CONTENT_TYPE_ENCRYPTEDKEY: DERObjectIdentifier = new DERObjectId... {readOnly}
+  RSASHA1_ALGID: AlgorithmIdentifier = new AlgorithmId... {readOnly}
+  SHA1_ALGID: AlgorithmIdentifier = new AlgorithmId... {readOnly}

-  _prepareRecipientInfo(byte[], byte[]) : RecipientInfo
+  CMSEnvelopeJC(Resolution)
+  CMSEnvelopeJC(byte[], Resolution)
+  getEncryptedKeyAndContent(byte[]) : byte[]
+  intToByteArray(int) : byte[]
+  prepareEnvelopedContent(byte[][], byte[], byte[]) : byte[]
```

Figure 16 Java Midlet Class Diagram (Cryptography in Javacard)

3.3.3.1 KoruganME

This class receives the commands from the mobile phone user, and triggers the corresponding function. The defined commands are view, show certificates, select certificate, certificate selection completed, encrypt, decrypt, back and exit.

It extends from the abstract class javax.microedition.midlet.MIDlet and implements the interface javax.microedition.lcdui.CommandListener. Some of the methods are described in more detail.

commandAction(Command c, Displayable d): This method is the implementation of the corresponding method in the interface CommandListener. It receives a command event that has occurred on a Displayable d. If the received command is *view* depending on the displayable, either the content of a file is displayed, or traveled through out the directory structure. If the command is *select certificate*, the selected certificate is stored in a member variable. If the command is *certificate selection completed*, the selected file is enveloped for the selected certificates by calling the method prepareEnvelopedContent method of the CMSEnvelopeME class. If the command is *encrypt*, the selected file is read in an array and the X.509 certificates are listed and the user is asked to select the recipients. If the command is *decrypt*, first the encrypted content and the encrypted key for the mobile phone user is extracted from the envelope by means of the CMSEnvelopeME class, and then the content encryption key is decrypted by the APDU communication with the java card applet, and then content is decrypted by the calling the decrypt method of the class SymmetricCipher. The decrypted content is then saved as another file. *Back* command returns the operation to the previous screen and *exit* command exits from the application.

showCurrDir(Command aCommand): This method lists all the files and directories in the current directory and according to the command argument sets the relevant commands in the screen.

showFile(String fileName): The file is displayed in a text viewer on the screen of the phone.

3.3.3.2 CMSEnvelopeME

This class prepares a lightweight CMS envelope of a file for a given number of recipient X.509 certificates, and returns back the encrypted content and encrypted content encryption key of an envelope for a given recipient certificate.

CMSEnvelopeME(byte[] aEnvelope): This constructor method takes the envelope in octet array and creates the ContentInfo object out of it.

prepareEnvelopedContent(byte[][] aCertificates, byte[] aEncryptionKey, byte[] aPlainText): This method first prepares the recipient infos by calling the method _prepareRecipientInfo. And then encrypts the plain text by calling SymmetricCipher class decrypt method. Finally creates the envelope by combining the encrypted content with the recipient information.

_prepareRecipientInfo(byte[] aCertificate, byte[] aKey): This method prepares the recipient information for a recipient by using the X.509 certificate and the content encryption key. It extracts the issuer, serial and the subjectPublicKeyInfo from the certificate by certificate parser, and then takes the hash of the issuer according to SHA-1 hash algorithm. And then encrypts the content encryption key with AsymmetricCipher class encrypt method. Finally, it packs the issuer hash, serial and encrypted key to form a RecipientInfo object.

getEncryptedKeyAndContent(byte[] aCertificate): This method iterates over the recipients of an envelope for the given certificate. If the envelope is for the given recipient then this method returns back the encrypted key and encrypted content.

3.3.3.3 SymmetricCipher

This class is used to encrypt and decrypt an input file for a given encryption algorithm with a given secret key.

SymmetricCipher(String aAlg, byte[] aKeyBits,String aKeyAlg,int aBlockSize,byte[] aIV) : This is the constructor method with the parameters symmetric encryption algorithms, the symmetric key, key algorithm, symmetric algorithm block size and the initialization vector. In the *Korugan* implementation the symmetric algorithm AES CBC with PKCS5 padding is used with the key size 16.

encrypt(byte[] aPlainText): This method encrypts the plain text according to the specified algorithm and the key in the constructor.

decrypt(byte[] aCipherText): This method decrypts the cipher text according to the specified algorithm and the key in the constructor.

3.3.3.4 AsymmetricCipher

This class only encrypts an input file for a given encryption algorithm with a given public key.

AssymetricCipher(String aAlg, int aKeyLength,byte[] aPublicKeyBits): This is the constructor method with the parameters asymmetric encryption algorithm, the key length and the key itself.

encrypt(byte[] aPlainText): This method encrypts the plain text according to the specified algorithm and the key in the constructor.

3.3.3.5 RandomData

This class is used to generate random symmetric key

nextBytes(byte abyte0[]): This method generates a pseudo random data for a given length.

CHAPTER 4

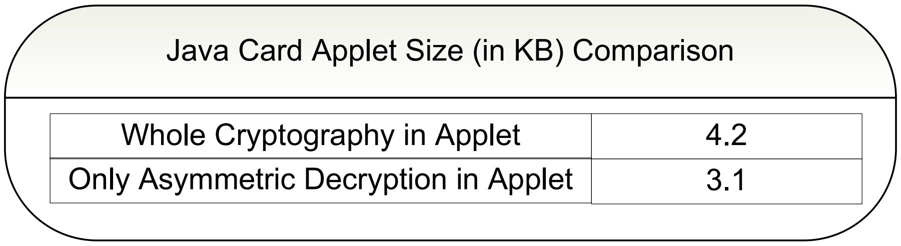
PERFORMANCE ANALYSIS

There are several dozens of mobile phone manufacturers and hundreds of different mobile phone models in the market [24] [25] These mobile phones differ from each other according to their technical specifications such as operating system, memory size, data storage media size, processing power etc. *Korugan* can be used by only the mobile phones which support java and use SIM card having java card capabilities. One of the motivations to implement two versions of *Korugan* is to get rid of the dependency to the limitations of the mobile phone by handling all cryptographic operations in the java card. A second motivation is to compare the performance of the application when the cryptographic operations are done in the mobile phone and in the java card.

Another issue of interest is the benefits of *Korugan* by introducing the lightweight CMS. If *Korugan* was implemented according to standard CMS, what would be the size and performance of the application? In order to make this comparison, a third version of *Korugan* is implemented which is preparing the envelope according to standard CMS. In order to answer these questions, *Korugan* versions will be compared in terms of size and performance in the following paragraphs.

The installed application size is one of the important criteria for the mobile phones which have limited storage media. Table 6 and Table 7 compare media storage size for two versions of *Korugan* and *Korugan* implementation with standard CMS.

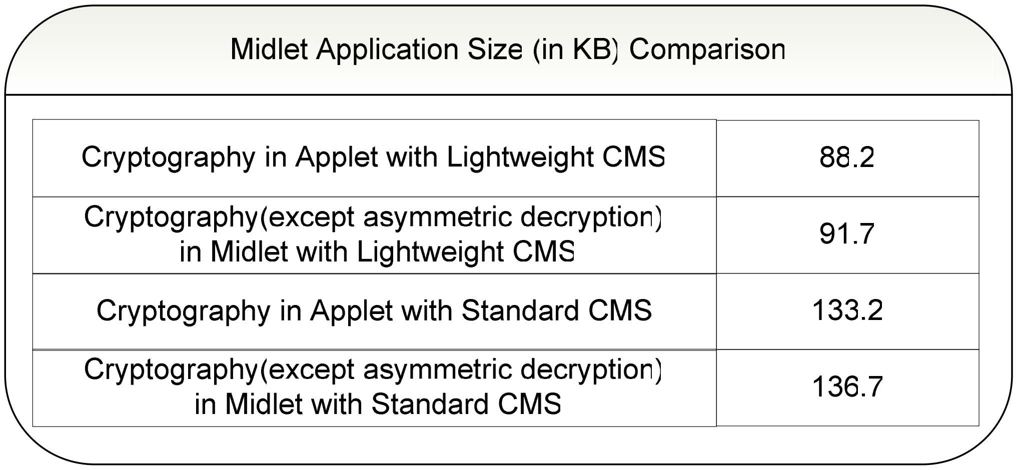
Table 6 Java Card Applet Size (in KB) Comparison



Java Card Applet Size (in KB) Comparison	
Whole Cryptography in Applet	4.2
Only Asymmetric Decryption in Applet	3.1

The size of the java card applet is almost the same for the two versions. The whole cryptography for the CMS envelope preparation and resolution can be done in the java card. But the memory limitations of the java card for symmetric encryption of big sized data must be considered as a drawback of this option.

Table 7 Midlet Application Size (in KB) Comparison



Midlet Application Size (in KB) Comparison	
Cryptography in Applet with Lightweight CMS	88.2
Cryptography(except asymmetric decryption) in Midlet with Lightweight CMS	91.7
Cryptography in Applet with Standard CMS	133.2
Cryptography(except asymmetric decryption) in Midlet with Standard CMS	136.7

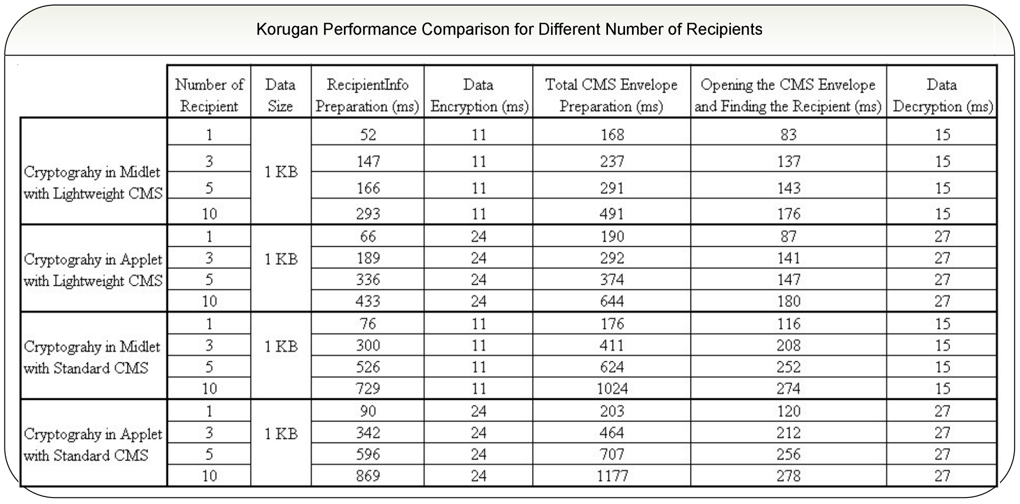
The java midlet size differs by 3.5 KB according to the application handling the cryptographic operations. Due to the drawback of the java card application in symmetric ciphering mentioned in the above paragraph, if the mobile phone is capable of handling cryptographic operations, it is a better media for handling these operations. Nevertheless, asymmetric decryption has always to be done by the java card.

The application size decreases approximately by 35 % (45 KB) with lightweight CMS in comparison to standard CMS. For the old generation mobile phones, this may be a significant amount of storage size, but for the new generations it may not be so important since the storage media size is increasing in chunks of MB as the time passes.

A second comparison criterion is the average speed of the envelope preparation and extraction for the CMS versions. In order to make the comparison, the midlets of *Korugan* versions are deployed into a Nokia N73 mobile phone. Unfortunately, the applet could not be deployed into the javacard due to the insufficient support from the mobile phone operators. The comparison data is collected as an average of 10 runs on the mobile phone. For the versions which cryptography is handled in the javacard applet and the asymmetric decryption operation, the cryptographic operations' durations are gathered from the javacard emulation environment.

In Table 8, *Korugan* versions are compared according to envelopes prepared for different numbers of recipients. The envelope preparation and extraction process are divided into components and the execution durations are measured for each component separately. When the cryptographic operations are handled in the mobile phone, CMS envelope preparation and extraction is apparently faster because of the memory and processing power limitations of the java card. The recipientInfo preparation duration is faster in the lightweight CMS versions, since the issuer and the serial field values from X.509 certificates are extracted in a more efficient way and unnecessary fields are not parsed. With the increasing number of the recipients in the envelope, the difference in the preparation time increases between versions.

Table 8 Korugan Performance Comparison for Different Number of Recipients



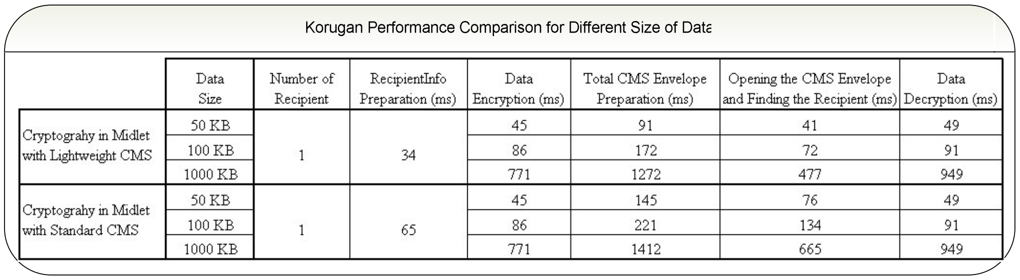
Korugan Performance Comparison for Different Number of Recipients

	Number of Recipient	Data Size	RecipientInfo Preparation (ms)	Data Encryption (ms)	Total CMS Envelope Preparation (ms)	Opening the CMS Envelope and Finding the Recipient (ms)	Data Decryption (ms)
Cryptograhy in Midlet with Lightweight CMS	1	1 KB	52	11	168	83	15
	3		147	11	237	137	15
	5		166	11	291	143	15
	10		293	11	491	176	15
Cryptograhy in Applet with Lightweight CMS	1	1 KB	66	24	190	87	27
	3		189	24	292	141	27
	5		336	24	374	147	27
	10		433	24	644	180	27
Cryptograhy in Midlet with Standard CMS	1	1 KB	76	11	176	116	15
	3		300	11	411	208	15
	5		526	11	624	252	15
	10		729	11	1024	274	15
Cryptograhy in Applet with Standard CMS	1	1 KB	90	24	203	120	27
	3		342	24	464	212	27
	5		596	24	707	256	27
	10		869	24	1177	278	27

Envelope extraction time for a recipient depends on two factors. The first factor is the position of the recipient in the *RecipientInfos* structure and the second factor is the comparison criteria to find a recipient. In Table 8, the extraction is done for the last recipient in the *RecipientInfos* structure in the envelope. As described in Section 3.1.4, the recipient is identified according to the issuerAndSerialNumber structure in standard CMS. The ASN.1 syntax of issuer structure in issuerAndSerialNumber is Name, which can be composed of many different attributes such as common name, organization, organizational unit, domain component, locality, country etc. The identification, encoding, decoding and comparison of these attributes are bypassed in lightweight CMS with the introduction of issuerHashAndSerial structure which results in the performance increase in Korugan compared to standard CMS implementations. With increasing number of comparisons, the performance of lightweight CMS is getting more obvious.

Since the javacard is not suitable for bulk data symmetric encryption, in Table 9 only Korugan midlet versions are compared according to size of the data to be enveloped. Due to the reasons described in the above paragraph, recipientInfo preparation and envelope extraction duration is shorter in lightweight CMS version. With the increasing size of the enveloped data, the envelope preparation and extraction duration is also increasing.

Table 9 Korugan Performance Comparison for Different Size of Data



Korugan Performance Comparison for Different Size of Data

	Data Size	Number of Recipient	RecipientInfo Preparation (ms)	Data Encryption (ms)	Total CMS Envelope Preparation (ms)	Opening the CMS Envelope and Finding the Recipient (ms)	Data Decryption (ms)
Cryptograhy in Midlet with Lightweight CMS	50 KB	1	34	45	91	41	49
	100 KB			86	172	72	91
	1000 KB			771	1272	477	949
Cryptograhy in Midlet with Standard CMS	50 KB	1	65	45	145	76	49
	100 KB			86	221	134	91
	1000 KB			771	1412	665	949

Another comparison criterion is the limit of the number of recipients in the envelope. For each recipient, the size of the envelope is increasing about 185 bytes. Since the data is encrypted once for all of the recipients, the number of recipients does not impact data encryption process. As a result, the number of recipients in an envelope is limited with the size of storage media of the mobile phone which it will be stored.

CHAPTER 5

CONCLUSION

During the last decade, mobile phones have become an indispensable part of our daily life. Nowadays, mobile phones comprise of several features, such as music player, camera, camcorder, e-mail client, etc. Several types of documents and multimedia content are being shared widely by means of MMS and e-mail between mobile phone users. Since the files in the mobile phone are generally not encrypted and the standard on-the-air encryption is very weak, these files can be easily captured via several types of attacks.

Despite commonly accepted and severe privacy concerns in mobile phone communication, there are no widely used and effective security measures against these privacy breaches. In order to prevent these breaches, the low level security standards of the network can be bypassed. The security of the multimedia content in the mobile phones can be provided from end to end by enveloping them with CMS. CMS is a standard published by IETF for protecting messages cryptographically. Any form of digital data can be digitally signed, encrypted and enveloped by using CMS which is architected around certificate-based key management.

Mobile phones have limited storage media, memory and processing power as compared to computers. Since the CMS ASN.1 structure is quite complex, its implementation is costly in the mobile phones. In this thesis, in order to meet the performance restrictions of mobile phones a lightweight CMS is proposed. The lightweight CMS is used to envelope the files to be stored in mobile phones or to be shared between users by means of MMS and e-mail. In lightweight CMS, unnecessary syntax components irrelevant for enveloping are removed and ASN.1 Name related syntax is simplified. A sample mobile application, which makes use of the lightweight CMS, named *Korugan* is developed. Using *Korugan,* any file in the mobile phone can be digitally enveloped for any number of recipients. In the envelope, the plain file is encrypted with symmetric encryption and the symmetric encryption key is encrypted with the public keys of each recipient extracted from their X.509 certificates. The recipients can resolve the envelope using their private keys, and obtain the plain file using *Korugan*.

In *Korugan*, cryptographic operations can be implemented either in the mobile phone or in the SIM card except the asymmetric decryption which requires a private key operation stored on the SIM card. Two different versions of *Korugan* are implemented. These implementations differ according to the media where cryptographic operations are handled. In order to make comparisons, another base implementation is also developed which uses standard CMS. The *Korugan* versions and the base implementation are compared to each other in terms of application size and performance. When the cryptographic operations are handled in the mobile phone, CMS envelope preparation and extraction are obviously faster due to the memory limitations of the SIM card. The application size and the envelope preparation duration decrease significantly when the lightweight CMS is employed in *Korugan* instead of standard CMS.

Lightweight CMS proposes syntax only for data envelopment for any number of recipients, although standard CMS enables encapsulation syntax for digital data signing, encryption for local storage, digestion for content integrity, authentication for content integrity for any number of recipients in addition to enveloping data. As the secret key distribution method, among the options key transfer, key agreement, key encryption key and password transfer which are available in standard CMS, lightweight CMS only supports the option key transfer.

Despite the privacy of the voice is not provided by *Korugan*; nevertheless it can be a good candidate for providing the privacy of the stored and transmitted huge amount of data by means of MMS and e-mail within the mobile phone.

REFERENCES

[1] RFC 2459 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

[2] ITU-T X.509 The Directory: Public-key and Attribute Certificate Frameworks

[3] RFC 3852 Cryptographic Message Syntax (CMS)

[4] Understanding PKI second Edition Concepts, Standards and Deployment Considerations, Carliste Adams and Steve Lloyd

[5] Planning for PKI, Best Practice Guide for Deploying Public Key Infrastructure, Russ Housley, Tim Polk

[6] ITU-T X.690 ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)

[7] ITU-T Recommendation X.501, Information Technology - Open Systems Interconnection – The directory models, 1993

[8] ASN.1 Communication between Heterogeneous Systems, Olivier Dubuisson, http://www.oss.com/asn1/booksintro.html , Last Access Date: 21.08.2007

[9] RFC 3280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (updates RFC 2459)

[10] ITU-T Recommendation X.520, Information Technology - Open Systems Interconnection – The directory : Selected Attribute Types, 1993

[11] RFC 3161 Internet X.509 Public Key Infrastructure Time-Stamp Protocol(TSP)

[12] ISO/IEC 7816-4 : Interindustry Command for interchange

[13] RFC 2898 PKCS #5 : Password-Based Cryptography Specification Version 2.0

[14] CCITT. Recommendation X.200: Refrence Model of Open Systems Inteconnection for CCITT Applications. 1984

[15] CCITT. Recommendation X:208: Specification of Abstract Syntax Notation One (ASN.1). 1988

[16] CCITT. Recommendation X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). 1988

[17] RFC 3850 Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling, July 2004

[18] RFC 3851 Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification, July 2004

[19] PKCS #12 v1.0: Personal Information Exchange Syntax, RSA Laboratories, June 24, 1999

[20] PKCS #10 v1.7: Certification Request Syntax Standard RSA Laboratories, May 26, 2000

[21] RFC 1422 - Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management

[22] RFC 2560 X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP, June 1999

[23] A Layman's Guide to a Subset of ASN.1, BER, and DER, Burton S. Kaliski Jr., RSA Data Security, Inc., Redwood City, CA, June 3, 1991

[24] Nokia, www.nokia.com.tr , Last Access Date: 21.08.2007

[25] Sony Ericsson, www.sonyericsson.com , Last Access Date: 21.08.2007

[26] LG, www.lgmobile.com , Last Access Date: 21.08.2007

[27] Siemens, www.siemens.com , Last Access Date: 21.08.2007

[28] GSM, Global System for Mobile Communications, http://en.wikipedia.org/wiki/GSM , Last Access Date: 21.08.2007

[29] IETF, Internet Engineering Task Force, www.ietf.org , Last Access Date: 21.08.2007

[30] SIM, Subscriber Identity Module, http://en.wikipedia.org/wiki/Subscriber_Identity_Module , Last Access Date: 21.08.2007

[31] PKI: Implementing and Managing E-Security , Andrew Nash, William Duane, Celia Joseph, Derek Brink, Osborne/McGraw-Hill, 2001

[32] Cryptography and Public Key Infrastructure on the Internet , Klaus Schmeh, John Wiley & Sons, 2003

[33] Public-Key Cryptography , Salomaa, Arto, Springer-Verlag, 1990

[34] Authentication: From Passwords to Public Keys , Richard E. Smith , Addison-Wesley, 2002

[35] Message Authentication Code, D.R. Stinson, Cryptography - Theory and Practice, CRC Press, Boca Raton, 1995.

[36] ITU-T, International Telecommunication Union, www.itu.int , Last Access Date: 21.08.2007

APPENDIX A

GUIDE FOR SUBSET OF ASN.1, BER AND DER

One of the most complex systems today, and one that also involves a great abstraction is Open Systems Inteconnection (OSI) [14] . OSI is an internationally standardized architecture that manages the interconnection of computers from the physical layer up to the user application layer. OSI's method of specifying abstract objects is called Abstract Syntax Notation One (ASN.1) [15] and a set of rules for representing abstract objects as strings of ones and zeros is called Basic Encoding Rules (BER) [16] For further information about ASN.1, the book 'ASN.1 Communication between Heterogeneous Systems, Olivier Dubuisson' [8] and the 'Layman's Guide to a Subset of ASN.1, BER, and DER' [23] can be referred.

In ASN.1, a type is a set of values; a value of a given ASN.1 type is an element of that type's set of values. There are 4 different ASN.1 types. These are simple types which are atomic that have no components, structures types which have components, tagged types which are derived from other types, and other types. CHOICE and ANY types are defined as other types.

Every ASN.1 type other than CHOICE and ANY has a tag. ASN.1 types are abstractly the same if and only if their tag numbers are the same. There are four different classes of tag.

universal for types whose meaning is the same in all applications; these types are only defined in X.208 [15]

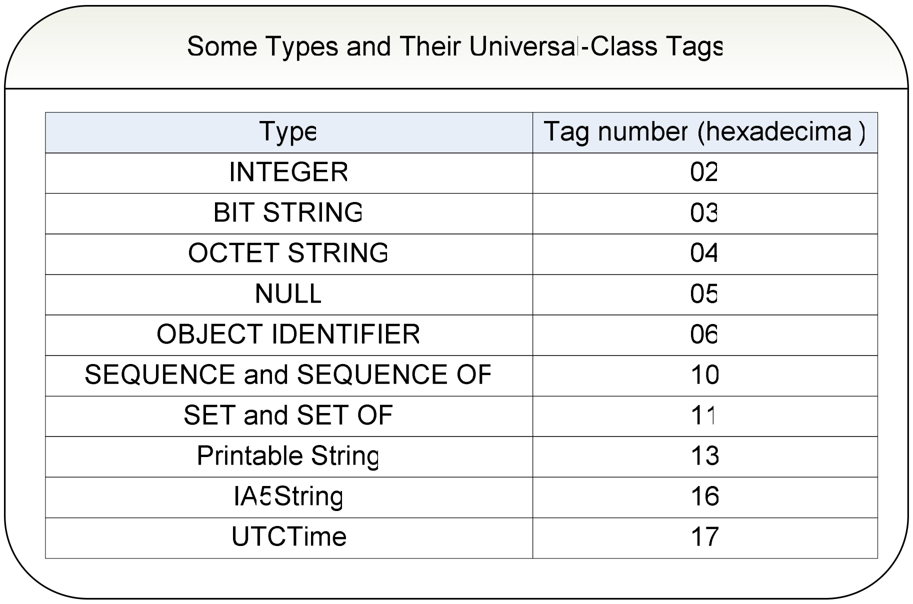
application for types whose meaning is specific to an application, such as X.500 directory services; types in two different applications may have the same application-specific tag and different meanings.

private for types whose meaning is specific to a given enterprise.

context-specific for types whose meaning is specific to a given structured type. Component types in two different structured types may have the same tag and different meanings

The types with universal tags are defined in X.208. [15] Table 9 lists some ASN.1 types and their universal class tags:

Table 10 Some Types and Their Universal-Class Tags



Some Types and Their Universal-Class Tags

Type	Tag number (hexadecimal)
INTEGER	02
BIT STRING	03
OCTET STRING	04
NULL	05
OBJECT IDENTIFIER	06
SEQUENCE and SEQUENCE OF	10
SET and SET OF	11
Printable String	13
IA5String	16
UTCTime	17

Simple Types

They are atomic types which those not consist of components.

BIT STRING is an arbitrary string of bits.

IA5String is an arbitrary string of IA5(ASCII) characters.

INTEGER is an arbitrary integer.

NULL is a null value.

OBJECT IDENTIFIER is an object identifier which is a sequence of integer components that identify an object.

OCTET STRING is an arbitrary string of octets (eight-bit values).

PrintableString is an arbitrary string of printable characters.

UTCTime is Greenwich Mean Time (GMT) value.

Structured Types

Structured types consists of components.

SEQUENCE is an ordered collection of one or more types.

SEQUENCE OF is an ordered collection of zero or more occurrences of a given type.

SET is an unordered collection of one or more types.

SET OF is an unordered collection of zero or more occurrences of a given type.

Other Types

CHOICE and *ANY* are the other ASN.1 types. The CHOICE states a union of one or more alternatives. The ANY states an arbitrary value of an arbitrary type.

Basic Encoding Rules (BER)

The Basic Encoding Rules for ASN.1, describes the way how to represent any ASN.1 value as an octet string.

There are three different methods to encode a value with BER. These methods change according to the type and length of the value. The first method is for primitive types with definitive length, the second method is for constructed types again with definitive length, the third and the last one is for constructed types with indefinite length.

ASN.1 BER encoding may have three or four parts depending on the encoding method. These parts are:

Identifier octets: These octets identify the type of the ASN.1 value, and indicate whether the method is primitive or constructed.

Length octets: If the length of the value is known than these octets identify the length of the ASN.1 value, else they indicate that the length is indefinite.

Contents octets: If the ASN.1 value is primitive than these octets represent the value. If the value is not primitive than these octets give the concatenation of the BER encodings of the components of the value.

End-of-contents octets: For the constructed, indefinite-length method, these denote the end of the contents. For the other methods, these are absent.

The three encoding methods are described as follows.

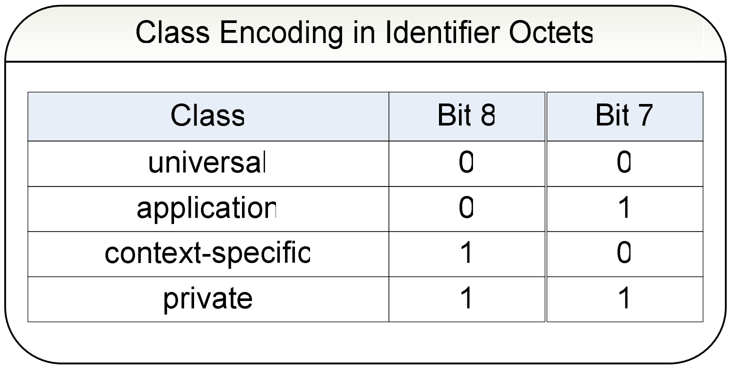
Primitive, definite-length method

This method is used in simple types and it requires that the length of the value be known in advance. The parts of the BER encoding are described as follows:

Identifier octets: There are two forms: low tag number (for tag numbers between 0 and 30) and high tag number (for tag numbers 31 and greater).

In Low-tag-number form it is one octet. Bits 8 and 7 specify the class, bit 6 has value "0," indicating that the encoding is primitive, and bits 5–1 give the tag number.

Table 11 Class Encoding in Identifier Octets



Class Encoding in Identifier Octets

Class	Bit 8	Bit 7
universal	0	0
application	0	1
context-specific	1	0
private	1	1

In High-tag-number form it is two or more octets. First octet is as in low-tag-number form, except that bits 5–1 all have value "1." Second and the following octets give the tag number in base 128.

Length octets: There are two forms: short (for lengths between 0 and 127), and long definite (for lengths between 0 and $2^{1008}–1$).

Short form is one octet. Bit 8 has value "0" and bits 7–1 give the length.

Long form is two to 127 octets. Bit 8 of first octet has value "1" and bits 7–1 give the number of additional length octets. Second and following octets give the length in base 256.

Contents octets: These give a concrete representation of the value (or the value of the underlying type, if the type is derived by implicit tagging

Constructed, definite-length method

This method applies to simple string types, structured types, types derived from simple string types and structured types by implicit tagging, and types derived from anything by explicit tagging. It requires that the length of the value be known in advance. The parts of the BER encoding are as follows:

Identifier octets: It is as described in primitive definitive length method, except that bit 6 has value "1," indicating that the encoding is constructed.

Length octets: It is as described in primitive definitive length method.

Contents octets: The concatenation of the BER encodings of the components of the value:

- For simple string types and types derived from them by implicit tagging, the concatenation of the BER encodings of consecutive substrings of the value.
- For structured types and types derived from them by implicit tagging, the concatenation of the BER encodings of components of the value.
- For types derived from anything by explicit tagging, the BER encoding of the underlying value.

Constructed, indefinite-length method

This method applies to simple string types, structured types, types derived simple string types and structured types by implicit tagging, and types derived from anything by explicit tagging. It does not require that the length of the value be known in advance. The parts of the BER encoding are as follows:

Identifier octets: It is as described in constructed, definitive length method.

Length octets: One octet, 80.

Contents octets: As described in constructed, definitive length method

End-of-contents octets: Two octets, 00 00.

Distinguished Encoding Rules (DER)

The DER is a subset of BER. In DER, there is only one representation of any ASN.1 value. For DER encoding the following restrictions apply:

- When the length is between 0 and 127, the short form of length must be used
- When the length is 128 or greater, the long form of length must be used, and the length must be encoded in the minimum number of octets.
- For simple string types definite-length method must be used.
- For structured types, the constructed, definite-length method must be employed.

Notation and Encoding of Some Types

Some of the ASN.1 types notation and the encoding of the values under DER is described in the following sections.

ANY

The ANY type denotes an arbitrary value of an arbitrary type. The arbitrary type is possibly defined in the registration of an object identifier or associated with an integer index. The ASN.1 notation for the ANY type is *ANY [DEFINED BY identifier]*.

The DER encoding of an *ANY* value is the DER encoding of the actual value.

BIT String

The BIT String type denotes an arbitrary string of bits (ones and zeroes). The ASN.1 notation for the Bit String type is *BIT STRING*.

In DER encoding, the first contents octet gives the number of bits by which the length of the bit string is less than the next multiple of eight (this is called the "number of unused bits"). The second and following contents octets give the value of the bit string,converted to an octet string.

1. The bit string is padded after the last bit with zero to seven bits of any value to make the length of the bit string a multiple of eight. If the length of the bit string is a multiple of eight already, no padding is done.
2. The padded bit string is divided into octets. The first eight bits of the padded bit string become the first octet, bit 8 to bit 1, and so on through the last eight bits of the padded bit string.

For example the DER encoding of the bit string "011011100101110111" is 03 04 06 6e 5d c0.

CHOICE

The CHOICE type denotes a union of one or more alternatives. The ASN.1 notation for the CHOICE type is

CHOICE {

[identifier$_1$] Type$_1$,

…,

[identifier$_n$] Type$_n$ }

The DER encoding of a CHOICE value is the DER encoding of the chosen alternative.

IA5String

The IA5String type denotes an arbitrary string of ASCII characters. The ASN.1 notation for the IA5String type is *IA5String*.

For example, the DER encoding of the IA5String value "test1@rsa.com" is 12 0d 74 65 73 74 31 40 72 73 61 2e 63 6f 6d.

INTEGER

The INTEGER type denotes an arbitrary integer. The ASN.1 notation for the INTEGER type is *INTEGER [{ identifier*$_1$*(value*$_1$*) … identifier*$_n$*(value*$_n$*) }]*. The contents octets give the value of the integer in base 256.

For example the DER encoding of 127 is 02 02 00 7F.

NULL

The NULL type denotes a null value. The ASN.1 notation for the NULL type is *NULL*.

For example, the DER encoding of a NULL value is always 05 00.

OBJECT IDENTIFIER

The OBJECT IDENTIFIER type denotes an object identifier, which is a sequence of integer components that identify an object such as an algorithm. The ASN.1 notation for the OBJECT IDENTIFIER type is *OBJECT IDENTIFIER*.The ASN.1 notation for values of the OBJECT IDENTIFIER type is

{ [identifier] component$_1$ … component$_n$ }

component i = identifier_i | identifier_i (value_i) | value_i

where identifier, identifier_1, …, identifier_n are identifiers, and value_1, …, value_n are optional integer values.

The contents octets are the concatenation of n−1 octet strings, where n is the number of components in the complete object identifier. The first subidentifier is 40value1 + value2 and the ith subidentifier, $2 \le i \le n-1$, is value_{i+1}.

For example for the object identifier { 1 2 840 113549 } the DER encoding is computed as follows.42 = 40 * 1 + 2, encoding of 42 is 2A, encoding of 840 is 86 48, encoding of 113549 is 86 F7 0D and the encoding of 1 is 01. The encoding result is 06 07 2A 86 48 86 F7 0D.

OCTET STRING

The OCTET STRING type denotes an arbitrary string of octets. The ASN.1 notation for the OCTET STRING type is *OCTET STRING*.

For example, the DER encoding of the OCTET STRING value 01 23 45 67 89 AB CD EF is 04 08 01 23 45 67 89 AB CD EF.

PrintableString

The PrintableString type denotes an arbitrary string of printable characters. The ASN.1 notation for the PrintableString type is *PrintableString*.

For example, the DER encoding of the PrintableString value "Test User 1" is 13 0B 54 65 73 74 20 55 73 65 72 20 31.

SEQUENCE

The SEQUENCE type denotes an ordered collection of one or more types. The ASN.1 notation for the SEQUENCE type is

SEQUENCE {

[identifier$_1$] Type$_1$ [{OPTIONAL | DEFAULT value1}],

…,

[identifier$_n$] Type$_n$ [{OPTIONAL | DEFAULT value$_n$}]}

where identifier$_1$, …, identifier$_n$ are optional, distinct identifiers for the components, Type$_1$, …, Type$_n$ are the types of the components, and value$_1$, …, value$_n$ are optional default values for the components.

The contents octets are the concatenation of the DER encodings of the values of the components of the sequence,in order of definition.

SET

The SET type denotes an unordered collection of one or more types. The ASN.1 notation for the SET type is

SET {

[identifier$_1$] Type$_1$ [{OPTIONAL | DEFAULT value1}],

…,

[identifier$_n$] Type$_n$ [{OPTIONAL | DEFAULT $_{valuen}$}]}

where identifier$_1$, …, identifier$_n$ are optional, distinct identifiers for the components, Type$_1$, …, Type$_n$ are the types of the components, and value$_1$, …, value$_n$ are optional default values for the components.

The contents octets are the concatenation of the BER encodings of the values of the components of the set,

UTCTime

The UTCTime type denotes a Greenwich Mean Time (GMT) value. The ASN.1 notation for the UTCTime type is *UTCTime*.

Contents octets give the characters in the string, encoded in ASCII.

For example, the ASCII string representation of 07 June 2007 Thursday 09:19:25 GMT+2 is 070607061925Z, and the DER encoding is 17 0D 30 37 30 36 30 37 30 36 31 39 32 35 5A.

APPENDIX B

SAMPLE X.509 CERTIFICATE ASN.1 ENCODING

In the sample X.509 Certificate encoding the values before : states the position in the data, the value of the tag and the length of the element.

```
 0 30  948    : SEQUENCE {
 4 30  668    :   SEQUENCE {
 8 A0    3    :     [0] {
10 02    1    :       INTEGER 2
              :       }
13 02    2    :     INTEGER 321
17 30   13    :     SEQUENCE {
19 06    9    :       OBJECT IDENTIFIER
              :         sha1withRSAEncryption (1 2 840 113549 1 1 5)
30 05    0    :       NULL
              :       }
32 30   35    :     SEQUENCE {
34 31   11    :       SET {
36 30    9    :         SEQUENCE {
38 06    3    :           OBJECT IDENTIFIER countryName (2 5 4 6)
43 13    2    :           PrintableString 'tr'
              :           }
              :         }
 47 31   20  :     SET {
 49 30   18  :       SEQUENCE {
 51 06    3  :         OBJECT IDENTIFIER organizationName (2 5 4 10)
 56 13   11  :         PrintableString 'ESYA'
             :          }
             :        }
             :      }
 69 30   30  :   SEQUENCE {
 71 17   13  :     UTCTime '070417075835Z'
 86 17   13  :     UTCTime '100111085835Z'
             :     }
101 30   24  :   SEQUENCE {
103 31   22  :     SET {
105 30   20  :       SEQUENCE {
107 06    3  :         OBJECT IDENTIFIER commonName (2 5 4 3)
112 0C   13  :         UTF8String 'Murat Kubilay'
             :          }
             :        }
             :      }
127 30  159  :   SEQUENCE {
130 30   13  :     SEQUENCE {
132 06    9  :       OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
143 05    0  :       NULL
             :       }
145 03  141  :     BIT STRING 0 unused bits
             :       30 81 89 02 81 81 00 9B 76 16 AD 44 12 A2 AB FF
             :       5B 8A DC A7 42 88 80 69 1E A8 17 11 BD B2 63 00
             :       89 A5 4B F4 16 4A E2 D7 E6 C9 E7 CE 97 2B E1 B7
             :       DB EC 5D A5 25 EB CE CD DA 85 CD 2D 77 7B 30 B9
             :       A6 12 8C 9C 03 C0 E5 E3 CD A8 9A C1 54 BB 1E 9A
             :       DD 69 FF 9E 52 03 C2 55 8C D9 E1 04 C8 59 1F B0
             :       33 9B C1 8F 34 E0 D6 CF 60 7C DF 33 F7 0D 5E 65
```

```
              :      A7 56 36 D0 48 84 2E 93 EE 1D 59 8E 47 2A CF 8C
              :          [ Another 12 bytes skipped ]
              :      }
289 A3  383   :   [3] {
293 30  379   :     SEQUENCE {
297 30   31   :       SEQUENCE {
299 06    3   :         OBJECT IDENTIFIER authorityKeyIdentifier (2 5 29 35)
304 04   24   :         OCTET STRING
              :           30 16 80 14 A7 16 EA B4 4F BB D5 DE 4B C5 10 66
              :           C1 F9 D1 37 B8 E2 CB 7A
              :         }
330 30   29   :       SEQUENCE {
332 06    3   :         OBJECT IDENTIFIER subjectKeyIdentifier (2 5 29 14)
337 04   22   :         OCTET STRING
              :           04 14 5F DD 34 C6 59 AA C4 11 C9 DA 09 09 D2 78
              :           D7 3C 3F BB 61 44
              :         }
361 30   14   :       SEQUENCE {
363 06    3   :         OBJECT IDENTIFIER keyUsage (2 5 29 15)
368 01    1   :         BOOLEAN TRUE
371 04    4   :         OCTET STRING
              :           03 02 05 20
              :         }
377 30    9   :       SEQUENCE {
379 06    3   :         OBJECT IDENTIFIER basicConstraints (2 5 29 19)
384 04    2   :         OCTET STRING
              :           30 00
              :         }
388 30  110   :       SEQUENCE {
390 06    3   :         OBJECT IDENTIFIER cRLDistributionPoints (2 5 29 31)
395 04  103   :         OCTET STRING
              :           30 65 30 28 A0 26 A0 24 86 22 68 74 74 70 3A 2F
              :           2F 32 30 2E 31 2E 35 2E 31 30 35 3A 38 30 38 30
              :           2F 45 53 59 41 53 49 4C 2E 63 72 6C 30 39 A0 37
              :           A0 35 86 33 6C 64 61 70 3A 2F 2F 32 30 2E 31 2E
              :           35 2E 31 30 35 3A 33 38 39 2F 43 3D 54 52 2C 4F
              :           3D 45 53 59 41 41 4B 54 41 52 49 4D 2C 6F 75 3D
              :           45 53 59 41 53 49 4C
              :          }
500 30  113   :        SEQUENCE {
502 06    8   :          OBJECT IDENTIFIER
              :           authorityInfoAccess (1 3 6 1 5 5 7 1 1)
512 04  101   :          OCTET STRING
              :           30 63 30 2B 06 08 2B 06 01 05 05 07 30 02 86 1F
              :           68 74 74 70 3A 2F 2F 32 30 2E 31 2E 35 2E 31 30
              :           35 3A 38 30 38 30 2F 45 53 59 41 2E 63 65 72 30
              :           34 06 08 2B 06 01 05 05 07 30 02 86 28 6C 64 61
              :           70 3A 2F 2F 32 30 2E 31 2E 35 2E 31 30 35 3A 33
              :           38 39 2F 43 3D 54 52 2C 4F 3D 45 53 59 41 41 4B
              :           54 41 52 49 4D
              :          }
615 30   59   :        SEQUENCE {
617 06    3   :          OBJECT IDENTIFIER subjectAltName (2 5 29 17)
622 04   52   :          OCTET STRING
              :           30 32 81 0F 6D 75 72 61 74 6B 40 75 67 2E 75 65
              :           6B 61 65 A0 1F 06 0A 2B 06 01 04 01 82 37 14 02
              :           03 A0 11 0C 0F 6D 75 72 61 74 6B 40 75 67 2E 75
              :           65 6B 61 65
              :          }
              :        }
              :      }
              :    }
676 30   13   :  SEQUENCE {
678 06    9   :    OBJECT IDENTIFIER
```

```
                :      sha1withRSAEncryption (1 2 840 113549 1 1 5)
689 05          :    NULL
                :    }
691 03  257     :  BIT STRING 0 unused bits
                :    8E 4E 73 60 B7 83 61 25 92 42 52 41 CD FD 4C 02
                :    26 FE 0E 4B 1B C9 2E E7 13 1A 1F 3F 41 3A 35 15
                :    65 41 74 3D 45 29 18 7C 6B A9 67 53 5C C5 3C A3
                :    F0 94 FC 5C DB 8B B2 F9 61 C3 F4 C0 FB E7 E9 74
                :    9E C5 1B 75 70 4B 5C 40 16 95 FE 59 BE 6B E6 B1
                :    80 75 FE 53 3F 60 43 D8 11 8E 3A 8F 45 68 01 94
                :    96 A3 91 E4 86 B8 B9 01 D8 E4 D3 C7 B6 04 DB B7
                :    7E CA D1 4E ED 3E 8D 16 49 6C 32 AE AB 8D 64 0C
                :            [ Another 128 bytes skipped ]
                :  }
```

APPENDIX C

SAMPLE LIGHTWEIGHT CMS ASN.1 ENCODING

In the sample lightweight CMS encoding the values before : states the position in the data, the value of the tag and the length of the element.

```
 0 30  399    : SEQUENCE {
 4 06    9    :   OBJECT IDENTIFIER envelopedData (1 2 840 113549 1 7 3)
15 A0  384    :   [0] {
19 30  380    :     SEQUENCE {
23 02    1    :       INTEGER 0
26 31  187    :       SET {
29 30  184    :         SEQUENCE {
32 02    1    :           INTEGER 0
35 30   35    :           SEQUENCE {
37 04   20:                OCTET STRING
              :               DA 39 A3 EE 5E 6B 4B 0D 32 55 BF EF 95 60 18 90
              :               AF D8 07 09
59 30    7    :             SEQUENCE {
61 06    5    :               OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
              :               }
68 02    2    :             INTEGER 321
              :             }
72 30   11    :           SEQUENCE {
 74 06    9   :           OBJECT IDENTIFIER
              :             sha1withRSAEncryption (1 2 840 113549 1 1 5)
              :           }
 85 04  128   :         OCTET STRING
              :           52 78 BE F2 57 8D 7C 30 06 6F 2D 59 81 9C 4C 38
              :           7A D0 BC C4 5E 16 6D 93 57 4B 50 EE C3 77 ED 4A
              :           1E C6 B7 2B BA F6 AC 47 91 C4 68 F3 4B 37 FA C1
              :           87 8F 30 E3 3C 25 77 64 DC EB 77 B6 AB EF BA 35
              :           A6 82 41 8B B3 A2 1D 74 39 8A 0D 4E 1E 4F A9 F5
              :           B9 3C BA 52 02 80 30 49 53 63 57 A6 01 23 24 0B
              :           A8 E9 2F A2 60 CC 61 57 D9 A0 7E DE 7F 3A 57 88
              :           75 AB 01 A3 B5 0A 05 CA 2F 2A 4A 5C EC 05 F2 29
              :         }
              :       }
216 30  184   :     SEQUENCE {
219 06    6   :       OBJECT IDENTIFIER encrypted content '1 3 4 5 6 7 8'
227 30   11   :       SEQUENCE {
229 06    9   :         OBJECT IDENTIFIER aes128-CBC (2 16 840 1 101 3 4 1 2)
              :         }
240 80  160   :       [0]
              :         C1 9C 19 A1 34 E0 28 E2 E0 65 62 E1 B9 CE 3E 07
              :         31 E5 EF 49 E5 48 90 DF 07 F3 6B 2E 5C B0 CD B7
              :         B8 30 4E D3 5F B5 C8 BE CE 62 5F 96 2F 8C 15 E7
              :         66 FD 50 45 6A 13 0F E2 EB EC 3B 1C 1F E3 65 BC
              :         9A 4E 6E 5F 2E 92 A0 CE 99 19 E9 2D 4E ED 9E 91
              :         CD F4 50 D7 7D 83 29 C2 7E 7B DE A1 9E B7 FB 86
              :         94 E2 70 29 74 4D 1F A2 53 94 CA D0 AE 0F 24 C4
              :         D8 E4 32 8A D5 2C 4A 72 2C E6 A8 DD 49 60 6E 65
              :                 [ Another 32 bytes skipped ]
              :       }
              :     }
              :   }
```