Department of Computer Science & Engineering
The University of Texas at Arlington

Detailed Design Specification
CSE 4317: Senior Design II
Spring 2024

ESMS
Encrypted SMS

Jacob Holz
Landon Moon
Parker Steach
Gilbert Lavin
Nam Huynh

Revision History

Revision	Date	Author(s)	Description
0.1	1.31.2024	LM	Document creation
1.0	2.11.2024	ALL	First draft completion

CONTENTS

List of Figures

List of Tables

1 Introduction

The design of ESMS revolves around the idea of utilizing an older protocol, SMS, and wrapping it with a secure encryption layer. Additionally, as a messaging app, ESMS takes key design and requirement details from most modern message apps. Full details about the Architectural Design and System Requirements can be found in corresponding documents in the ADS and SRS respectively.

The system requirements include basic attributes of the app along with ESMS specific implementation requirements. Some basic app attributes include requiring a contacts, messaging, and settings view along with a 'conventional' look to each to ease user interaction. ESMS specific requirements include the need to use only the SMS protocol for network communication, and allowing for multiple encoding algorithms, some of which are cryptographically secure. The Architectural Design Document separated the app into three layers: OS Interaction, Cryptography, and Views. These layers include subsystems which interact to fulfil the system requirements.

2 System Overview

The System Architecture has three layers: OS Interaction, Cryptography, and Views. The OS Interaction Layer is intended to abstract most of the implementation details of interacting with the phone's APIs. Due to the requirements of the ESMS app, the subsystems include SMS protocol interaction, contacts list reading, and data storage for saved parameters. The Cryptography Layer is intended to isolate cryptography from other systems for security. The outward-facing subsystem of the layer is the Cryptography Engine Generator which provides an encrypt/decrypt service for any of the predefined or custom cryptography engines. Lastly, the Views Layer outlines and specifies what other subsystems the three views interact with. The three views are the Contacts View, Conversation View, and Parameter (settings) View. These views generate user input and use the services of the other layers in order to get the assets required to provide an interactive visual display to the user.

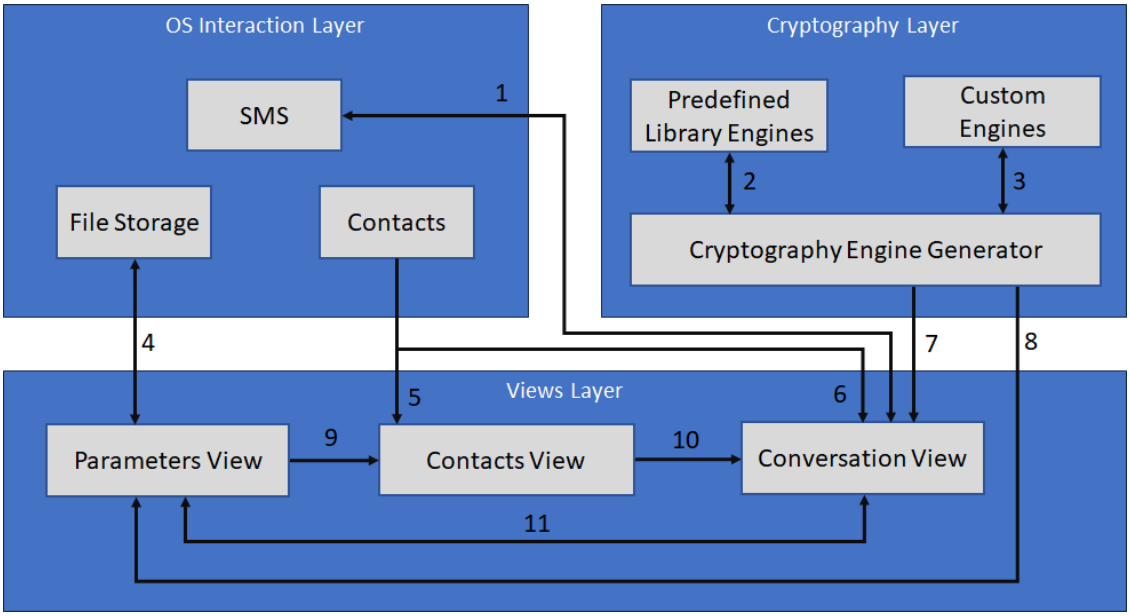

Figure 1: System architecture diagram

3 OS Interaction Layer Subsystems

3.1 Layer Hardware

As a phone app, ESMS indirectly interacts with the hardware contained on the phone. The details of the hardware vary between phones and are abstracted away by the operating system. Every modern phone includes a touch-screen input device, telephony sender/receiver, CPU, RAM, and data storage along with multiple other systems that are unused by ESMS. As ESMS does not directly interact with these systems, but instead does so through the OS, more details are included in this section.

3.2 Layer Operating System

The ESMS app runs on Android phones and needs to interact with different services through the Android operating system. The way ESMS interacts with the OS is through the Android API.

3.3 Layer Software Dependencies

The only notable dependencies used in this layer are the Android API calls and the conveniences granted by the Kotlin language.

3.4 SMS Subsystem

The SMS subsystem in the OS Interaction layer encapsulates any SMS interaction required by ESMS. This subsystem further abstracts away the implementation details of sending, receiving, and reading previous SMS messages so other subsystems can utilize SMS as an easy to use service.

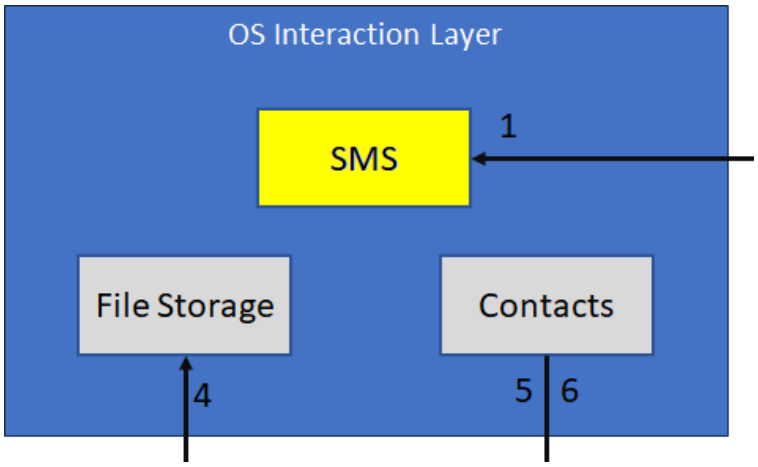

Figure 2: SMS Subsystem diagram

3.4.1 Subsystem Software Dependencies

The Android API is required to interact with services provided by the phone.

3.4.2 Subsystem Programming Languages

ESMS is primarily coded in Kotlin due to its ease of use, phone support, and because of our UI framework of choice. More details about why can be found in the views layer.

3.4.3 Subsystem Data Structures

SMS message information is requested from the phone in an SQL-like format and received as a list result. This request is executed through an interface in the Android API.

3.5 Contacts Subsystem

The contacts subsystem in the OS Interaction layer encapsulates any contact interaction that will be required by ESMS. By abstracting the OS interaction into a subsystem, other subsystems can access contacts in the phone's contact information in an easy to use and more context specific manner.

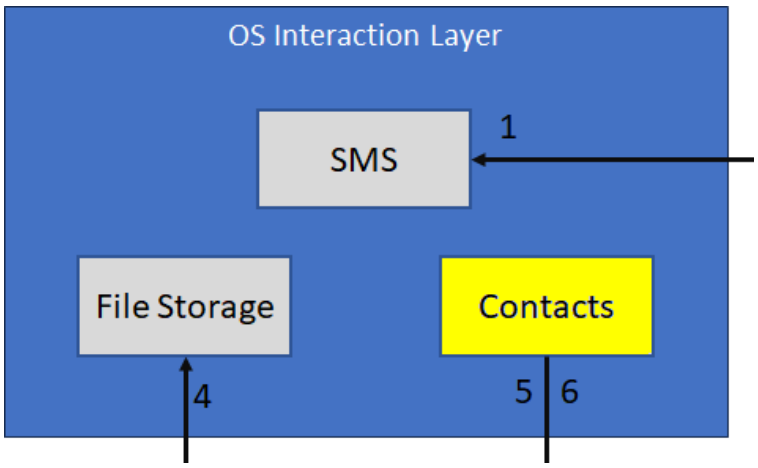

Figure 3: Contacts Subsystem diagram

3.5.1 Subsystem Software Dependencies

The Android API is required to interact with services provided by the phone.

3.5.2 Subsystem Programming Languages

ESMS is primarily coded in Kotlin due to its ease of use, phone support, and because of our UI framework of choice. More details about why can be found in the views layer.

3.5.3 Subsystem Data Structures

Phone contact information is requested from the phone in an SQL-like format and received as a list result. This request is executed through an interface in the Android API.

3.6 File Storage Subsystem

ESMS as well as many phone apps need to have some persistent data between executions. This can include multiple kinds of setting such as the ESMS Theme, default encryption method, or conversation-specific parameters. These settings are saved by first stringifying and encrypting the internal map of settings elsewhere and utilizing the Android API to save the encrypted string between sessions. It can then be retrieved and interpreted on startup.

3.6.1 Subsystem Software Dependencies

The Libary ESMS uses is Jetpack Compose which is a Android recommended toolkit that helps build a native UI.

3.6.2 Subsystem Programming Languages

Kotlin is the main languague used to create ESMS. Kotlin is a general purpose programming language that is used by our framework (Jetpack Compose) and is designed to be easy to use for mobile development.

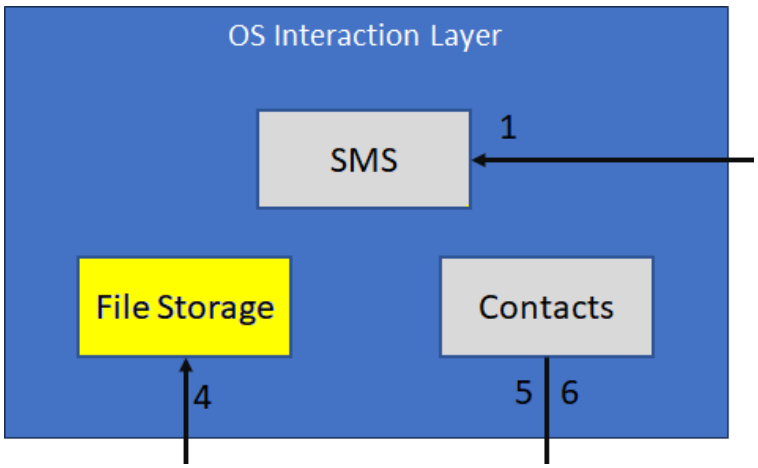

Figure 4: File Storage Subsystem diagram

3.6.3 Subsystem Data Structures

The File Storage subsystem primarily interacts with the parameters view. Due to this the data-structure that this subsystem receives is a Map<String, Map<String, String». This data is structured such that each entry in the top level map represents the values of one stored variable, thus any non-map variable is turned into a map of size one containing the data of that variable in the [""] position of the map. All other stored values are maps, and are stored as such. The key of those top level entries is a constant that identifies the variable being saved.

3.6.4 Subsystem Data Processing

Due to a limitation in saving information to a file in the Android API, any saved information has to be serialized. This means that the data-structure mention above needs to be converted into a single string. This is done by using the Map.entries.joinToString() function twice to stringify both layers of the Map<String, Map<String, String» data-structure. This stringification returns a string which can be easily encrypted elsewhere then saved. When loading, this process is done in reverse with decryption followed by deserialization.

4 Cryptography Layer Subsystems

4.1 Layer Hardware

As a phone app, ESMS indirectly interacts with the hardware contained on the phone. The details of the hardware vary between phones and are abstracted away by the operating system. Every modern phone includes a touch-screen input device, telephony sender/receiver, CPU, RAM, and data storage along with multiple other systems that are unused by ESMS. As ESMS does not directly interact with these systems, but instead does so through the OS, more details are included in that section.

4.2 Layer Operating System

The ESMS app runs on Android phones and needs to interact with different services through the Android operating System. The way ESMS interacts with the OS is through the Android API

4.3 Layer Software Dependencies

The Cryptography Layer uses javax packages to access predefined cryptographic engines.

4.4 Cryptography Engine Generator Subsystem

The Cryptography Engine Generator Subsystem keeps track of all cryptographic engines available for the users and provides new instances upon request.

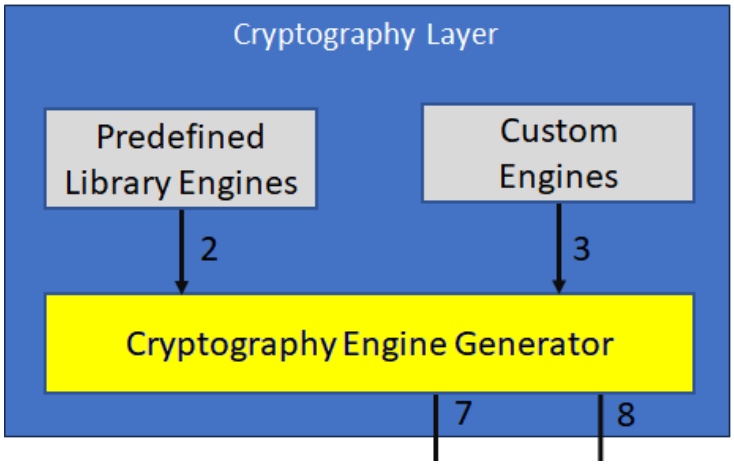

Figure 5: Cryptography Engine Generator Subsystem diagram

4.4.1 Subsystem Software Dependencies

Since ESMS is an android application will be utilizing the Jetpack Compose framework for all subsystems.

4.4.2 Subsystem Programming Languages

ESMS in programmed almost entirely in Kotlin which ultimately compiles down to Java code. Due to this this subsystem is able to be developed in Kotlin.

4.4.3 Subsystem Data Structures

The Cryptography Engine Generator utilizes a map for identifying different engines utilized by the Generator.

4.5 Predefined Library Engines Subsystem

Because ESMS is trying to implement multiple encryption methods, the custom ESMS encryption methods are bound to not be perfect. Custom implementations of security algorithms are difficult to make secure because they have not been tested thoroughly. Because of this ESMS will utilize some of the encryption methods already provided by the javax.crypto package to make ESMS more secure.

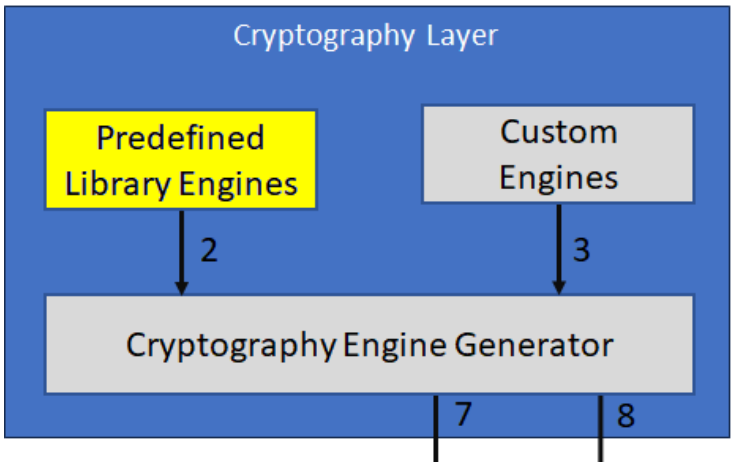

Figure 6: Predefined Library Engines Subsystem diagram

4.5.1 Subsystem Software Dependencies

The Predefined Library Engines Subsystem requires access to the javax.crypto package in the java standard library. All implementations of cryptographyEngine within this sub-system are direcly passed to functions within this package and received as a string.

4.5.2 Subsystem Programming Languages

ESMS is programmed almost entirely in Kotlin which ultimately compiles down to Java code. Due to this this subsystem is able to be developed in Kotlin and still utilize the javax.crypto package even though it is written in Java.

4.5.3 Subsystem Data Structures

The custom engines will utilize a 'Cryptography Engine' interface created by ESMS. This creates a standard for creating new types of encryption methods to keep consistency in between encryption types. Each engine will have a function for 'encrypting' and 'decrypting'.

4.6 Custom Engines Subsystem

Custom Engines will be encryption methods created and implemented by the ESMS team. Most custom engines provided will not be very secure due to the challenge of making complex, secure encryption methods. Instead, they will be more fun and simple encryption methods, such as the ceasar cipher. This will be an opportunity for the team to learn and gain experience creating encryption methods.

4.6.1 Subsystem Software Dependencies

Since ESMS is an android application, it will be utilizing the Jetpack Compose framework for all subsystems.

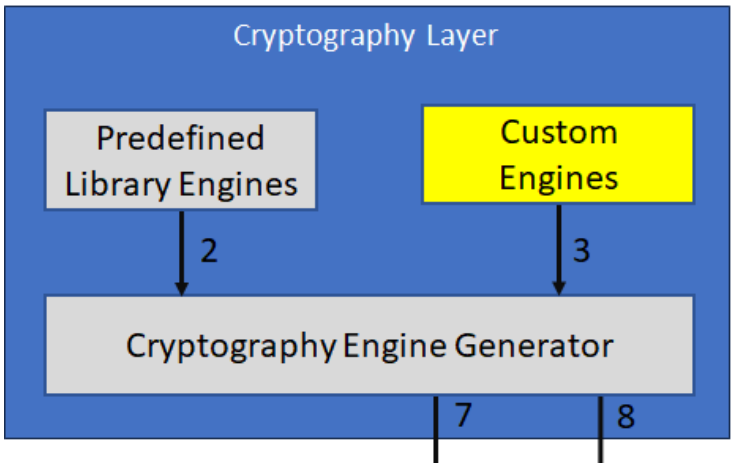

Figure 7: Custom Engines Subsystem diagram

4.6.2 Subsystem Programming Languages

ESMS is programmed almost entirely in Kotlin which ultimately compiles down to Java code. Due to this this subsystem is able to be developed in Kotlin.

4.6.3 Subsystem Data Structures

The custom engines will utilize a 'Cryptography Engine' interface created by ESMS. This creates a standard for creating new types of encryption methods to keep consistency in between encryption types. Each engine will have a function for 'encrypting' and 'decrypting' with an initializing parameter string that characterizes the key used to encrypt and decrypt messages.

5 Views Layer Subsystems

5.1 Layer Hardware

Since ESMS can run on Android phones, the hardware differs between devices. Our app will mostly be utilizing the touch-screen, telephony sender/receiver, CPU, RAM, and data storage. ESMS has to use the OS to directly interact with these systems. The views layer uses the OS layer to interact with these systems. More information of OS interaction is available in that section.

5.2 Layer Operating System

This layer depends on the OS layer in order to interact with the underlying phone. As a result, the Android operating system is required but not directly utilized by this layer.

5.3 Layer Software Dependencies

The primary framework for the UI and adjacent systems is Jetpack Compose. Jetpack Compose provides an easy to use library of different components that can be utilized in many scenarios. Jetpack Compose uses Kotlin as a programming language due to its better syntax for UI development compared to Java.

5.4 Contacts View Subsystem

The contacts view will show you contacts that can be messaged within the application. This view also requires interaction with the encryption layer in order to get the contact list and the associated contact images.

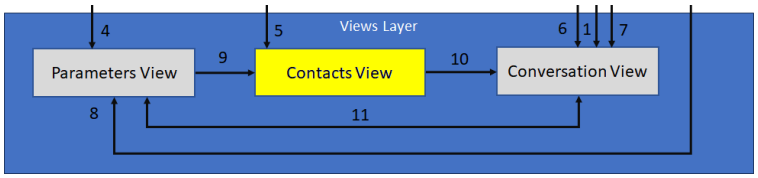

Figure 8: Contacts View Subsystem diagram

5.4.1 Subsystem Software Dependencies

The display of contact images is handled by the Coil library as they are extracted from the Contacts subsystem and displayed asynchronously when the image is loaded. Jetpack Compose is the primary software dependency as it is the framework for the applications visual design and state management.

5.4.2 Subsystem Programming Languages

The primary programming language for the contacts view is Kotlin.

5.4.3 Subsystem Data Structures

Through the interaction with the Contacts Subsystem in the OS Interaction layer this subsystem retrieves a list of contact. This is done by defining some searching and sorting criteria to the OS Interaction layer, and receives an appropriate list of contacts to fill the contacts view with. The contact objects have the name, number, and internal image URL for a given contact on the phone.

5.4.4 Subsystem Data Processing

The contact data us used to construct the list of contact buttons that is displayed to the user. The timestamp of the last message loaded is also stored so that the list can be ordered accordingly.

5.5 Conversation View Subsystem

The conversation view will show you messages you have sent and received. This view also requires interaction with the encryption layer in order to encrypt and decrypt messages when sending and receiving as well as interacting with the SMS subsystem for sending, receiving, and viewing previous SMS messages.

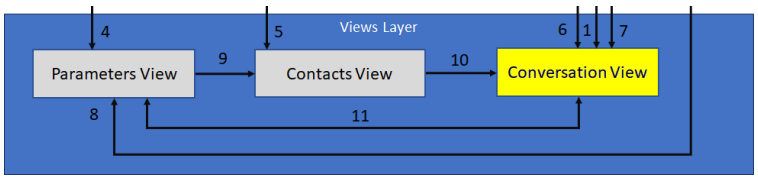

Figure 9: Conversation View Subsystem diagram

5.5.1 Subsystem Software Dependencies

The display of contact images is handled by the Coil library as they are extracted from the Contacts subsystem and displayed asynchronously when the image is loaded. Jetpack Compose is the primary software dependency as it is the framework for the applications visual design and state management.

5.5.2 Subsystem Programming Languages

The primary programming language for the Conversation view is Kotlin.

5.5.3 Subsystem Data Processing

The conversation view will first receive data in the form of an encrypted message from the OS interaction layer. This message comes from the phones SMS service. After the user sees the data they will be able to click on it and send it to Cryptography Engine acquired from the Cryptography Layer. The Cryptography Engine Generator will receive data such as keys from the parameter view to create the Cryptography Engine used to decrypt the message. Once the message has been decrypted it will be sent back to the conversation view and showed to the user. The same is done in reverse when sending an encrypted message.

5.6 Parameters View Subsystem

The parameters view is the interface for the user to change both global and conversation-specific parameters. These parameters are then used to define certain parts of how the ESMS app works. These parameters can define what type of encryption the conversation view will utilize from the encryption layer, or the color theme of ESMS.

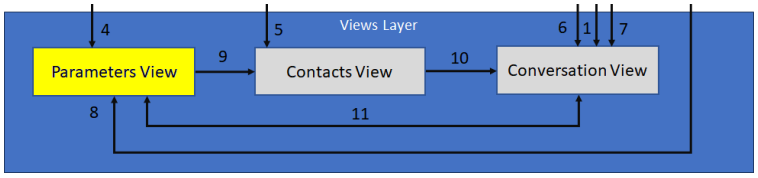

Figure 10: Parameters View Subsystem diagram

5.6.1 Subsystem Operating System

This subsystem uses the File Storage subsystem to persist parameters across sessions in an encrypted manner.

5.6.2 Subsystem Programming Languages

Due to the use of Jetpack Compose, the primary programming language for the contacts view is Kotlin.

5.6.3 Subsystem Data Structures

The global parameters are saved within memory as a mappings from string keys to string values (Map<String, String>) or as single values. The conversation-specific parameter are also saved the same way using the conversation's phone number as the key. During runtime, the conversation-specific parameters mask over the global parameters so these changes effect specific conversations.

6 Appendix A

References