Methods for implementing improved privacy and security in iOS messaging applications

Sarah Michaela Sauseng

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2018

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, June 25, 2018

Sarah Michaela Sauseng

Contents

Kurzfassung

Heutzutage ist die Kommunikation durch mobile Geräte zu einem wesentlichen Bestandteil unseres Lebens geworden. Insbesondere die Verwendung von Messaging-Apps für unsere täglichen Unterhaltungen hilft uns leicht mit anderen in Kontakt zu bleiben. Dabei darf jedoch nicht auf die Sicherheit und Privatsphäre der Nutzer und ihrer Daten vergessen werden, die gerade beim Austausch von digitalen Nachrichten eine wichtige Rolle spielen. Dies stellt auch ernstzunehmende Probleme im Bezug darauf dar, wie persönlich und intim die Nachrichten sind, die gesendet werden. Niemand möchte es riskieren, dass Fremde auf den Inhalt einer Nachricht zugreifen können oder dass die eigenen gesendeten und persönlichen Daten unkontrolliert verbreitet werden. Dies ist speziell dann möglich, wenn diese Inhalte auf dem Client oder Server persistiert werden und jederzeit abrufbar sind. Der Schwerpunkt dieser Arbeit liegt daher in der Konzeption und Umsetzung einer iOS Anwendung, die anonyme Kommunikation ermöglicht und für Transienz der verwendeten Daten sorgt, wodurch weder am Client noch am Server Informationen erhalten bleiben. Dazu werden insbesondere die Methoden der Verschlüsselung von Daten, der Beibehaltung von Anonymität durch Verwendung ausschließlich generierter Benutzeridentifikation, und der Transienz der Nachrichten durch automatisiertes Löschen auf Client und Server eingesetzt. Gerade clientseitig stellt dies eine zusätzliche Herausforderung dar, da auch die Möglichkeit in Betracht gezogen werden muss, dass die Nachricht mit Mitteln ausserhalb der Applikation (Screenshot) persistiert werden kann. Durch Umsetzung dieser Massnahmen wird gewährleistet, dass mobile Kommunikation unter maximaler Beibehaltung von Privatsphäre und Sicherheit durchgeführt werden kann und die Rückverfolgbarkeit von Benutzern und deren Daten auf ein Minimum reduziert wird.

Abstract

Nowadays the communication through mobile devices has become a substantial part of our everyday life. In particular, the use of messaging apps for our daily conversations helps us easily stay in touch with others. Nevertheless, one should not forget the security and privacy of users and their data, which play an essential role in the exchange of digital data. This also poses severe problems concerning how personal and intimate the messages being sent are. Nobody wants to risk strangers being able to access the content of a message, or that personal data gets distributed uncontrolled. This is particularly possible if these contents are persisted on the client or server and are retrievable at any time. Therefore, the focus of this work is the conception and implementation of an iOS application that enables anonymous communication and ensures the transience of the data used, whereby information is not preserved either at the client or the server. In particular, the methods of data encryption, the retention of anonymity by using exclusively generated user identification, and the transience of the messages by an automated deletion on client and server are used. Especially on the client side, this poses an additional challenge, since the possibility must also be considered that the message can be persisted with means outside the application (screenshot). The implementation of these measures will ensure that mobile communications can be conducted with maximum privacy and security while minimizing the traceability of users and their data.

Chapter 1

Introduction

1.1 Motivation

Today the importance of privacy and data security in digital communication is becoming an increasingly significant topic. Nowadays almost every company contributing services that operate with user-related data has to implement methods to preserve and protect the customer's data. Especially with mobile messaging applications, it is of high value concerning how personal the type of messages sent are and how precisely someone can get traced back. In general, one can assume that personal privacy is essential for everyone, but when it comes to using messaging applications for exchanging our private thoughts, feelings, images and so on we like to forget about the impact on it. Nearly all messaging platforms store chat data to provide specific functionalities or for statistical analyses. Therefore it can be considered that the ownership of a message gets lost somehow when a person sends it through mobile communication.

Another remarkable aspect regarding the privacy of users is the traceability of data. Even if most of the applications already provide end-to-end encryption techniques, it is always possible to find out who is behind a message or to persist the data sent. Every messaging application requests personal information at the registration. Thus users are always verifiable by their mobile number, email address or other additional information. In addition to this, relations between users are possible to be traced back by their contact book data and by the sender-receiver data sent with a message.

Therefore new methods for mobile communication that solve these issues or even lead to the full anonymity of users combined with common, strong security aspects should be developed. These include the assurance of privacy by establishing full messaging functionality without using any personal user information and the guarantee that the content of a message does not get stored on any server or gets persisted in another possible way. Moreover, it is essential to ensure that relations between users are not traceable and that only the targeted receiver is able to uncover the data of a message. These methods should then get combined with all further security aspects necessary to support the protection of application functionalities.

1.2 Goal of the thesis

Concerning the above-described motivation, this thesis aims to find new methods for implementing higher privacy and security in mobile communication. Therefore the challenge will be figuring out which technologies and methods improve the privacy and security of a user and how they can be implemented to even guarantee anonymity for a person that uses a messaging application for private communication. A new approach for mobile conversation should be developed where data can be exchanged entirely private, but still in a secure way. Users themselves should be able to verify who is behind a message, even if the information that is stored and transferred is kept anonymous. Furthermore, it is essential that all options that would make it possible to persist the message data and would lead again to the traceability of a user, get eliminated as far as possible. Therefore several techniques that prevent the data persistence of messages have to be implemented. By combining all these aspects, it should be possible to find a new approach for a more secure and private mobile communication.

1.3 Overview

First, in chapter 2 the current state of the art and the critical concerns regarding privacy and security of today's digital communication and their relation to mobile environments are briefly described. Then chapter 3 refers to related work of already existing techniques and methods for implementing higher security and privacy in mobile messaging. Furthermore, the most popular messaging applications and their concepts to provide improved privacy protection get demonstrated. In chapter 4 the requirements, the conception of building a secure messaging application and the architecture get derived from the research results. Next, in chapter 5 a detailed description of the implementation and the technical design of the frontend and backend application is presented. After that, an evaluation of the developed methods is described in chapter 6 by comparing them with an already existing application from the predefined requirements. Finally, chapter 7 draws a conclusion of the outcome of the thesis and the implemented methods, and a brief perspective into further research is provided.

Chapter 2

Background

In this chapter, the current state of the art regarding privacy and security in mobile communication gets described briefly. Since the primary focus of this thesis lies on developing new secure patterns for messaging applications in iOS, all the parts that get explained are related to this platform. This part is about the main concerns in mobile security and their importance for ensuring confidentiality of users. In the following pages some of the critical concerns of today's digital communication and their relation to mobile environments get demonstrated.

2.1 Security challenges in mobile messaging

Before developing a new concept to establish secure and private messaging, it is necessary to be informed about the main challenges that appear in current mobile communication. There exist several problem areas that have to be considered when dealing with the security aspects of messaging applications. They can be divided into three major parts: the *trust establishment problem*, the *conversation security problem* and the *transport privacy problem* [7], and get individually disclosed hereafter.

2.1.1 Trust establishment problem

This problem is about the verification of users assuring that they are communicating with the parties they intend. That includes the guarantee of cryptographic long-term key distribution, divided into the parts of long-term key exchange and long-term key authentication. Long-term key exchange refers to the process where users send cryptographic key material to each other. Long-term key authentication is the mechanism allowing users to ensure that long-term keys get associated with the correct real-world entities.

2.1.2 Conversation security problem

The second problem area includes the terms of conversation security and is about ensuring the protection of exchanged messages during conversations. Therefore a conversation security protocol should be used to protect the security and privacy of exchanged

messages. That includes how messages are encrypted, the attached data and which cryptographic protocols are performed.

2.1.3 Transport privacy problem

The last problem affects the transport privacy that is responsible for hiding the communication metadata. The goal is to hide message metadata such as the sender, receiver, and conversation to which the message belongs. By using transport privacy layers and schemes, the exchange of the messages gets defined and may also get used for privacy-preserving contact discovery.

2.2 Verifiability of users

As already mentioned in section 2.1.1, the verifiability of users is an essential concern regarding trust establishment when exchanging mobile messages.

2.2.1 Access control

When sending messages between recipients, access control is a necessary part for ensuring that only users that are authorized can retrieve the information sent. It is essential to set access restrictions to protected data for all other stranger parties so that only the meant end users receive the information. In general, this concept is divided into three main parts: identification, authentication, and authorization.

Identification

Identification is the first step of the process for gaining access to specific data. The term refers to providing a subject that the entity claims to be. In most cases, this means some username, email or mobile number.

Authentication

The next step is about proving that the identity of the subject belongs to it. Usually, this is implemented by making use of a password, an authentication phrase, a pin or cryptographic keys.

Authorization

The last step is then about controlling access of the given identity. Only intended entities should be able to gather specific data.

2.2.2 Personal privacy

Another issue that is specifically related to this master thesis is the personal privacy of users. Therefore a different perspective about user verification in regards to the relation of providing personal registration/verification data and the permanent persistence of this data on a server. So it is not only about the importance of protecting access to specific data or the identity proof of entities, but it can also be considered as a concern

for personal privacy since the digital identification data is always persisted and leads to the real person behind it. A typical example would be using the mobile number or email address for user registration.

So another approach for user verification would be to stay anonymous as a person behind the sent messages as far as possible, but still enabling an access control process to guarantee that only authorized persons can use the messaging application for their purposes. Therefore no personal registration data has to be provided or refers to a specific person and also does not get persisted on some remote server or database. Nevertheless, all functionality of the application should be protected by giving only access to verified entities.

2.3 Traceability of data

Nearly every messaging application provides some chat history for users to read earlier messages at a later time. In addition to this, the data gets saved either on a server or locally on the smartphone of both communication participants, the sender, and receiver. In the majority of cases, this is done intentionally to provide a feature for the user allowing him to retrace a conversation at any later time. However, often it appears that people send very personal data and are not aware of the fact that it gets persisted somewhere and can not be deleted that quickly. Also, messages can be distributed very fast by forwarding it to other users and as a result get again stored on other additional smartphones. There exist several possibilities to retrace data in messaging applications, they can be divided into the following parts: the sender-receiver relation and the persistence of the data.

2.3.1 Sender-receiver relation information

When sending messages between two or more communication parties, there is always a relation between the sender and the receiver(s). Even if the chats are end-to-end encrypted and the content of the messages cannot be detected so quickly, there always exists some possibility to check which users communicate with each other. By using their destination-addresses (mobile number, email address or username) with the metadata of sent messages, it is possible to discover that these entities communicate with each other.

2.3.2 Persistence of data

Any data that is related to the communication process of messaging applications get on some point of data transmission persisted on either a server, a database or on smartphones. By doing a backup of the chat history or saving sent images/videos to the photo library, the content gets stored. Also by the distribution of messages, each recipient might persist them on their smartphones or likewise in their chat history. Another place where the chat data gets stored is on the server of the messaging applications, even if this is often only for a short time until the receiver reads the message.

In addition to this there exist potential other methods to save chat data. When exchanging messages with other users, the receiver(s) can also persist the content by

taking a screenshot, recording a video clip or sound recording with either, the receiver's smartphone, or by any other third party media device. So, there are several ways where messages get persisted and thus every content that gets sent via digital communication potentially leaves behind tracks.

2.4 Threats in mobile messaging

The last point worth noting is about possible threats in mobile messaging applications. These are significant issues that, when paid to little attention to them, could potentially lead to disclosing sensitive information. Especially in messaging applications there exist several entry points on the whole communication process that are vulnerable to possible attackers. Regarding the kind of implementation, the possible threats differentiate in how they have to be prevented. It can be distinguished between mobile web applications and native applications. Moreover latter can be divided into the primary platforms iOS, Android, and some additional ones. Since this work focuses on privacy and security for native iOS messaging applications, only topics that are related to this kind of implementation get mentioned.

2.4.1 Possible issues

When counteracting to possible threats that can occur in mobile messaging applications, as a first step, it is essential to define the issues that lead to them. One central issue found during the research is about jailbreaked devices [2, p. 14]. Furthermore, the topics of weak server implementation, insecure data handling, insufficient transport-layer protection, non-validated user input and broken cryptography are also of primary concern [3]. Each of them gets described in more detail hereafter.

Jailbreaked devices

Jailbreaking an iOS smartphone leads to higher control of the device, more access to system files and therefore enables higher functionality to personalize features. However, they also come with the disadvantage that jailbreaking smartphones are more accessible to exploit for hackers.

Weak server implementation

A necessary part, when using client-server communication, is that the server-side part gets also protected against potential attacks. Poor authorization and authentication would be one issue that would support attackers gaining access to sensitive data.

Insecure data handling

When sensitive data that is not encrypted, is stored on a device or when data is long-term cached, but not intended for it, can lead to the exposure of sensitive information and privacy violations. This issue also appears when generally failing to leverage best practices for a particular platform. In addition to this sensitive information, disclosure

can appear as result of hardcoding such data as login credentials, shared secret keys, access tokens or other business logic that an attacker might be able to access.

Insufficient transport-layer protection

For client-server communication, it is necessary to use secure transport-layer protocols. Otherwise, the data sent could get revealed by man-in-the-middle attacks.[1]

Non-validated user input

Every user-related input has to be validated and examined on possible code injections that would otherwise lead to the exfiltration of data or the escalation of privileges.

Broken cryptography

Particular attention should get paid to cryptography and the issues that can appear with it. When using custom instead of standard cryptographic algorithms or hardcoding cryptographic keys into the application code itself, it can result in a loss of data confidentiality or privilege escalation.

2.4.2 Possible adversaries

In addition to the definition of potential issues, it is also of importance to know about possible adversaries in secure messaging. When considering a variety of opponents, the following entities can be determined [7, p. 233].

Local adversaries

That would be attackers controlling local networks like open wireless access points.

Global adversaries

These include attackers controlling large segments of the internet, such as powerful nation-states or large internet service providers.

Service provider

For messaging systems that require a centralized infrastructure (e.g., public-key directories) the service operators can also be considered as potential adversaries.

2.5 App architectures

Another mentionable point for developing an iOS messaging application is the range of architectures that can be implemented. The most popular are the MVC, MVP and MVVM for structuring such applications. These architectures also exist for data-driven

[1] A man-in-the-middle attack occurs when an attacker secretly relays and alters the communication between two parties.

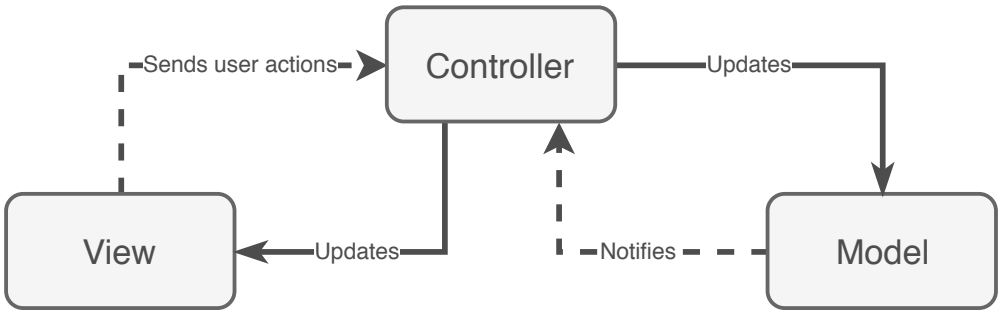


Figure 2.1: Structure of the MVC architecture.

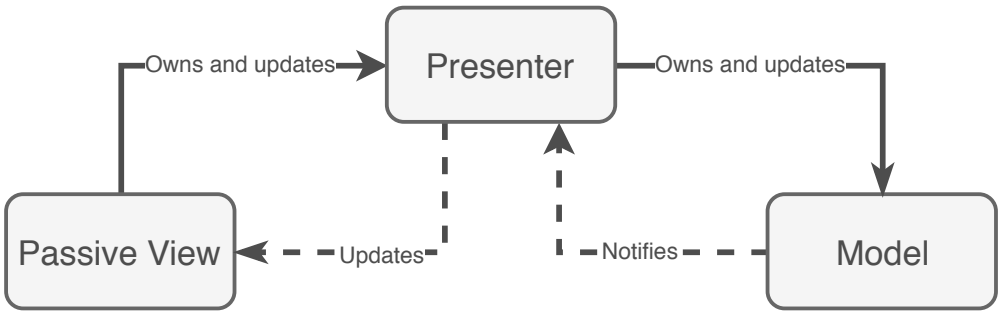


Figure 2.2: Structure of the MVP architecture.

web applications, but partially with minimal adjustments. All of them facilitate a separation of development of the graphical user interface from the development of the business logic. Hereafter a brief overview of them should be given. See for more detailed explanation about these patterns for iOS under [26].

2.5.1 MVC

The MVC (*model-view-controller*) architecture splits the responsibilities into the *model*, the *view* and the *controller*. In figure 2.1 the structure is shown. The *model* is responsible for the data or a data access layer which manipulates the data. The *view* has the function to display and send user actions to the *controller*. The *controller* is the mediator between the *model* and the *view*. It updates the *view* with changes from the *model* and vice versa. In iOS reality the *view* and *controller* are so tightly coupled through the lifecycle of the *view* that they are not really separated anymore.

2.5.2 MVP

In MVP (*model-view-presenter*) the responsibilities are divided into the *model*, the *view* and the *presenter* (see figure 2.2). Again the *model* is responsible for the data or data access layer. The *view* is also called "*passive view*", it owns the *presenter* and gets updated by it. The *presenter* owns the *model* and gets notified of changes in it, subsequently it updates the *view*.

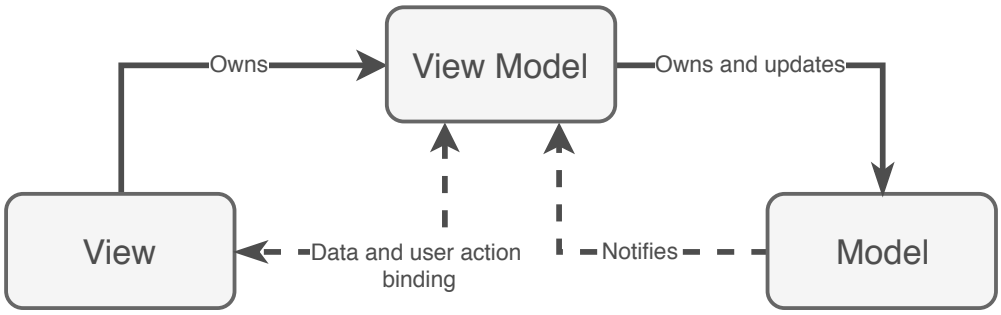


Figure 2.3: Structure of the MVVM architecture.

2.5.3 MVVM

The MVVM (*model-view-viewmodel*) includes the *model*, the *view* and the *viewmodel*. The structure is shown in figure 2.3. Here as well, the *model* contains the data and logic for the data access layer and the *view* is only displaying content. So there is no tight coupling between the *view* and the *model*. The huge difference to the MVP architecture is that it does data and event binding between the *view* and the *viewmodel*. Therefore the *view* is automatically updated when the *viewmodel* changes.

Chapter 3

Related Work

The research about privacy protection and data security in mobile communication has been growing a lot over the last years. Nearly every messaging application supplies several techniques to protect user data and has implemented secure communication processes. This section provides a short overview of the techniques, methods and related applications that were found during the research.

3.1 Techniques and standards

In this section, the techniques and standards that are relevant for developing a secure mobile messaging concept and for understanding the following chapters get described.

3.1.1 CIA triad

The CIA triad [25] is one of the fundamental concepts in information security and combines the three terms of confidentiality, integrity and availability, each standing for a principle that supports higher security for users in organizations when being well maintained.

Confidentiality

Confidentiality is about the principle of "least privilege". This principle states that access to information, assets, and so on, should be granted only on a need to know basis, so that information is only available to specific users. Concerning this, the principles of identification, authentication, and authorization support confidentiality and are related to it through their multiple access and privacy controls.

Integrity

The sender and receiver of a message may have a need for confidence that the message has not been altered during transmission, integrity ensures this. Information stored in underlying systems, databases, and so on, must be protected through access controls, and there should be an accepted procedure only to change the stored/transit data.

Availability

The availability concept is about making sure that the services of an organization are available. For any information system to serve its purpose, the information must be available when it is needed. By this the computing systems used to store and process the information, the security controls used to protect it, and the communication channels used to access it must be operating correctly.

3.1.2 Cryptography

Cryptography aims to store and transmit data in a form that only the authorized parties can interpret it. Nearly all messaging applications use some cryptography standard to prevent third parties from having plaintext access to messages. Only the users that are communicating with each other should be able to read the content. Cryptography includes techniques and standards that are essential for privacy and secure communication on the internet. It is divided into several field areas, the main cryptography types that are related to secure messaging and this thesis are the following: symmetric-key cryptography and public-key cryptography. See [19, 5] for a more detailed description.

Symmetric-key cryptography

In symmetric-key cryptography sender and receiver share both the same key for encryption and decryption of messages. An algorithm is used for key creation, encryption, and decryption. The sender and receiver must both keep a copy of the secret key in a secure place. It is a very effective and fast approach. The key only has to be generated, sent over to the receiver and the decryption can take place immediately. One drawback of symmetric-key cryptography is that if someone can get hold of the key, they can also encrypt and decrypt the data.

Public-key cryptography

In contrast to symmetric-key cryptography, an asymmetric key pair gets used for encryption and decryption. This pair is split up in a private key and a public key. Whereas the private key must be kept secret and is only used for decryption of the received data, the public key can be exchanged for possible communication entities. So anybody can encrypt plain text with the public key, but only the holder of the private key can decrypt the ciphertext[1] back to plain text. This method is more secure than symmetric-key cryptography, but it consumes more power and hardware processing time.

Encryption algorithms

These get used for generation, the modification, and transportation of the keys. An encryption algorithm is also used to turn plain text into ciphertext and vice versa. Below some of the main and most popular algorithms get explained briefly. In [19, 1] these algorithms get specified in more detail. Additional an overview can be found in table 3.1. However, before going through the algorithms, a small recap on a couple of terms related to these algorithms is given.

[1] The ciphertext is the result of encryption that was performed on plain text.

Parameters	**DES**	**AES**	**RSA**	**ECC**
Type	Symmetric	Symmetric	Asymmetric	Asymmetric
Key Length (Bits)	64 (56 usable)	128,192, 256	Key length depends on number of bits in the module	Smaller but effective key
Block Size (Bits)	64	18	Variable block size	Stream size is variable
Level of Security	Adequate security	Excellent security	Good level of security	Highly secure
Encryption Speed	Very slow	Faster	Average	Very Fast

Table 3.1: Comparison of encryption standards [1, p. 304].

Encryption key size: The number of bits in a key used by a cryptographic algorithm.
Block size: Fixed length groups of bits used for data encryption of block cipher.
Stream cipher: Plaintext digits get combined with a random cipher digit stream.
Block cipher: Encryption of a specific block size (known as rounds) and then padding the plain text so that it is a multiple of a block size.

DES: DES (Data encryption standard) is one of the most well-known symmetric-key data algorithms. It uses 56 bits key for encryption and decryption. It completes the 16 rounds of encryption on each 64 bits block of data. Now it is an old algorithm and is widely considered insecure. The algorithm was fortified with new updates called 2DES and 3DES, merely layering the cipher so that it would have to decrypt three times to each data block.

AES: The AES (Advanced encryption standard) was selected as a replacement to 3DES and is a symmetric block cipher. In more detail, it consists even of three block ciphers. Each cipher encrypts and decrypts data in blocks of 128 bits using cryptographic keys of 128 bits, 192 bits, and 256 bits. AES is considered as very safe and low power consuming algorithm.

RSA: The RSA (Rivest-Shamir-Adleman) is the most important, best known and widely used asymmetric-key algorithm. It uses an asymmetric block cipher and large integers like 1,024 bits in size. The main disadvantage is its algorithm speed, but it provides a reasonable level of security.

ECC: The ECC (Elliptic curve cryptography) provides an alternative mechanism for implementing asymmetric-key cryptography. It creates faster, smaller and more efficient

keys compared to other encryption algorithms. It uses an approach based on elliptic curves over finite fields and can offer the same level of cryptographic strength at much smaller key sizes. Some disadvantages would be that it increases the size of encrypted text and that it is dependent on very complex equations which lead to increase the complexity of encryption algorithm.

Other mentionable algorithms would be the Diffie-Hellman (asymmetric), PGP (asymmetric), IDEA (symmetric), MARS (symmetric), RC5 (symmetric), Blowfish (symmetric), Twofish (symmetric) and Threefish (symmetric). More information see at [4, 5].

Digital signature

Another essential part when exchanging encrypted text via digital communication is the signature of messages. A signature is used to verify the sender and validates if the expected entity indeed created the message. Another advantage of using a signature is the guarantee that the message was not altered in transit and that the sender cannot deny having it sent. Digital signatures can only be applied together with asymmetric-key algorithms for the reason that the private key is needed to generate the signature.

3.1.3 Authentication

To ensure that not everyone can send messages, spam a server with requests and that not everyone can get access to specific data, a proper authorization process has to be provided. There exist several options for this, some of them get shortly demonstrated afterward. Also mentionable is the digital signature scheme as explained before in section 3.1.2, because it also refers to authenticating the real sender behind a message.

Bearer token authentication

The bearer token authentication is an HTTP authentication scheme that involves security tokens called bearer tokens. The token is a secret string that gets generated by the server and has to be provided by the client in the authorization header each time sending a request.

Basic user-password authentication

The basic user-password authentication is a simple authentication scheme built into the HTTP protocol. When transmitting a request to the server, the client sends his username and his password with the authorization header.

Digest authentication

The digest authentication works similar to the basic authentication scheme, but it applies a hash function to the username and password before sending them.

3.1.4 Network transport security

Another point worth noting is the establishment of a secure client-server communication to prevent man-in-the-middle attacks. Therefore it should be ensured that a proper

network transport security gets implemented. Usually, the following protocols get used to providing higher security in network communication.

TLS

TLS (Transport layer security), former known as SSL, is a cryptographic protocol that provides communication security over a computer network. It is composed of two layers: the TLS record protocol and the TLS handshake protocol. The record protocol provides connection security, while the handshake protocol allows the server and client to authenticate each other and to negotiate encryption algorithms and cryptographic keys before data gets exchanged [29].

HTTPS

HTTPS (Hypertext transfer protocol secure) is a protocol for accessing a secure web server when authentication and encrypted communication is possible. It directs the message to a secure port number, and then the session is managed by a security protocol. HTTPS is used together with TLS to encrypt the session data. In addition to this, the use of digital certificates gets supported so that a user can authenticate the sender [28].

3.2 Overview of related messaging applications

The following part gives an overview of common most frequently used messaging applications and implement approaches for security and privacy protection. In table 3.2 a summary of the most important properties is portrayed. In addition to this, each application and its functionality gets explained shortly. WhatsApp, Telegram, and Viber are the most popular and a more detailed comparison of them is published in [6]. Other applications that provide similar approaches to communication security, but do not get described, are Signal, Wickr, Whisper, and Blind.

3.2.1 WhatsApp

WhatsApp[2] is one of the worldwide most known messaging applications and offers text and audio messaging, free voice calling or sharing different document types. It allows reaching all contacts in the address book on the smartphone who have installed the same application. Messages, files, and even phone calls are end-to-end encrypted so that neither WhatsApp or any third-party can access it [34]. Data that gets collected are account information (mobile number, mobile address book, other account data). Messages get persisted until they get delivered to the recipient or for 30 days, rather than media content which gets stored for a longer time on the server. WhatsApp uses no kind of screenshot protection or time limitation of messages, but there exists the option to delete messages afterward [33].

[2] https://www.whatsapp.com/

Parameters	WhatsApp	Telegram	Viber	Snapchat	Dust	Confide
Saves data on server	✓	✓	✓	✓	✓	✓
Mobile number as registration data	✓	✓	✓	✓	✓	✓
End-to-End Encryption	✓	✓	✓	-	✓	✓
Screenshot detection	-	-	-	-	✓	✓
Screenshot protection	-	-	-	-	-	✓
Time-limited messages	-	-	-	✓	✓	✓
Option to delete messages	✓	✓	✓	✓	✓	✓

Table 3.2: Overview of related messaging applications.

3.2.2 Telegram

Telegram[3] is a cloud-based mobile and desktop messaging application with a focus on security and speed. It has many similarities to WhatsApp. Again the phone number is used as primary user identification, contacts in the smartphone that use the same application get found, and messages are end-to-end encrypted. In addition to this Telegram provides functionality that users can also create a username as a unique id to use it for conversation. Thus privacy protection gets enhanced. Also, a higher variety of file types gets provided and are possible to share without size limit. Chats can be created as general or as secure conversations. For general chats, the data sent gets stored on the server. Also, the registration data (mobile number) and contact book data get persisted. When using secret chats, nothing except media files gets saved. There is no screenshot protection or time limitation of messages given, but again the option to delete a message is provided [31].

3.2.3 Viber

Viber[4] is a mobile messaging application that provides some further features in addition to the standard functionality of the other applications. That would be high-quality video- and voice call, a calling service that can dial to any mobile number, even if the dialed number is not using Viber and so-called public chats that allow users to

[3] https://telegram.org/
[4] https://www.viber.com/

communicate openly. The mobile number gets used for registration, conversations are end-to-end encrypted, and messages can be deleted. As for the other applications, account information gets stored permanently and chat data gets saved temporarily until the recipient reads it. Viber also does not offer some screenshot protection algorithm or the time-limitation of messages [32].

3.2.4 Snapchat

Snapchat[5] is a mobile image and multimedia messaging application. Therefore it provides more functionalities than a simple messaging application. One of the principal concepts of Snapchat is that users can share text, pictures, and videos that are only available for a short time before they get automatically deleted. Meanwhile, there exists the possibility to save messages in so-called memories. Also, the option of creating a Snapchat story exists. These consist of "snaps"[6] and are posted for 24 hours, though they can be deleted at any time, but can also be saved by other users. The name and mobile number get used for registration and all additional account data (username, email, birthday) provided gets stored on the server. There can be found no further information if chats are end-to-end encrypted and in iOS, there is no screenshot detection functionality provided [30].

3.2.5 Dust

Dust[7] is another messaging application that combines the time-limitation of messages, a screenshot detection method and end-to-end encrypted chats to provide secure communication. Users have to provide their mobile number, email address and a username to use it. These account information also gets stored on the server, in addition to the user's contact book and photo rolls that get sent. Messages get deleted automatically after the receiver read it or after an amount of time that can be set by the sender. Screenshots are detected, and the sender gets informed if a user has taken a screenshot [24].

3.2.6 Confide

Confide[8] is a messaging application that is available for various platforms and ensures private communication. It uses end-to-end encryption of chats and allows to send text, photos, videos, documents, and voice messages. One feature is the self-destruction of data after it gets read. This messenger has a highly developed secure screenshot protection technology which is called Screenshield-Kit.[9] In the premium version of Confide, the users can retract messages after they were sent. The application also stores the registration data (phone number, email address and name) on the server, whereas messages only get saved temporarily [23].

[5] https://www.snapchat.com/
[6] "Snap" is the shortcut for snapshot.
[7] https://www.usedust.com/
[8] https://getconfide.com
[9] https://screenshieldkit.com

3.3 Summary of common concepts

After all the related applications got investigated, the following concepts turned out to establish increased security and privacy protection in mobile communication.

3.3.1 User authentication

When using any messaging application, a necessary part is user authentication. It has to be ensured that only valid users have access to the full functionality of the application and that the communication parties can be identified. By this, the data gets protected, and users always know with whom they are communicating.

3.3.2 End-to-end encryption

The most used technique for secure communication is cryptography, of course. Nearly every messaging application provides end-to-end encryption to ensure that only the recipients can read the plain message text. When using asymmetric key pairs, the data transmission gets even more secure, because only the holder of the private key can decrypt the content of a message. In addition to this, by using protocols like TLS, possible communication attacks get prevented, and an extra layer of security and privacy is established. By using cryptographic keys to sign and verify messages, it can also be detected if the message was altered on the network transport and the identity of the sender can be guaranteed.

3.3.3 Deletion and time limitation of messages

The option to delete messages or that they get removed automatically from a server when the recipient reads it is a further method to gain higher data security. Another technique used to avoid data recording is to work with time-limited messages. By removing the messages from the server and smartphone after the user has read them, it gets ensured that the data stays in a transient stage and therefore is not verifiable any longer. There is no possibility to gain access to any message that has been sent at a later time point. A disadvantage of this concept would be that the users have to remember every part of the conversation. However, when finding new methods to establish higher privacy and data protection, this seems like a right approach.

3.3.4 Screenshot detection/protection

Another point to consider is the possibility for users to screenshot the content and therefore to risk that it gets stored once again. By this, the potential of making the impermanent permanent is given and thereby the previously stated time limitation technique would be revoked. Therefore an additional concept to prevent users from persisting messages and keeping the data transient is to implement methods for screenshot detection and screenshot prevention.

Chapter 4

Conception

In this chapter, the concept of building a secure and anonymous messaging application gets explained. First, the preliminary considerations that are necessary for planning the further implementation get explained. After that, the requirements get described and resulting from there the concept is determined. Finally, an application architecture gets planned from the previous considerations.

4.1 Preliminary considerations

Before thinking about any further implementation plan, it is necessary to elaborate on all topics and concepts that establish higher privacy and security in mobile messaging applications. Correlated to this it must be questioned what the main related goals are and furthermore which points have to be considered for the implementation to realize all these objectives.

4.1.1 Research results

Hereafter the principal results of research get described briefly and afterward get discussed in more detail. Each of them is highly relevant for providing improved privacy and security in a messaging application, and build the basis for the further conception and implementation.

Registration

A significant topic concerning the privacy of users is about the kind of data used for registration, how personal it is and which parts get saved for later authentication. Regarding the personal privacy and to provide anonymity, registration should be possible without any personal data.

Exchange of contact data

If there exists no personal registration data that can be used as an address for sending the messages to, some way of exchanging the contact data has to be found. This part should not demand much effort for the user. It should be kept simple, and the necessary

steps to add new contacts should be straightforward. Therefore two different options, one for exchanging the contact remotely and one for exchanging it locally has to be found.

Data security

In general, the correct performed implementation of data security is also highly relevant. By this, all data that gets used at some point in the application flow is protected. Therefore one of the main issues is the encryption of messages and additional content of the communication process.

Data transience

This part is strongly related to data security. By providing a short lifespan for all messages, the content sent keeps in a transient state, and it gets ensured that it is not traceable. It leads to automatic data maintenance, and therefore no extraordinary user input is needed. The data gets deleted automatically after reading it and disappears forever.

Prevention of persistence threats

The last requirement for building a secure messaging application is to eliminate other possibilities where the data gets persisted. That would be for example taking screenshots, filming, and taking audio records of the messages. Therefore some solution should be found to prevent users from persisting the data through these activities.

4.1.2 Traceability of users

Regarding the traceability of users, all implementation options that support privacy protection have to be prioritized. Anonymity should be a significant part to ensure that no personal data is retraceable to its origin. In particular, the topics of registration and data security, as described in section 4.1.1, refer to this concern.

There should be no particular prerequisites for the registration, and the user should not have to provide any personal details for the registration process. Therefore no mobile numbers, email addresses, names or other data should be required. The registration process should be automated at the first launch of the application and run in the background. This problem could be solved by generating a random id. This id can then be utilized as an address for sending messages to.

In addition to this, the encryption of the messages is also a relevant topic. All parts of the communication process should be secured and must not be accessible or readable by any third party. Therefore the chats should not only be symmetric end-to-end encrypted but deal rather with a lot more reliable procedures like using asymmetric keys for encryption and signing the data. Thus just the intended endpoints can access the plain text or content of a message. Another significant consideration is the protection of the private key against unauthorized access. Therefore it has to be stored in a highly secure part on the smartphone.

When no personal data is used for the registration process, another secure authentication method has to be found. This process is necessary for protecting the access to

specific application functionalities and for verifying the sender behind a message. Only registered users should be able to use the full function range of the messaging application. Thus some authentication process that works automatically in the background and again uses no personal data should be implemented. The verification procedure for finding out the real sender of a message can be performed by using the asymmetric keys for signing the message data.

4.1.3 Exchange of contact data

A decisive factor for ensuring anonymity is the part of exchanging contact data, as briefly described in section 4.1.1. When only a randomly generated id is used as verification address of a user, there has to be provided a way how it still can be exchanged.

There occur several challenges for providing privacy protection while swapping the respective digital user addresses to start communication. In particular, these include the detection and implementation of a local and a remote option for the exchange. Irrelevant if this happens via a local or remote communication channel, it has to be as secure as possible. This is highly relevant due to the kind of data that is required for adding a contact. As fundamental elements, this data contains the id as address endpoint and the public key for the encryption of messages. At least these are necessary to facilitate the basic messaging functionality.

Concerning usability, it is also of high importance that the contact exchange is easily executable for the user and that no extraordinary effort is necessary. Possible options for locally exchanging the contacts would be scanning a QR-code and transferring it via NFC or Bluetooth connection. For transmitting the data on a remote channel, it is unavoidable to use another digital communication medium. Therefore this part has to be implemented to be as safely as possible.

4.1.4 Data transience

Another critical point is the transience of data in the whole communication process, as explained in section 4.1.1. Messages should only exist until the receiver read them and should not get persisted on any server, database nor on the smartphone.

By implementing a functionality which tracks when a user has obtained a message, the right time point to delete it can be recognized. So it does not get persisted too long but also does not get deleted too fast. Each content should automatically self-destruct and erased from every storage medium included in the communication process. With the step of removing the data from the smartphone and the database after it has been read, the data stays in a transient stage and is not verifiable afterward.

Concerning usability, there should be no extra effort for the user to care about the deletion of the messages manually. Therefore everything should happen automated in the background. On the contrary of general messaging applications, the only significant change by using transient data is that there exists no chat history and the user has to remember the last messages. By this, a new communicational aspect is added to every conversation, because it has to be memorized and therefore seems like a usual face-to-face talk.

4.1.5 Prevention of persistence threats

As shortly characterized in section 4.1.1, the last point worth noticing regards to occuring persistence threats, even if the data gets automatically deleted from client and server. For providing higher privacy protection, attention must also be paid in preventing these additional persistence threats.

By this, all activities a user or other person can undertake to persist the messages are meant. Primarily associated with this is to take screenshots with the smartphone. Therefore some functionalities should be implemented that either detect and prevent a user from taking a screenshot or at least making them unusable by automatically destroying them through image processing.

One major problem concerns all further persistence methods that include additional recording mediums. By these, taking pictures or videos of the messages with other devices is possible. It will become difficult to prevent all these specific threats in the messaging application. Therefore it is essential to research and try out several methods of protecting the data nevertheless.

4.2 Requirements

After the several challenges appearing in secure messaging were determined, the next step is to find out how the goals can be achieved. According to the issues described before, following topics have been emphasized to be the primary requirements for building a secure messaging application.

4.2.1 Frontend

The basis for all further steps of the project implementation is to develop an application that provides full fundamental functionality for messaging. These include the part for registration, the management of contacts and the underlying messaging functionality. The onboarding should be automated. Thus the account gets created without any further user input and gets stored on the server database in the background.

Furthermore, the application has to include functionalities for the management of contacts. Basic options like adding, editing and removing the contacts should be given. Therefore the data should be stored locally on the smartphone database and should be manageable at any desired time point. The last fundamental requirement is to enable the basic messaging functionality. Hence an effective client-server communication has to be implemented. In addition to this, it should be possible to send several data types like text, images, and so on.

4.2.2 Backend

In addition to the frontend, the backend has to be implemented to provide the client-server communication functionality. The responsibilities of the server include the following topics: communication with the application, the registration and authentication process, providing a working database connection, preliminary file storage and sending push-notifications if a user has received a message.

One of the primary purposes is to establish a fully functioning communication to the application and the connection to the database to manage the data. All other processes depend on these basic methods. In addition to this, it should be possible to store files on the server for sending images and other data temporarily.

At the registration, a new user should be created automatically by sending a request to the backend. By this, an id and a token for the authentication process get generated. A basic token authentication will protect the access to all other functionalities. So a user has to be a member to get access to the full features of the application. Through this, the unintended spamming of third-party entities should be prevented.

Another task would be the automated sending of push-notifications when a user received a message. This function has to be done by the backend when a request for adding a new message data record to the database has been received.

4.2.3 Encryption/decryption of messages

For the encryption of messages, a suitable encryption method and algorithm has to be found. As already defined before, the method used should be asymmetric. The creation of the keys should happen at the onboarding without any further user input. In addition to this, the private key has to be stored for performing encryption and decryption of the messages. The memory point where the key gets stored has to be highly secure. In iOS, this is possible by saving it on the secure enclave[1] of the smartphone. Only the private key gets stored because the public can always be generated from it.

After this step, the key pair can be used at any later time point. For example, when exchanging the public key with other users, so that they can encrypt data with it or for decrypting received messages with the private key.

4.2.4 Verifying/signing of messages

The asymmetric key pair is also very suitable for signing data, so that a simple verification process can be executed at the decryption of messages. For this, an appropriate signing algorithm and process has to be found and implemented. It should work together seamlessly with the chosen encryption algorithm and key pair. By signing a message with the own private key and sending the signature hash additionally to the encrypted data, the receiver can use these contents for verifying the sender. Both contents get decrypted, and the values get compared. If they are the same, the sender is verified, and it is ensured that the data did not get manipulated.

4.2.5 Exchange of contact data

For exchanging the contact data, a proper implementation has to be developed. Therefore two separate options have to be designed, a local and remote functionality. For the local variant, a QR-code scanner should be deployed. If a contact is added, the QR-code of the person can be scanned, and the related id gets stored to the application database. If a user wants to add a contact remotely, the idea is to send the public id via a third channel. In the application should be an option where the own id can be copied to the

[1] The secure enclave is a hardware processor that uses encrypted memory and provides cryptographic operations for data protection [21, p. 7].

clipboard and then can be reused in any other communication channel to transmit it. For adding a new contact, an input field should be provided. In this field, the id can be filled in and gets saved to the application. Additionally, a local database storage should be implemented. By this, it gets ensured that contacts only are saved to the smartphone, and no relations between users can be determined server-side.

4.2.6 Data transience

A mechanism for ensuring that all data stays in a transitory stage should be developed and implemented. The idea is to fetch all received messages from the server, where they should be immediately deleted afterward. Concurrently the messages should be saved to the application database. When a user opens a message to read it, it also has to be removed locally from the smartphone. To provide these functionalities the server has to allocate an endpoint for sending a request to, that allows fetching the messages and simultaneously deleting them from the remote database. For the application, the local storage, display, and removal of the messages should be implemented.

4.2.7 Prevention of persistence threats

The last goal for the application is to develop the part to prevent users from any further possibility to persist the messages. Several opportunities that would threaten the privacy and data security of the users have to be considered and disabled. In particular, these include taking screenshots, pictures or any further recording to persist the content of the message. For this, a possible solution would be to use animations in combination with image processing. The text of the messages should be animated repeatedly in a reading direction. So it gets ensured that the user did not miss something and read the whole text. For the display of images and other files, a likewise method should be used, with the addition of implementing blurriness for hiding specific areas. Therefore the content of the image should be covered with an extra layer that obfuscates nearly everything, except a small stripe that gets displayed. Furthermore, this layer should then be animated in a certain speed, so that the user can see the whole picture. However, only a small area will be visible if someone tries to persist the content by taking a screenshot or photo of it.

4.3 Backend architecture

Based on the preceding considerations an architecture for the frontend and the related backend functionality should be determined. The first steps for the project are the planning and structuring of the development process. Furthermore, the architectural conception used for the implementation should be selected.

For the backend implementation, the architecture mainly depends on the required functionalities and the chosen framework for building the server-side part. Another point to consider is the kind of database that should be implemented. It should conform to the structural requirements of the data objects and relations needed for the application. Due to the consideration that the frontend and backend part should be implemented

as a full-stack application, the framework chosen is Vapor[2] which is also based on the language Swift. It will be explained in more detail in section 5.1, but for now, it is only mentioned to explain the further architecture plan. The pattern that it is based on is MVC, and all components are built upon this structure. In the following subsections, each component needed for the application gets described in its mean and functionality.

4.3.1 Models

A model is the representation of the related object in the database. Each property of a model is mapped to its belonging attribute in the database table. For managing the entities, some basic methods are provided. These include the CRUD[3] operations which enable functionalities for maintaining the data resources. Each model has its own lifecycle and can be manipulated at each different stage of the cycle. For the messaging application, the following models are planned to be implemented.

User model

The *user model* should be created at the registration and should exist for authentication and an address to send messages to. It should consist of properties used for identification and further metadata.

Message model

The *message model* has to be created each time a message is sent to a user and should then be saved in the database. It should consist of the message content(s), sender-receiver information and additional metadata. A one-to-many relation between user and messages should exist so that a user can receive multiple messages.

Token model

The *token model* should be created simultaneously with the user model at the registration and should exist for establishing a simple token authentication. Typically it consists of a token string and has a one-to-one relation to the belonging user object.

4.3.2 Controllers

A controller is used to create RESTful resources and organizing related functionalities in a single place. It contains the logic for managing incoming requests, sending responses and working with the data models. Each method of a controller can be mapped as an action on a specific route. These routes can be executed by sending a request to it.

User controller

The *user controller* should manage the related *user model* and therefore is responsible for the registration, editing, and deletion of the models. In addition to this, the controller should provide a function for loading and deleting the messages of a specific user.

[2] https://vapor.codes/

[3] CRUD (Create-Read-Update-Delete) are the basic functions for persistent storage editing.

Message controller

The *message controller* includes all tasks for working with the related *message model*. These include the creation of the messages, but also loading attached files from the server.

4.3.3 Middleware

A middleware allows modifying incoming requests or outgoing responses so that further functionality can be added before forwarding the request to the controller action or the response to the client. In general, middlewares are suitable for catching errors or interacting with requests. It is also possible to chain several middlewares before passing it to the controller action. For the backend part of the messaging application, the following middleware should be implemented.

Authentication middleware

A middleware for authenticating users should be implemented. By this, all functionalities that should not be able to access by unregistered users get protected. It should use a basic token authentication to verify user permissions and to protect specific routes from being executed.

4.3.4 Database

An essential part of the backend implementation is the database. It will be the storage location for all the data used by the messaging application. It should be ensured that the kind of database chosen, meets all the requirements needed and can be smoothly integrated. The idea for this application is to implement a document-oriented database, which means that the data for each model gets stored in one single document instance instead of fixed tables. Each document can have a dynamic scheme, can be retrieved by an id and relations between objects can be set. Using this database type better scalability, high availability and performance can be established.

4.4 Frontend architecture

For the application, the decision was to implement an iOS application using Swift as the programming language. Again MVC or MVVM, as described in section 2.5, exist as common concepts for building such application architectures. These are usually used patterns for developing mobile applications, but they also bring some disadvantages with them. Therefore the VIPER architecture that provides new concepts and other approaches for iOS development has been chosen.

4.4.1 VIPER

VIPER [27] is a modern architecture for iOS application development and a backronym for *view*, *interactor*, *presenter*, *entity*, and *router*. It implements the single responsibility

principle[4] to create a cleaner and more modular structure for iOS projects. The idea behind this pattern is to isolate the application's dependencies and therefore to balance the delegation of responsibilities among the entities.

Therefore this architecture is characterized by more reusable and extensible code. Adding new features is easy, and it is highly testable. Each application responsibility is based on an use case, and then the architecture gets split up into modules based on these use cases. In figure 4.1 the architectural structure of one module is displayed. In the following pages the structure for building the messaging application based on VIPER gets described.

Router: The *router* contains the navigation logic for switching between the modules. It receives input commands from the *presenter* to what screen it should route to. In addition to this, it is responsible for passing data from one module to the other.

Entity: The *entity* encapsulates different types of data and is usually treated as a payload among the other components. One important thing to notice is that the *entity* is different from the other data objects that get used in the data access layer, which should be handled by the *interactor*.

Presenter: The *presenter* contains the view logic for preparing the data for display and for reacting to user input. The *presenter* receives input events coming from the *view* and reacts to them by requesting data from the *interactor*. Another function of it is receiving the data structures coming from the *interactor*, applying view logic over this data for preparing the content and finally telling the *view* what to display.

Interactor: The *interactor* contains the business logic related to the entities and is responsible for sending the correct data to the *presenter*. It should be implemented entirely independent of the user interface.

View: The *view* sends the user actions to the *presenter* and shows whatever the *presenter* tells. When the user triggers an event that requires processing, the *view* delegates it to the *presenter* and waits for its response what should be displayed.

For now, only the underlying architecture for one module in the application has been illustrated. Every module should be built upon this structure, and there have to be added some modifications for the final structure that will get described hereafter.

4.4.2 Provider

All the parts that provide a specific service should be accessible throughout the several modules. Therefore some protocols and components have been added and contain the general business logic that is needed on various points in the application. Each component should be handled as a singleton and reachable via a shared property. For the moment the following providers should be added to the architecture.

[4]SRP (Single-Responsibility-Principle) is an architectural pattern that separates different responsibilities of an module into their own parts.

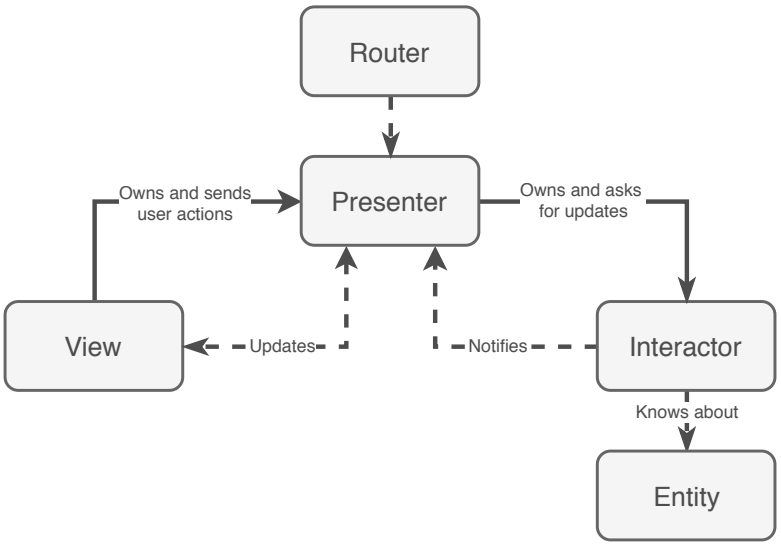


Figure 4.1: Structure of one module in the VIPER architecture.

Backend provider

The *backend provider* should establish all the functionality for the communication with the backend and fetch the needed data from the database. It has to include all features for sending REST calls to the backend and handling the responses. Also, some specific models used to send and receive the data are needed for exchanging the data.

Crypto provider

The *crypto provider* should be responsible for creating the asymmetric key pair and saving it to the secure enclave. Moreover, the functionality for encryption and signing/verifying of the messages has to be provided.

Storage provider

The *storage provider* should include the methods for storing all the needed data to the application database. That would be saving the own authentication values and storing the contacts and messages received. Here again, some models used for storing the data correctly in the database have to be implemented.

4.4.3 DTOs

The DTOs[5] should be basic model objects used on several points through the application. In the architecture the DTOs defined, are one for managing the contacts, one for displaying the message-info in the overview and one for showing the full message. They have to be considered differentiated from the entities of a VIPER module for the reason that they get used globally throughout the application.

[5]DTO (Data-Transfer-Object) is an object that carries data between processes.

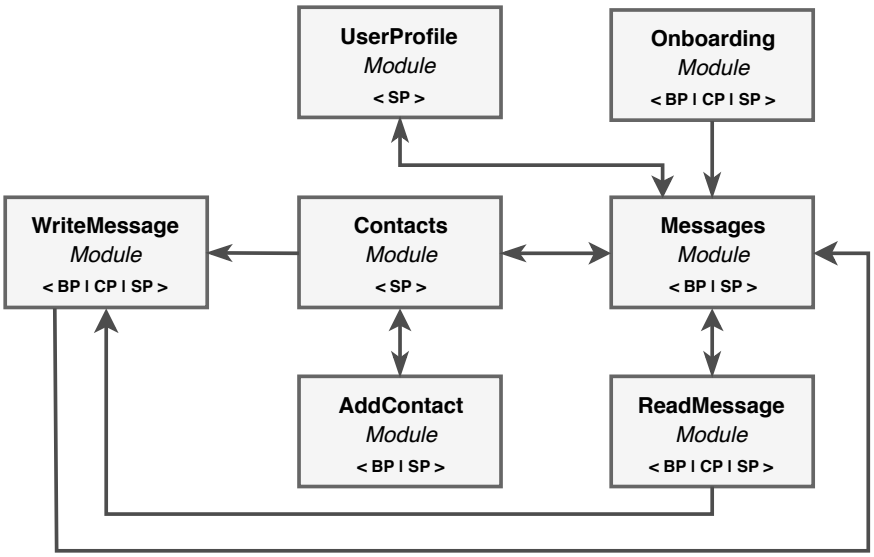


Figure 4.2: Architecture plan of the individual modules.

4.4.4 VIPER modules

For the application the following modules have been specified. They are split up by their use cases and can easily be modified. Therefore the whole architecture can be extended simple. In figure 4.2 all relations between the modules are displayed. Furthermore their relations to the providers are presented in the figure, by using the shortcut BP for *backend provider*, CP for *crypto provider* and SP for *storage provider*.

Onboarding module

The *Onboarding module* is responsible for the registration of the user and the generation of the asymmetric key pair for encryption. Furthermore storing the keys in the secure enclave and saving the account data into the application storage are tasks of this module.

Messages module

The *Messages module* serves as the main entry point of the application after the registration and provides the functionality for fetching the messages from the server database. In addition to this, it is responsible for the presentation of the messages and operates as navigation view that leads to all the other modules and features.

Contacts module

The *Contacts module* serves as a presentational overview of all the contacts that have been saved to the local application storage. It also gets used for choosing the intended receiver when a user wants to write a new message. Therefore it is responsible for loading all stored contacts from the database and passing the selected data to the next module where it can be processed.

WriteMessage module

The *WriteMessage module* is responsible for providing an input area for writing and furthermore sending the messages via a request to the backend. Before that, each message gets encrypted and signed with the public key of the receiver. To make this possible, another functionality for loading the user data from the application storage should be maintained.

ReadMessage module

The *ReadMessage module* provides the functionality for decryption and verification of the message chosen to be displayed. Another responsibility is the animation of the text and the image processing of files for the prevention of persistence threats. Also, the deletion of a message from the local application storage after it has been opened is related to this module.

AddContact module

The *AddContact module* provides the option to add a new contact. It includes two different features for storing a contact, one that works locally and one that can be used remotely. The first one is a barcode scanner where the user can scan the QR-code that includes the contact data of another user. The second option is to enter an id to an input field received via a third channel.

UserProfile module

The *UserProfile module* is indispensable for the contact exchange part because it displays the generated QR-code and provides a button for copying the id for the remote exchange. Thus the responsibilities are loading the own authentication values and the generation of the id-based QR-code.

Chapter 5

Implementation

In this chapter, a more detailed description of the implementation and the technical design of the frontend and backend gets declared. First, the deployed technologies and methods are shown, then the actual application flow and all the corresponding processes will be illustrated. Finally, the design decisions that have been taken throughout the development process get described briefly.

5.1 Backend

For the backend part, the implementation is mainly based on the chosen framework. As shortly mentioned before in section 4.3, Vapor has been used for the serverside application. Therefore all methods and technologies are built upon it. Hereafter the framework gets described briefly to give an understanding of how the remaining parts have been developed.

5.1.1 Vapor

Vapor [17] is a modular framework used for building web applications and follows the MVC principle. In addition to the provided base structure, the code is split up into modules which are grouped in packages. Each module provides some particular functionality needed for a specific part of the server implementation. Therefore applications that use this framework can be individually adjusted to specific needs. It uses Swift as the programming language and thus supports a full stack development of the project.

Droplet

The droplet is a service container which is the basis for every application and gives access to all further functionalities provided. It is the main entry point and therefore includes all the configuration and setup to ensure everything works together. Here all the provider with specific functionalities or third-party libraries get added to the application. In addition to this, the connection to the database gets established, and the routes to the RESTful services of the controller and middlewares get configured. For the messaging application, the provider and frameworks used get explained in the following paragraphs.

```
_id: "5AD45AA1-89FA-46E5-AF5E-D6E5CC8441A5"
public_key: "BDUh8mWxGfmi/Cq8tklW5DBZYeEM1aKtduRVcsBSIPfuAp+NCqR4GRrsEyaNt/TstNnndV..."
push_token: "06f726313ac8ef24fd32270f4fa43e75eb0f873e06419982066cba0436d1f365"
device_info: "92F9BE70-157F-4937-A63E-DF2072634F74"
```

Figure 5.1: The `UserModel` in the database.

Fluent-Provider: The Fluent-Provider [12] adds the Fluent module [11] to the project and gives the functionalities to work with it. Fluent provides ORM[1] functionality for working with the database. It gets inherited by the models and provides CRUD functionalities to work with the database objects.

Mongo-Provider: The Mongo-Provider [14] itself is a module that combines the functionalities of the MongoKitten [15] framework and the Mongo-Driver module [13] to support the usage of Fluent models together with the MongoDB[2] database.

Auth: Auth [10] is another module that provides functionality for authentication and authorization. In the project, it is used to allocate the simple bearer token authentication to prevent stranger parties from spamming the server.

Vapor-APNS: Vapor-APNS [18] is a simple Swift library that provides methods for configuring the APNs-service[3] for sending push-notifications together with Vapor. It has been implemented due to the essentiality of today's messaging applications to deliver notifications in real-time.

5.1.2 Models

As defined in section 4.3, only three models have been implemented. All of them inherit the Fluent module to enable the ORM functionality for working with the database. There exists one model for the messages, one for the users and one for the tokens to provide the authentication. Each model is easily convertible into JSON, so that it effortlessly can be used, either for sending responses to the application or storing data directly in the database.

User model

The `UserModel` represents a user in the database. As illustrated in figure 5.1, it consists of an id, a device-info, a push-token and a public key. The id is used as an address to send messages to. The device-info helps with managing different versions of the application. The push-token is needed as an endpoint for sending the push-notifications to. The last property is the public key that has to be provided for other users so that they can encrypt messages with it.

[1] ORM (Object-Relational-Mapping) is a technique for converting data between incompatible type systems using object-oriented programming languages.

[2] https://www.mongodb.com/

[3] APNs (Apple-Push-Notification service) [20] is a service provided from Apple for sending automated push notifications.

```
_id: "16D04FC0-6FCF-4856-A4D6-9E262B43D0B1"
user__id: "5AD45AA1-89FA-46E5-AF5E-D6E5CC8441A5"
token: "78CE4E39-8A39-4B7F-87FE-84DDB592E031"
```


Figure 5.2: The `TokenModel` in the database.


```
_id: "400C6616-D38B-454F-B56A-23AAE1D5C0AD"
content: "BD8ftCzW7jSWSuA8n4rcewqUwvpvs5gKhOdhqXQiz6sVBaE/wYPzBkIfVmrtxgoqxgJF/c..."
date_sent: 1529604558.464085
sender_id: "EDAB3C21-AA50-4F3D-8866-1DAEB000613A"
attachment: "26223946-447E-4342-90DA-840C4697F9A3.txt"
user__id: "5AD45AA1-89FA-46E5-AF5E-D6E5CC8441A5"
signature: "MEQCIE533uJQQ6vg41SbyUfTI8j5Zp6IV1TD5MnOI0Xu85SzAiAuVWWU9oFCuNm/It65Kr..."
```


Figure 5.3: The `MessageModel` in the database.

Token model

The `TokenModel` gets used for the authentication of a user. The properties defined can be looked up in figure 5.2. It includes a unique id, a token, and a user id as the foreign key that leads to the related user.

Message model

The `MessageModel` is used for the messages in the database. In figure 5.3 each attribute gets displayed. In the content property, the encrypted text of the message gets saved. Additionally, the signature hash for the verification process is stored in the signature. If a file gets sent, an id that is representative for the location of the related file on the server gets added in the attachment property. The receiver id is a foreign key that leads to the user that the message belongs to. The last parameters are one for the sender id and one for the date when the message has been sent.

5.1.3 Controllers, routes and middlewares

As described in section 4.3.2, the main functionality for the communication between the application and the backend is maintained by the controllers. Each action is mapped to a specific route used for sending requests to the backend. As explained in section 4.3.3 an additional middleware has been implemented for the messaging application. Its purpose is to protect selected routes from unauthenticated access. Hereafter only the implementation of the controllers and their actions get described.

User controller

The `UserController` includes the functionality for creating, updating and deleting a user, but also for retrieving and deleting the messages of the user. In addition to this, another option to load the public key of a particular user is implemented. An overview of all actions that are related to this controller is shown in table 5.1, and the essential actions get described in more detail hereafter.

Action	*Method*	*Path*	*Auth*
`createUser`	POST	api/user	X
`getUserKey`	GET	api/user/:user_id	✓
`updateUser`	PUT	api/user/:id	✓
`deleteUser`	DELETE	api/user/:id	✓
`getUserMessages`	GET	api/user/:id/messages	✓

Table 5.1: Overview of the `UserController` actions.

Action	*Method*	*Path*	*Auth*
`createMessage`	POST	api/message	✓
`getMessageAttachment`	GET	api/files/:name	✓

Table 5.2: Overview of the `MessageController` actions.

`createUser`: When a user gets created, also the related authentication token is generated, and their relation gets stored in the database.

`getUserKey`: This action is provided for adding a user as a new contact. It loads the related public key of the specific user and returns it in the response.

`getUserMessages`: By calling this action, all messages related to the specified user get loaded and sent back as a response. Immediately after it, all messages get deleted from the database.

Message controller

The `MessageController` provides all the methods for creating a message, sending a push notification to the receiver when a new message was created and setting the correct relation to the user in the database. In table 5.2 an overview of the actions that are implemented can be looked up. Hereafter they get clarified more precisely.

`createMessage`: This action is for creating a new message in the database. If a file gets uploaded, its name gets changed to a random id and gets stored in the `public` folder on the server. In addition to this, another functionality is the automated sending of a push-notification to the receiver of the message.

`getMessageAttachment`: When a file was sent with the message, the related path has been saved in the attachment property of the message object. By this action, the file can be loaded from the server and sent back as a response. In addition to this, the file gets immediately deleted afterward.

5.2 Frontend

For the frontend application, the implementation has been realized as described in section 4.4, where the application architecture is explained. In the following pages each module, its structure, and functionalities get illustrated. After that, the details and technologies of the individual requirements characterized in section 4.2 that are important for the outcome get outlined.

5.2.1 Module structure

Before the individual modules get described, the exact structure of the implementation of each component gets emphasized briefly. Therefore the classes and interfaces mentioned, are signed in the text with the prefix `<Module>`. An overview of the architecture is displayed in figure 5.4. As can be seen, the entity component of the VIPER architecture is missing. This is due to the reason that all entities needed in the application are either adopted by the providers, created as individual models for the view display or get used by several modules and therefore are implemented as DTOs.

Storyboard

Every module has its `<Module>Storyboard`[4] where the layout of an individual screen is defined. It consists of all UI elements required for the specific use case that it should display. Furthermore, it contains connections (`IBOutlets`[5], `IBActions`[6]) to the UI elements in the code.

View

Each view component consists of a `<Module>View` which is an interface, including the functionalities called by the `<Module>Presenter`. Furthermore, a `<Module>ViewController`[7] implementing this interface and connected to the related storyboard is established. All event handler (`IBActions`) that listen to specific user actions on the UI elements are carried out in the view controller. In addition to these, the `<Module>View` owns the event handler interface of the `<Module>Presenter` for passing the specific user action events to it. For some of the views, an extra `<Module>ViewModel` has been implemented.

Interactor

The `<Module>Interactor` is responsible for managing all the use cases related to the module. Therefore a `<Module>UseCases` interface, consisting of the methods for the several use cases, is implemented by it. It also contains relations to the specific providers needed for executing the use cases.

[4] A `UIStoryboard` is used to build the user interface.
[5] A `IBOutlet` references an element of the user interface.
[6] A `IBAction` exposes a method as a connection point with a certain element of the user interface.
[7] A `UIViewController` manages the views and user interface elements.

Presenter

The `<Module>Presenter` contains relations to the router, the interactor, and the view and implements a `<Module>EventHandler` interface. It invokes methods of the interactor for requesting specific data, forwards commands for the display to the view and passes navigation calls to the router. The functionalities of the event handler get called by the connected view when a particular event should be executed.

Router

The router is separated into a `<Module>Router` interface implemented by the `<Module>Wireframe`. The `<Module>Router` contains methods for navigating between the modules, and the wireframe has a relation to the `<Module>ViewController` for displaying it. By providing a static class function named `show`, the wireframe gives the functionality to instantiate and connect all the other components from the outside. Due to the reason that this method is responsible for connecting all the components, the following lines of code display the basic structure of the call hierarchy:

```
1  class func show() -> UIViewController {
2      let viewController = UIStoryboard(name: Constants.storyboardName,
3      bundle: Bundle.main).instantiateInitialViewController() as! ModuleViewController
4
5      let useCase = ModuleInteractor(backendProvider: Provider.backendProvider,
       storageProvider: Provider.storageProvider,
6      cryptoProvider: Provider.cryptoProvider)
7
8      let router = ModuleWireframe(viewController: viewController)
9
10     let eventHandler = ModulePresenter(view: viewController, router: router,
       useCase: useCase)
11
12     viewController.eventHandler = eventHandler
13
14     return viewController
15 }
```

By enforcing the `show` method, the following steps get executed:

- The `<Module>ViewController` gets instantiated by connecting it with the related storyboard and setting the view controller for the display (line 2).
- After that, the `<Module>Interactor` with references to the needed providers and the use case(s) of the module gets constructed (line 4).
- The next step is to create the `<Module>Wireframe` and apply the generated view controller to it to pass it later to the display (line 6).
- Then the `<Module>Presenter` is set up and its relation to the view controller, wireframe and interactor are established (line 8).
- The last procedure is then to attach the presenter to the view controller so that it can access the event handler methods for invoking them at the specific user action events (line 10).
- Finally, the method returns the view controller with all set properties, to display it in the window's root view controller which handles the presentation on the screen (line 12).

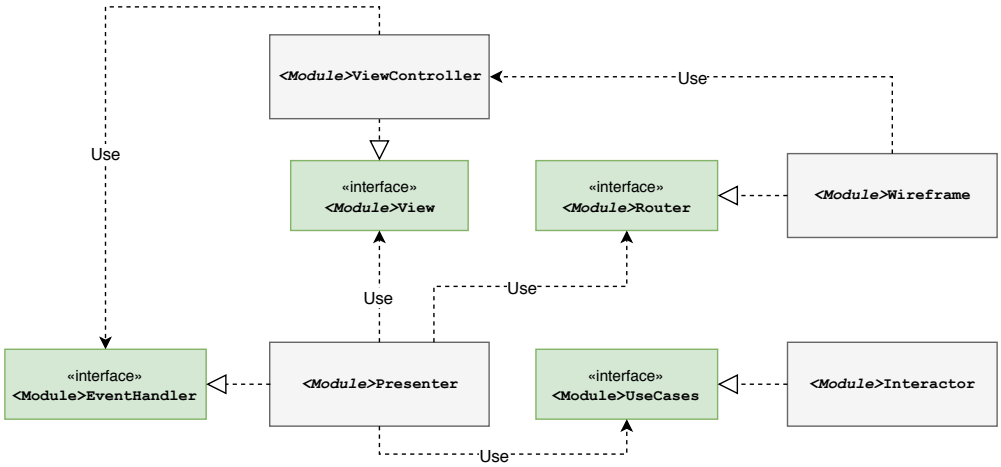


Figure 5.4: Overview of the structure of one module in the application.

For each module this procedure stays the same, if another module should be loaded by the router, the wireframe's interface method for executing the needed `show` method of the module that has to be shown as next, gets called and manages the further process. Therefore the interface including methods for navigating between the modules, is also implemented by the wireframe.

5.2.2 Onboarding module

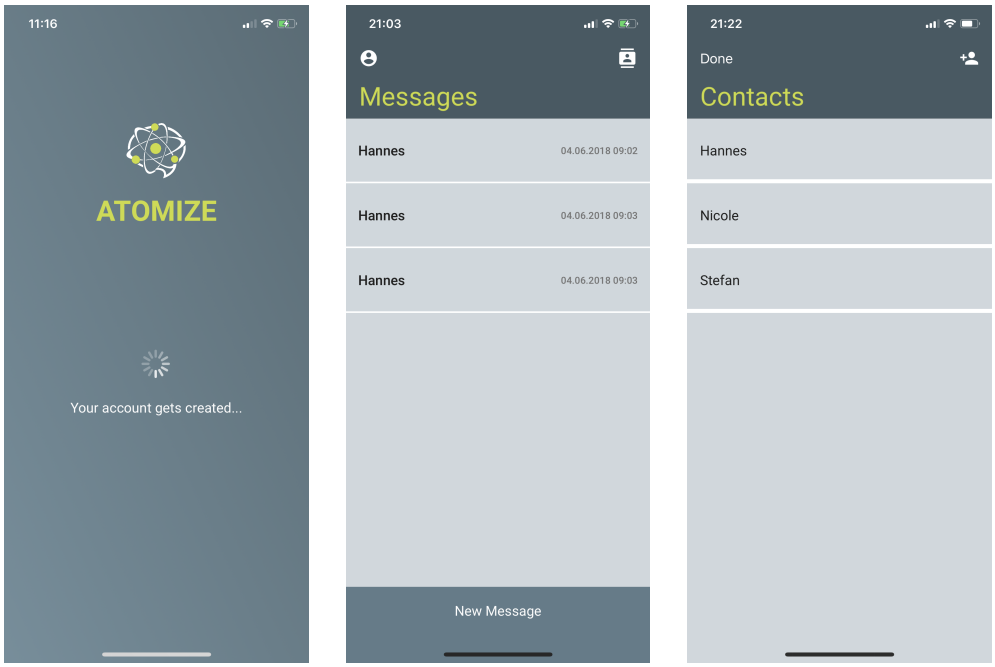
In the *Onboarding module*, the use case is the registration of the user. This module only gets called at the first invoking of the application.

The `OnboardingWireframe` gets called by its `show` method and sets up all relations. Furthermore, its interface provides the `showMessages` method for routing to the *Messages module* when the onboarding is finished.

The `OnboardingViewController` is responsible for displaying the `OnboardingStoryboard`. It shows kind of a loading screen as can be seen in figure 5.5.

The `OnboardingInteractor` has references to all three providers because all of them are needed for the use case of this module. The use case is the registration of the user. Therefore the `CryptoProvider` is used to generate the asymmetric key pair. If this has been successful, a request is sent to the `BackendProvider` for creating a new user account on the server. A response that includes the public id and the authentication token generated on the server is received, and these values get stored as authentication model by the `StorageProvider` in the application database.

The `OnboardingPresenter` listens to an event if the notification center has successfully received a push token. When this is the case, the `OnboardingInteractor` use case for the registration is called. If the registration was successful, the presenter calls the router to switch to the *Messages module*.



(a) (b) (c)

Figure 5.5: Screens of the *Onboarding module* (a), the *Messages module* (b) and the *Contacts module* (c).

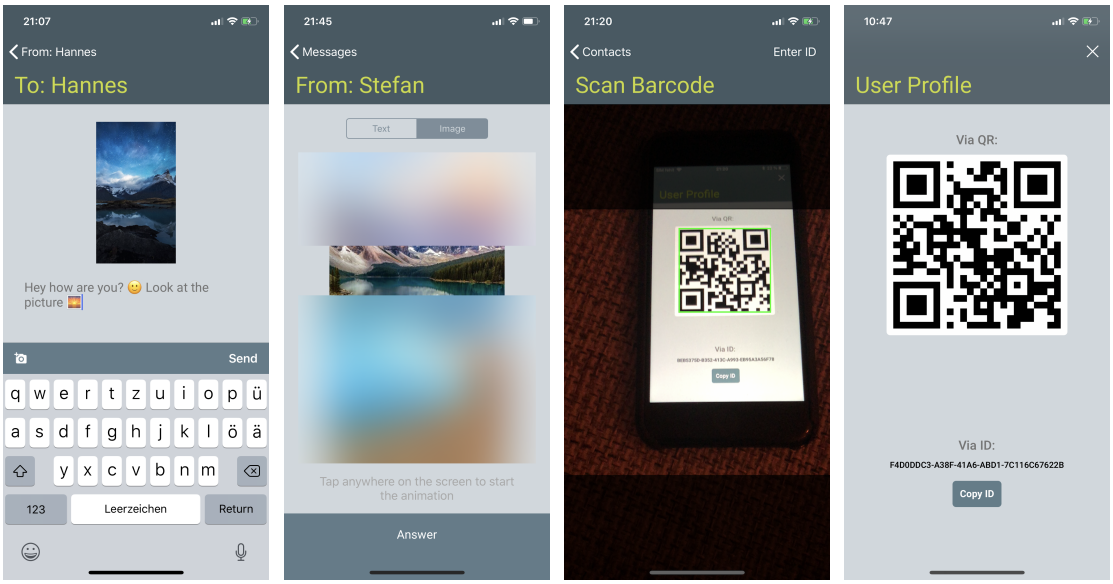
5.2.3 Messages module

When a user has been onboarded, the *Messages module* is the entry point of the application. Its use case is the management of the messages that have been received.

The `MessagesWireframe` gets invoked by its `show` method when the onboarding has finished, as the entry point when the application launches and the user has been registered already or as central navigation view after returning from the other modules to the main screen. The router contains methods for navigating to the *UserProfile module*, the *Contacts module* when either a contact should be added or chosen to send a message too or to the *ReadMessage module* when a message should be opened.

The `MessagesViewController` displays a table for the messages, a menu for navigation and a button for writing a new message, as can be looked up in figure 5.5. When the view is loaded the first time, the user gets asked for permission to access the media library of the smartphone. Furthermore, the controller has implemented all event handler methods for responding to received actions on the UI elements and the methods that get called by the presenter to configure the data that should be displayed.

The `MessagesInteractor` contains the use cases for fetching the messages from the server and for loading the stored messages from the application database. When the messages should be loaded from the server, first the own authentication values get obtained by the application database. Then the `BackendProvider` is used for sending a request to the server and returning the responded messages. These get then stored by the `StorageProvider` in the application database. The other use case is then to load



(a) (b) (c) (d)

Figure 5.6: Screens of the *WriteMessage module* (a), the *ReadMessage module* (b), the *AddContact module* (c) and the *UserProfile module* (d).

the stored messages from the application database and preparing them for the display.

The `MessagesPresenter` contains the event handler methods for triggering the navigation to other modules which are received by view controller actions and for refreshing the messages. Both call the related use case methods of the interactor and pass the received values to the view controller to display them.

5.2.4 Contacts module

The *Contacts module* is responsible for the use case of managing the contact data stored in the application database.

The `ContactsWireframe` can either get invoked with an editing mode or without this option by the `show` method. Depending on which mode has been chosen, the available functionalities differ. Either a contact can be selected for writing a message to or for being edited. The router interface methods provide actions for navigating to the *AddContact module*, the *WriteMessage module* or for returning to the *Messages module*.

The `ContactsViewController` includes a table that displays all stored contacts and a button for adding a new contact, as can be seen in figure 5.5. Depending on the mode that has been chosen, the appropriate event handler gets called when any user action happens on a UI element.

The `ContactsInteractor` contains the `StorageProvider` for the use cases of either loading, deleting or editing the name of a contact.

The `ContactsPresenter` handles the received user inputs for working with the contacts and loads all needed data by calling the interactor. It also calls the wireframe for routing to the other modules when it is required.

5.2.5 WriteMessage module

This module conforms to the use case for writing and sending a new message to a user.

The `WriteMessageWireframe` is invoked by its `show` method when, either the user wants to write a new message or for writing back. It implements the router interface that provides the navigation back to the *Messages module*.

The `WriteMessageViewController` displays an input field, a keyboard and a button to upload files, as can be seen in figure 5.6. It contains the event handlers for reacting on actions like pressing the send button or for the file upload. When an image should be sent, an image picker is used to display the media library selection.

The `WriteMessageInteractor` handles the use case for sending the written message to the receiver. Therefore all three providers are needed. If the contact has already been stored to the application database, it gets loaded with the `StorageProvider`, and the message gets encrypted with the public key by the `CryptoProvider`. Afterward, the `BackendProvider` is used to send the message to the server. For the case that the contact is unknown, one additional step has to be taken before the message gets encrypted. This step is fetching the related public key from the server with the `BackendProvider`.

The `WriteMessagePresenter` is responsible for reacting on the view actions like pressing the button for sending a message and for telling the view which name for the contact should be displayed. Furthermore, it invokes the interactor for sending the message and uses the wireframe to route back when this has been successful.

5.2.6 ReadMessage module

The use case of the *ReadMessage module* is the presentation of the received message in clean text or clear view of the picture.

The `ReadMessageWireframe` gets called by its `show` method when the user presses on a message to read it. The methods implemented by the related router interface are for routing to the *WriteMessage module* if the user wants to write back or navigating back to the *Messages module* if the user dismisses the message.

The `ReadMessageViewController` shows the clear message and image that have been received, a button for writing back and if the contact is unknown, a button for adding as a new contact as is shown in figure 5.6. For all these UI elements the proper event handler actions are implemented and react on specific user input. In addition to this, the animations for screenshot prevention are also built in the view controller and get invoked by pressing the finger on the screen.

The `ReadMessageInteractor` contains three use cases. One for loading and removing the message, another for adding the contact if it is unknown and the last for loading the attachment. For these, all providers get used with their functionalities. The simplest use case is loading the message attachment. For this, a request to the server is sent by the `BackendProvider`, and the encrypted data gets returned for further usage. When a message should be loaded and displayed, first the `StorageProvider` loads the specific contents of the message and the related contact data from the application database. After that, the `CryptoProvider` gets used for the verification process. If everything succeeded, the message gets encrypted. Then it gets checked if an attachment has been sent, it gets loaded and subsequently decrypted by the `CryptoProvider`. If everything worked fine, the message content is stored temporarily in a DTO, and the

`StorageProvider` removes the connected data from the application database. The last use case is adding the contact to the database. For this, the `BackendProvider` is used to fetch the related public key, and then the `StorageProvider` stores the new contact.

The `ReadMessagePresenter` handles specific events that are obtained by either the view controller or the interactor. In particular, these include reacting on the view actions for storing the contact or for writing back. Also, the part for requesting the message content from the interactor for delegating the data back to the view controller is included.

5.2.7 AddContact module

The *AddContact module* provides the use case of adding a new contact to the application.

The `AddContactWireframe` gets called by its `show` method and sets up all relations. The `AddContactRouter` contains the method for dismissing the action of adding a contact and returning to the previous module.

The `AddContactViewController` opens the QR-code scanner. It includes a menu with an option for navigating back and an option for adding the new contact as shown in figure 5.6. It is possible to add a contact by entering the id or by scanning the QR-code. Therefore the functionality for scanning a QR-code and passing the recognized id to the presenter is implemented. Also, some methods for displaying error messages or requesting a name for the added contact are provided.

The `AddContactInteractor` has references to the `StorageProvider` and `BackendProvider` and implements the use case to store a new contact in the application database. When the related id of a contact has been successfully scanned, the `BackendProvider` is used to load the connected public key. When the response values are received, the `StorageProvider` saves the collected contact data into the database.

The `AddContactPresenter` has relations to all other components and implements the event handler methods that get called, when either the view controller has scanned the id or when a name should be added to the new contact which has been entered in the related view element. Furthermore, it takes care of storing the new user by invoking the interactor.

5.2.8 UserProfile module

The last module facilitates the display of the own user profile that includes the QR-code and the id for exchanging the contact data.

The `UserProfileWireframe` gets called by its show method when the user presses the related button in the *Messages module* and sets up all relations. The router provides the action for navigating back to the *Messages module* if the user wants to dismiss it.

The UserProfileViewController displays the QR-code that is generated from the id and additionally, the id in plain text with an accordingly button for copying it, as can be seen in figure 5.6. It includes the actions for pressing the button for returning to the *Messages module* and for copying the id to the clipboard. Furthermore, it delegates all needed requests for display to the presenter.

The `UserProfileInteractor` confirms to the use case for loading the own contact data to further processing it. Therefore it keeps a reference to the `StorageProvider`. It gets used when the data should get loaded from the application database.

The `UserProfilePresenter` is responsible for requesting the own contact data from the interactor and passing it on to the view controller for display. It also receives actions from the view for routing back to the *Messages module*.

5.2.9 Application entry point

The main entry point for every iOS application is called the `AppDelegate`.[8] In the messaging application, it is used for setting some general configurations like the navigation bar and window appearance. Furthermore, it gets checked, if the application launches for the first time or if it has already been opened. This is observed by using the `CryptoProvider` for verifying the asymmetric key pair. When it exists, the onboarding has happened before, and the related user credentials can get loaded from the application database. If there is no key pair, it still has to be created, and the user registration has to be performed. Depending on these facts, either the *Onboarding module* or the *Messages module* gets invoked by applying their view controller to the root view controller, through a call to their wireframes show method. The other responsibility of the `AppDelegate` in the application is the registration for push notifications and the setup of the notification center for handling incoming notifications. Due to the reason, that the user has to give permission allowing the application to send these notifications, the configuration part should happen at the beginning of the application flow.

5.2.10 Crypto provider

The `CryptoProvider` is responsible for managing the asymmetric key pair, the encryption/decryption and verifying/signing of message data. To accomplish all these functionalities, Apple's Security Framework[9] has been used to develop all the necessary features. Since the keys get stored in the secure enclave of the smartphone, the used algorithm and key size had to be adapted to fit the conditions of which kind of private key is allowed to be saved. These guidelines are defined by Apple and must be met to ensure that the keys are correctly stored in the secure enclave, though some restrictions get explained in the following lines.

The secure enclave stores only 256-bit elliptic curve private keys. Therefore the encryption algorithm had to be an ECC algorithm. The exact algorithm used is based on the ECIES (Elliptic Curve Integrated Encryption Scheme) scheme which uses a hybrid encryption procedure. This means that it combines an asymmetric method for sending a key, with a symmetric process which then operates the encryption of the messages. More specific, this implies that an asymmetric key pair gets generated, the private key is then stored in the secure enclave, and the public key gets always created by it on later time points. When a communication exchange with another party should begin, a symmetric key used for sending messages between users is constructed and gets encrypted with the public key of the other communication entity. Thus only the creator of the symmetric key and the holder of the private key that can decrypt the symmetric key and the data that has been encrypted with it.

[8] The `AppDelegate` contains methods that respond to certain lifetime events of the application and are called by the singleton `UIApplication` that manages the application [22].

[9] The Security framework provides functionality to secure the data and access of an application [9].

In specific, the symmetric AES algorithm with a key size of 256-bit is implemented. For the signing of the messages, an appropriate algorithm that also uses the ECC scheme and matches the encryption algorithm had to be found. Therefore the ECDSA (Elliptic Curve Digital Signature Algorithm) has been chosen. Now that the fundamental topics have been explained, the implementation and functionality of the `CryptoProvider` get described briefly. An overview of these methods is shown in table 5.3.

`loadKeyPair`: When this method gets called, an attempt to load the private key from the secure enclave is executed. If the key exists, the related public key gets generated from it and returned. If there is no key saved, it implies that this is the first launch of the application and therefore the key pair has to be generated. This is done by creating a private key, storing it in the secure enclave and again generating and returning the related public key from it. Here is an example of how such a generated public key looks like: `BLJ17WvMRvSQsZOKL+URXZXbKQIqygXsU9zUOko7aFrfYHhJbYQi8LQakI7wUpzo9NZ5Lg1UqNxpfg8nZM6/6I=.`

`encrypt`: The encryption of messages works by applying the used algorithm, in combination with the public key of the intended receiver, on the text or image that has been transformed into a data hash before. This generates a cipher that can be reused in the further process. This is how a part of such a cipher looks like: `BFf31B8FbttcNAzv8pQQcxC cCNjexKhKhXmD3kIFzJA7tgvjcvl3WO4tZYSilYlV13T/UIkSX9jfqKybx4gGO0EIvSojiyik+/HOea4otfvA==.`

`decrypt`: The decryption process operates the other way round. First, the private key gets loaded from the secure enclave. Then the received cipher gets decrypted by applying the same algorithm used for encryption, in combination with the private key on it. This procedure leads to a data hash can either be transformed into text or the image that has been received.

`sign`: When a message is signed, the operation is transposed on the already encrypted cipher. As an initial step, the private key is loaded from the secure enclave and gets then applied by the signing algorithm on the cipher. By this, the signature has been created and can be further processed.

`verify`: For the verification, to prove that a message has not been altered on the communication exchange, the mechanism is to compare the encrypted message hash with the signature hash by utilizing the signing algorithm in combination with the public key of the sender on both data values. If the results are equal, the verification process has been successful, and the sender has been confirmed in its identity confidentiality.

5.2.11 Backend provider

For enabling the communication between the application and the server, the `Backend-Provider` has been implemented. It provides all functionalities for performing requests to the server and handling the incoming responses. For enabling the exchange via HTTP, some data models that can be transformed into JSON and back have been implemented. In table 5.4 an overview of all the methods and related data models is displayed, and they get outlined hereafter.

Method/Action	*Parameters*	*Return value*
`loadKeyPair`	-	public key
`encrypt`	message, key	encrypted message
`decrypt`	decrypted message	encrypted message
`sign`	message cipher	signature
`verify`	encrypted message, signature, sender key	success

Table 5.3: Overview of the `CryptoProvider` methods.

`createAccount`: When an account is created, this method sends the needed input values for registration and handles the result values from the response. The `BackendRegisterInputValue` model includes the device info, the push token and the public key that should get stored on the database. In return, the public id and authentication token values are received and get further processed by mapping them into the `BackendRegisterResultValue` model.

`sendMessage`: For sending a message to a user, the `BackendSendMessageInputValues` model is used for handling the content, the attachment, the signature, the sender id and the receiver id. Additionally, the authentication token gets sent with the request to allow access to the server functionality. Otherwise, the authentication would fail and the message would not arrive. If everything worked fine, a response transformed into the `BackendSendMessageResultValues` model and which includes a success message gets received.

`getMessages`: Every time, when a refresh for loading the new messages is triggered in the application, this method gets called. A request with the own public id and authentication token gets sent to the server, and the incoming response is mapped into an array containing the `BackendMessagesResultValues` for each message. These consist of the content, the attachment, the signature, the sender id and the date sent.

`getPublicKey`: When a new contact should be added to the messaging application, a request to the server has to be executed for loading the related contact data. The input values consist of the public id of the new contact, the own public id and the authentication token for enabling access to the action on the server. In return, the value for the public key arrives and is mapped to the `BackendPublicKeyResultValue` for being able to further processing it.

`getMessageAttachment`: If a message includes an attachment, an extra request for loading the data from the server is implemented by this method. By sending the filename and the authentication token in the request, the related data gets loaded from the server and returned as response mapped into the `BackendMessageAttachmentResultValue`.

Method/Action	***Data models***	
	Input values	*Result values*
`createAccount`	device info, push token, public key	public id, authentication token
`sendMessage`	content, attachment, signature, sender id, receiver id	success
`getMessages`	public id, authentication token	content, attachment, signature, sender id, date sent
`getPublicKey`	receiver id, sender id, authentication token	public key
`getMessageAttachment`	filename, authentication token	file data

Table 5.4: Overview of the `BackendProvider` methods.

5.2.12 Storage provider

To assure anonymity, it was of high importance that all contact data (and additional data) gets saved only on the smartphone. Therefore a database had to be implemented for managing the application storage. For this Realm[10] has been used as the framework. In the `StorageProvider`, all functionalities for managing the database are implemented. These include methods for storing, retrieving and deleting the data. In addition to this, several data models that are necessary for enabling the transformation of the data to be exchangeable with the database have been included. In the following, each model gets illustrated briefly.

`StorageAuthenticationModel:` This model consists of the public id and an authentication token that both get returned after registration from the server.

`StorageMessageModel:` It includes the content, the attachment, the signature, the sender id and the date sent and is used to save all messages.

`StorageContactModel:` The contact is stored by using this model and includes the information of the public id, the public key and a name that gets chosen for the user.

[10]The Realm platform provides two parts, a database and an object server for developing mobile applications [16].

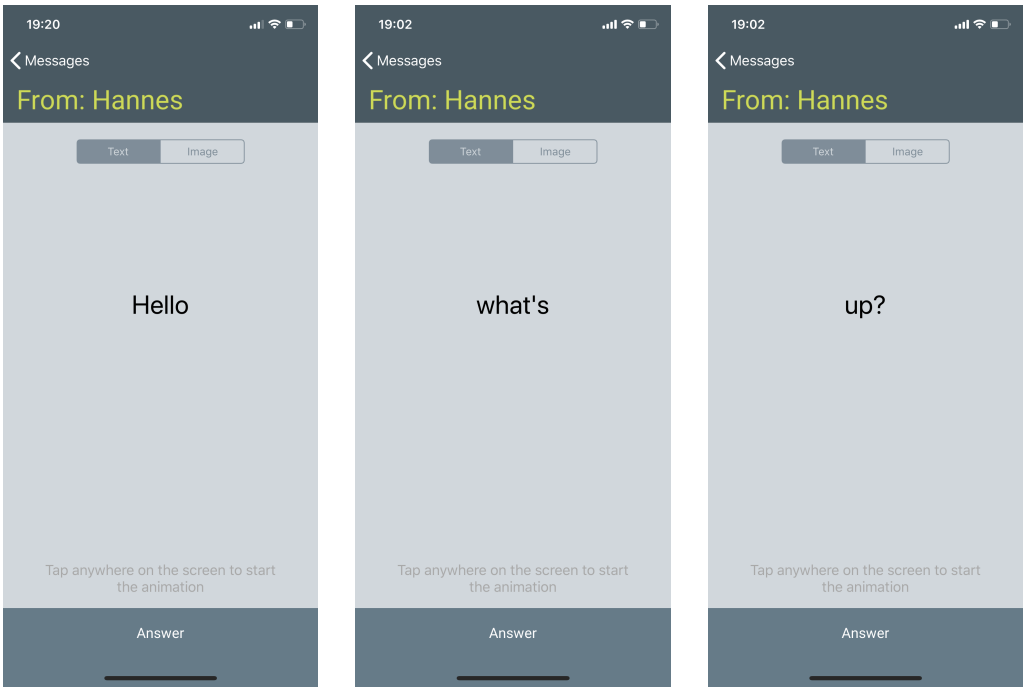


Figure 5.7: The text animation for the screenshot prevention that has been implemented in the *ReadMessage module* as described in section 5.2.15.

5.2.13 Push notifications

Every messaging application should provide some way of sending and receiving push notifications to inform the user about received messages, even if the application is closed or running in the background. To achieve this, the APNs-service of Apple has been used to implement the needed functionalities. Therefore a unique certificate for the application had to be generated and added. After that, the push notification service had to be enabled and implemented in the project. The registration for push notifications is separated into two steps. First, the user permission must be obtained, and if that was successful, a push token gets automatically generated from the system as an address to send messages to. In addition to this, all functionalities for handling the incoming notifications have been implemented, as already got described in section 5.2.9.

5.2.14 QR-code scanner

Another essential functionality is the QR-code scanner implementation which is used by the *AddContact module*. This has been realized by using Apple's AVFoundation[11] framework. Therefore a video capturing with the back camera of the device gets started, and by scanning the QR-code and transforming the captured metadata into a barcode object and furthermore into a string, the QR-code can be read out.

[11] The AVFoundation framework provides functionality for working with time-based and audiovisual media [8].

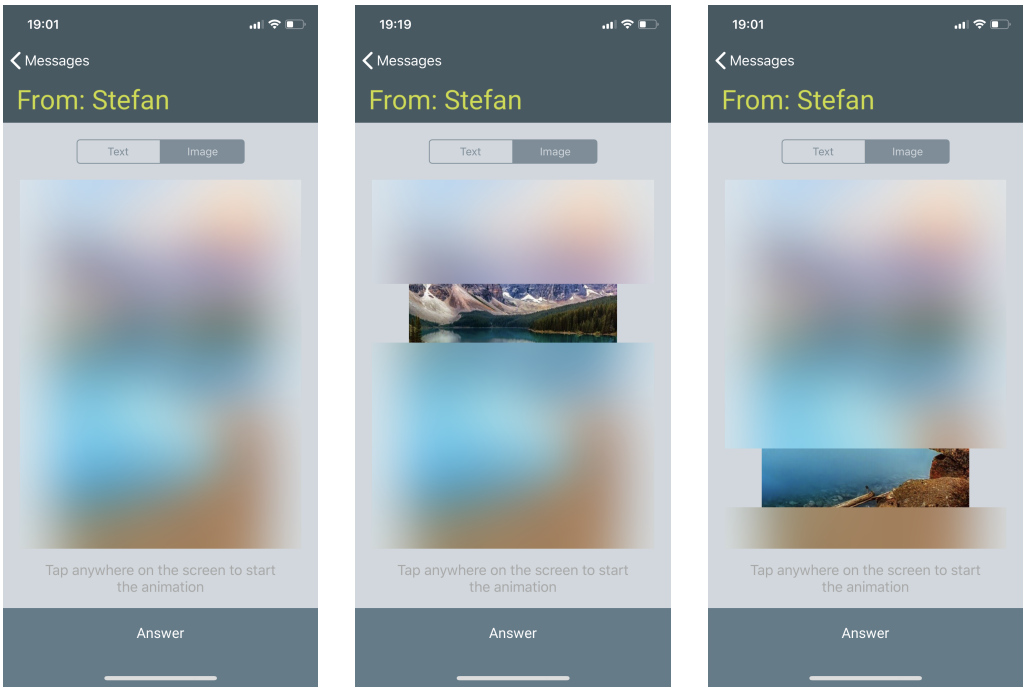


Figure 5.8: The image animation for the screenshot prevention that has been implemented in the *ReadMessage module* as described in section 5.2.15.

5.2.15 Animations

For the prevention of message persistence, the methods implemented are built upon animations. There are two variants which are adopted either for the text or the image sent in a message. Both of them are implemented by the *ReadMessage module* and are part of the `ReadMessageViewController` display. The animation of the text repeatedly visualizes one word after another and can be controlled by touching the screen. If the finger leaves the screen, the animation stops. Thus the information of the message can be assimilated in a natural reading flow, but only one word gets visible if a screenshot is taken, as is displayed in figure 5.7.

A quite similar approach has been implemented for the images sent. By using blurred layers and animating them over the whole picture, it gets ensured that always only a small stripe of the picture is visible. An example of taken screenshots is shown in figure 5.8. Again the animation can be controlled, by touching the screen it starts moving the blurred parts, and by leaving the screen, it stops. Therefore the possibility is given to observe also details of the transparent part of the image. The blurred layers are transferred at such a speed that the display of the picture keeps recognizable and comfortable for the user to view. Additionally, the animation is stoppable by pressing with the finger on the photograph and continuing it by re-releasing the screen. Thus the possibility of observing parts of the picture in more detail is established.

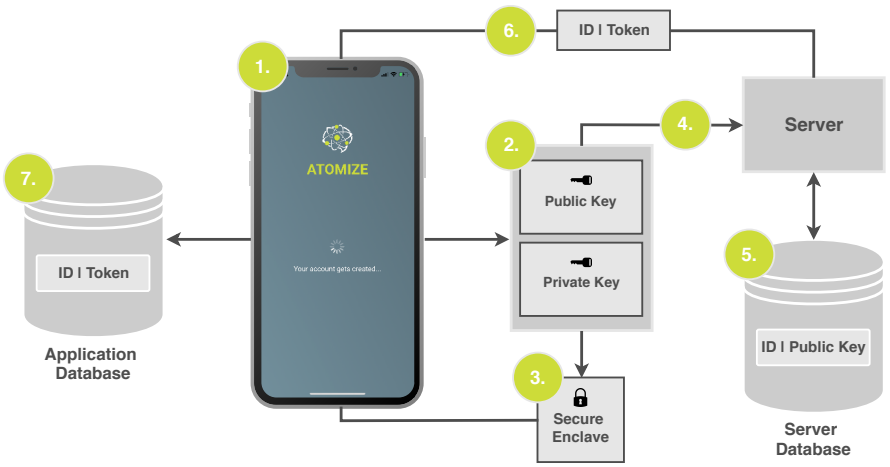


Figure 5.9: The *Onboarding module* application flow as described in section 5.3.1.

5.3 Application flow

In this section, the application flow for each module gets described and illustrated. Each step taken is demonstrated and gets explained so that the exact procedure can be comprehended. The *UserProfile module* and *Contacts module* are not mentioned, because of their simplicity and self-explanation.

5.3.1 Onboarding module

The application flow for the *Onboarding module* gets described hereafter and is displayed in figure 5.9.

1. The view of the *Onboarding module* gets displayed and shows a loading symbol to demonstrate the account creation.
2. Meanwhile, in the background, the asymmetric key pair gets generated.
3. After that, the private key gets saved to the secure enclave of the application.
4. The public key then gets sent via a request to the backend server.
5. On the server, a new user and an additional authentication token get generated and saved to the database. The automatic created primary key of the user serves from now on as public id.
6. The public id with the related authentication token gets sent back via response to the application.
7. After receiving the account data, it gets saved to the application database.

5.3.2 Messages module

In figure 5.10 the *Messages module* flow can be looked up and is structured as follows.

1. After the registration or as the main entry point, when the user has already been boarded, the *Messages module* gets displayed.
2. A request for fetching the messages gets sent to the server.

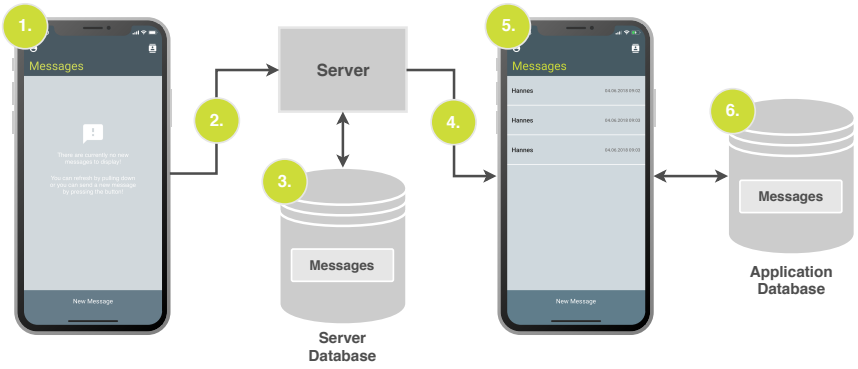


Figure 5.10: The *Messages module* application flow as described in section 5.3.2.

3. On the server, all messages referring to the user get loaded from the database.
4. All messages get sent back via response to the application and get immediately deleted from the server database.
5. The fetched messages get displayed in the table ordered by the date sent.
6. To persist the messages for the moment, they get saved to the local database.

Additionally, the user can pull down in the overview to send a request for fetching the messages manually. However, it is also possible to receive push notifications when a new message arrives. This happens when the application runs in the background, when it is closed or when the user has opened the *Messages module*. In each of these cases, the messages get updated automatically.

5.3.3 WriteMessage module

The *WriteMessage module* flow describes the process when a message is written and is displayed in figure 5.11. Each process step gets described hereafter.

1. First, a contact has to be chosen for sending the message to. For this purpose, the *Contacts module* gets displayed. There is also the option to add a new contact.
2. When a contact is selected, the related data gets loaded from the local database.
3. Then the *WriteMessage module* gets displayed, and the contact data get prepared for it. For the case that a user writes back on a received message, this is the entry point in the module flow, and step 1 and 2 get skipped.
4. The user can input a message and/or upload a file that should be sent.
5. In the background, the message gets encrypted with the public key of the receiver. In addition to this, a signature of the encrypted message gets also generated. The same process is executed for the message attachment.
6. Then a request with the whole message content gets sent to the server.
7. On the server, the message gets created and saved to the related user. If a file has been sent with the request, it gets stored in a folder on the server.
8. Subsequently, a push-notification gets sent by the server to the message receiver.

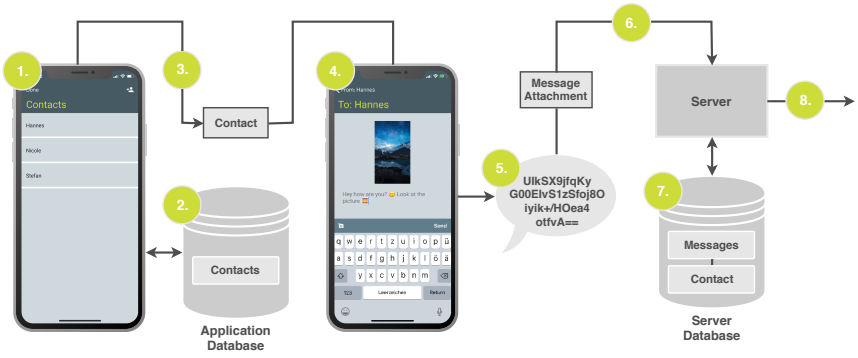


Figure 5.11: The *WriteMessage module* application flow as described in section 5.3.3.

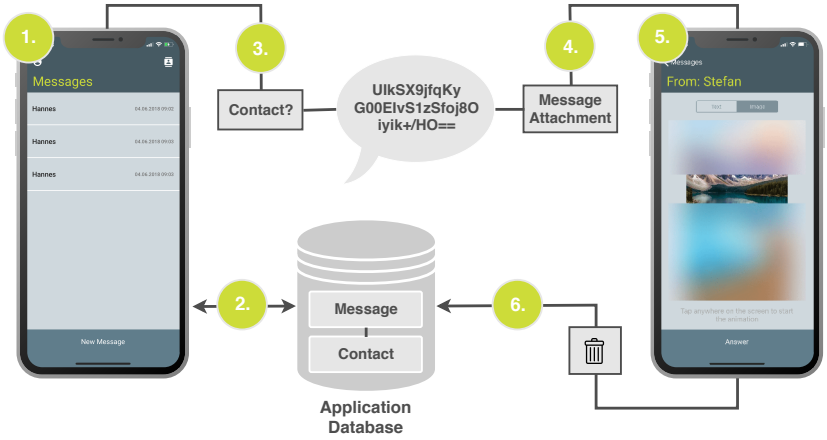


Figure 5.12: The *ReadMessage module* application flow as described in section 5.3.4.

5.3.4 ReadMessage module

The application flow for reading a message is implemented as shown in the figure 5.12 and is described in the following lines.

1. In the *Messages module*, all messages are displayed in a table.
2. They are loaded from the local storage of the application.
3. If a related contact exists, it gets passed further to the *ReadMessage module*.
4. Then the message and attachment get decrypted.
5. After that, the decrypted message and attachment get displayed in the view, and the animations for the screenshot prevention are applied.
6. The last step is the deletion of the message from the application database.

5.3.5 AddContact module

When a new contact is added, the application flow works as following and can be looked up in figure 5.13.

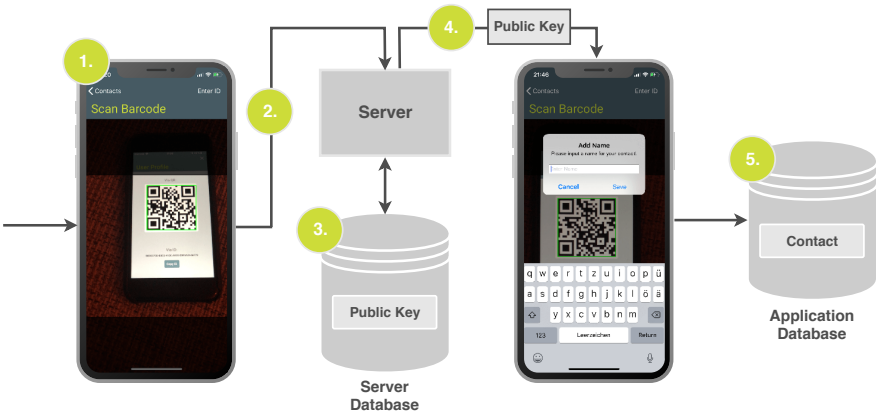


Figure 5.13: The *AddContact module* application flow as described in section 5.3.5.

1. When the module for adding a new contact has been invoked, the barcode-scanner gets displayed and additional, a navigation bar including a button for the option to manually enter an id is shown.
2. After scanning the QR-code or entering the id, a request with the id for the requested contact gets sent to the server.
3. The contact data referring to the public id gets loaded from the database.
4. The public key gets sent back in response to the application.
5. Finally, a new contact gets created with the key and id. Additionally, an username can be specified and then all data gets saved to the application database.

5.4 Reflection on design decisions

In this section, the design decisions taken in the implementation process get described, and the decision-making gets explained. It should get clear why the several parts have been developed in this kind of way. For this, the backend and frontend are again split up and declared hereafter, but first, the choices that are related to both implementation parts get clarified.

5.4.1 General

Hereafter all decisions that influence both implementation parts get described briefly by their aspects that are interesting to comprehend.

Message transience

By message transience, the procedure of deleting the messages after they have been read is meant. There exist two execution tasks lead to the deletion of the messages. One handles the elimination of all received messages from the server database after each request from the application. The other task removes a single message from the application database if it has been opened for reading it.

The reason why the messages get entirely removed from the server after every fetch is to guarantee that the data will persist just as long as necessary on a remote place. By storing them subsequently in the application database, they still get persisted, but only on the own trusted local device. Therefore the threat of other parties gathering the messages is decreased to a minimum level. In addition to this, the second task of deleting a message after it has been opened is included. It is implemented for the case to prevent any other person from still figuring out the chat history or with whom messages have been exchanged if the smartphone falls into the wrong hands.

File handling

Another interesting decision is, what file types and how they are handled in the whole application flow. The file type for the attachments sent with each message has been restricted to send only images. Theoretically, it is possible to send any other type of file, but this choice is related to the additional persistence threats that can occur. For this application, the focus lied on finding techniques for the prevention of screenshots or pictures taken with another device. Video or audio files can hypothetically always get persisted by filming or recording them via a third device or screen recording technique. It is also challenging to find an option for these files to always show only a small part because they have to be viewed as a whole. Other types like text files or PDFs would also have the requirement to open and scroll throughout them. Here an animation, which always blurs much of the text, would be experienced as annoying by the user. Therefore the file types have been limited for sending only images.

5.4.2 Backend

For the backend implementation, the following parts include some interesting choices that have been made for the overall application flow.

Model design

In this part, the design of the several data models for the server database should be explained shortly. Each model is listed and declared in its structure hereafter.

User: As displayed in figure 5.1, a user consists of the id, the device info, the public key and the push token. These pieces of information are all stored together in one document, so that contacts can easily be added by retrieving them through their id. The push token is included for enabling the server to quickly gather the token and then send notifications to the specific device. Although it is theoretically possible to use the id in combination with the key, to furthermore encrypt and send messages to a specific user, the real person behind it stays anonymous and can not be investigated.

Token: The model for a token includes an id of the related user model and the token used for the authentication. In figure 5.2 this structure is presented. These models can not be accessed or manipulated and are entirely served for authentication purposes.

Message: A message contains the text content, the attachment, the signature, the receiver id, the sender id and the date sent, as shown in figure 5.3. The attachment includes the name of the related file stored on the server. To avoid the traceability of files, the real name gets exchanged with a randomly generated id, so it is kept more secret. The receiver id gets saved to be able to associate to whom the messages belong. The sender id is needed for the verification process to allow loading the related public key for verifying the signature from the application database.

Token authentication

Another interesting point is how the authentication on the server has been implemented and why it is kept so simple. The primary demand is to prevent any other entity from spamming the server or getting access to specific functionalities for manipulating the data. Although it is a basic token authentication, it provides the advantages that no further login data is required, which is a primary requirement for keeping the anonymity of the users. In addition to this, there is no additional effort for the user to authenticate because everything works seamlessly in the background.

Storing files

When image file gets uploaded as attachment of a message, it gets stored in the `public/-files` folder. Only the receiver of the message has the information which file is meant because the file becomes a new random id as a name when it is uploaded. For the reason that each file is any way encrypted before it is sent to the server, its data is protected from external accesses. Even if somebody would gather the files, it would never be possible to convert them to the real content.

5.4.3 Frontend

Hereafter the taken decisions for the frontend application implementation get explained.

VIPER architecture

The VIPER architecture has been chosen for several reasons. A significant advantage is the separation of the use cases into their modules. Therefore the implementation has its clear arrangement and is easily extendable. Also, a big part of the code is reusable by this structuring. Another benefit is that the code is highly testable and error handling is kept simple. Thus locating an error is done fast and debugging is accomplished with less effort than in other application architectures.

Encryption methods

The finally used encryption method depends on the decision that the private key should be stored in the secure enclave. Therefore, as described in section 5.2.10, only 256-bit elliptic-curve keys are allowed to be used, which is a guideline from the Apple Security framework. Furthermore, this also implied that the algorithm used, supports a hybrid encryption scheme and combines the advantages of both the symmetric and asymmetric methods. Also associated with this topic was the establishment of higher security for

the file management. Therefore each file is encrypted and then transformed into plain text. On the server, every attachment is stored in a text file including the encrypted file data. Thus no relations or options to access the original data are given.

Contact exchange

For the contact exchange, it turned out very fast that two separate implementation options have to be provided. Therefore the QR-code scanner has been chosen for the local variant, because of its simplicity. The user does not have high effort to add a new contact. Only the QR-code in the user profile has to be scanned with the camera, and then the new contact gets generated from it. Also, the generation of the QR-code was possible to implement very uncomplicatedly. For the remote variant, it was the most straightforward decision to give the option to show and copy the own contact id and exchanging it via a third communication channel. By using these two methods, the swap of the contact data stays secure and is kept very simple. Every additional data needed can be fetched via the id from the server.

Screenshot prevention

For the animations used for screenshot prevention, the decision making has been made from the idea that always only a small part of the text or image should be able to be persisted. By this, the rest of the data is not recognizable. Therefore both animations are based on the principle always to show only a tiny hint of the content. By animating in the correct speed, the user should still be able to recognize the full meaning of the data, and it should also be comfortable to watch. Also, the option to control the animation helps to get a clear understanding of the message content. The advantage of this is that the user can recognize everything, but it is not possible to persist the whole content at once by taking a screenshot.

Chapter 6

Evaluation

In this chapter, the developed methods for implementing higher privacy and data security of the messaging application—called Atomize—get compared with an already existing application, which is the Confide messenger that already has been illustrated shortly in section 3.2. Both get evaluated based on the terms related to privacy and security in mobile communication and have been defined in section 4.1. First, the overall evaluation conditions get determined. Then the comparison is drawn, and the further outcome and findings get discussed. After that, the limitations for private and secure messaging applications and the possible improvements that arose during the development of the thesis get characterized.

6.1 Evaluation conditions

Before the actual comparison can be drawn, the conditions for testing and evaluating the applications get discussed beforehand.

6.1.1 General

By the terms that have been worked out in section 4.1, the topics for the comparison can be determined as the following:

- Traceability of users,
- Data security,
- Exchange of contact data,
- Data transience,
- Prevention of persistence threats.

All of them are important for increasing higher privacy and security in messaging applications. By traceability of users, all aspects regarding the identification of the real person are meant. A central goal was to preserve the anonymity of a person as far as possible. The term data security is also related to the traceability and all methods that keep the data secure, e.g., the encryption of messages. The exchange of contact data includes the techniques used for transferring the information between the application endpoints. Data transience indicates the usage of ephemeral messages and how this is

Test specifications	
Operating System	iOS 11
Device(s)	iPhone X, iPhone 7, iPhone 6
App version	*Standard* (Confide), *Standard* (Atomize)
Technical requirements	Apple A7 or later A-series processor with secure enclave, Back camera

Table 6.1: Overview of the general test specifications.

carried out. Finally, the prevention of persistence threats represents all functions that prevent users from storing the message content.

In addition to this, the overall specification values for the evaluation get declared hereafter. In table 6.1 an overview of these terms is displayed. The name of the developed application has been decided to be Atomize. The operating system supported is iOS, and the lowest compatible version is version 10. For the testing, the applications an iPhone X, iPhone 7 and iPhone 6 were used. The technical requirements for the smartphones were that the device contains a secure enclave, which is a hardware feature of Apple A7 or later A-series processor, and of course, a back camera.

Confide has been tested by the aspects of its *Standard* version. There also exists a *Plus* version where further functionalities are given, but the users have to pay for it. For this evaluation, the free version has been chosen for the comparison, but also the features of the extended version will be mentioned briefly. All these just named terms should provide a better understandability and clarity when the specific applications get reviewed in their characteristics.

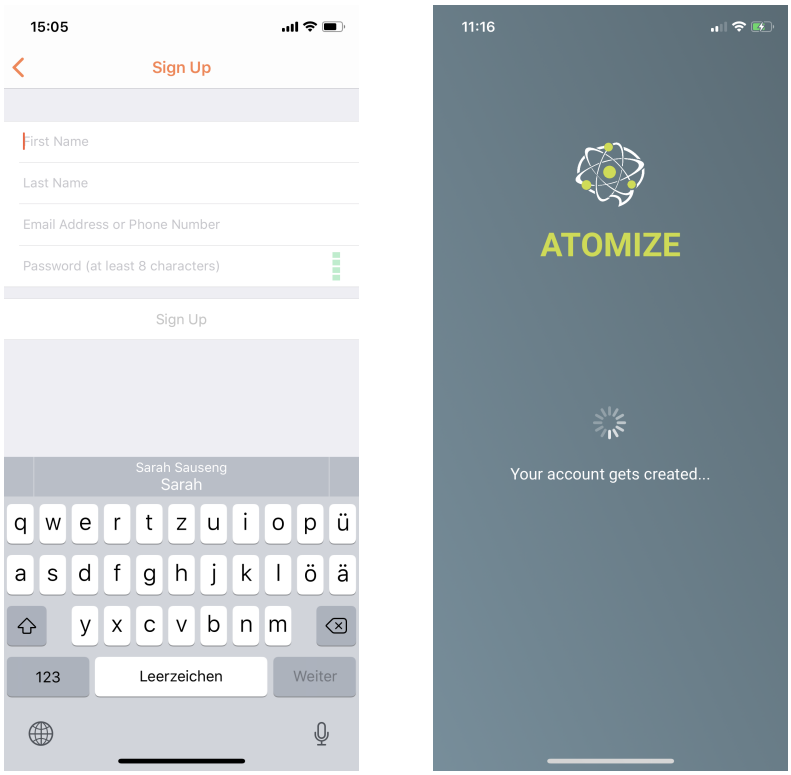
6.1.2 Black-box testing

The method used for the evaluation is called black-box testing. With this approach, the functionality of an application is examined without peering into its internal structure. By defining test cases built around specific specifications and requirements, the software is evaluated. These cases are primarily functional and can be derived from descriptions of the application, including the specifications and design parameters.

For the reason that there exists no possibility to access the internal application code of Confide, the comparison and evaluation will be done by black-box testing. The test cases applied are related to the before mentioned topics for the comparison.

6.2 Comparison

In this section, the comparison of both applications is accomplished. The test environment terms are based on the evaluation conditions that have been specified previously in section 6.1. The evaluation is based on the requirements for building a secure and private messaging application in sections 4.1 and 4.2. After analyzing each application in its functionalities, the several specifications get discussed and evaluated.



(a) (b)

Figure 6.1: Comparison of the onboarding in the applications, the Confide onboarding—Enter personal data (a) and Atomize onboarding—No personal data is required (b).

6.2.1 Traceability of users

Regarding the traceability of users, the components are tested to conform to the privacy and anonymity of a person in the application.

Confide

For the registration, the user has to provide the following personal data: first name, last name, email or mobile number and a password for authentication. This can be observed in the onboarding part of the application, as shown in figure 6.1. Furthermore, in the privacy policy, it is stated that this personal information is collected to identify a person and gets retained over a longer time-period. Therefore this information is also used for finding users in the application and for the sender identification of a message. Moreover, it is also indicated that the data is processed only for the application functionality, additional statistics and is protected by the terms of the privacy policy. More about this can be looked up at [23] in “Retention of personal information”.

Atomize

As described in section 5.2.2, where the onboarding implementation of the application is described, the user has not to share any personal data for the registration. The automatically generated id is the only reference to a user and is stored on the server and the smartphone. In addition to this, the id gets also used for the sender identification in the messaging process. The onboarding screen of Atomize is shown in figure 6.1.

6.2.2 Data security

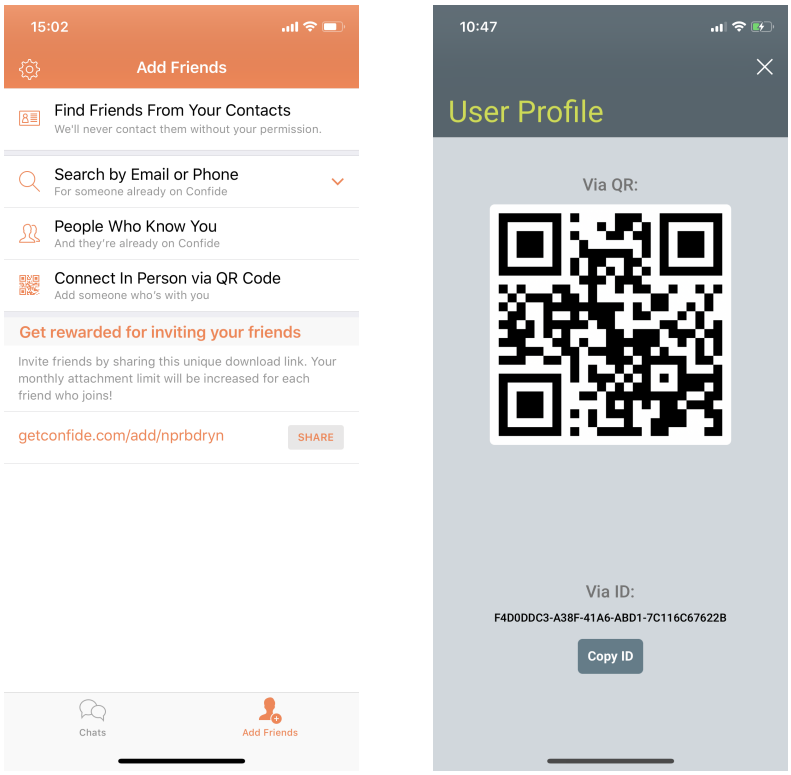
The topic of data security is related to the before analyzed traceability of users, but now get reviewed by their safety aspects.

Confide

In Confide the communication process with another user is end-to-end encrypted. Here also asymmetric encryption is applied, the key-pair is generated on the device, and the private key stays local in the smartphone storage. The data stored and collected by the application contains the following parts: contact information, address book information, message information and billing information. For the contact information, the previously described personal data is saved. The address book information stores the contacts in the address book on the device. The user is asked for permission to give access to this data. If this is agreed on, Confide stores the information in an anonymized/hashed form on their servers. The message information contains the encrypted message content, log and contact data and additional information. The last data about the billing information is only needed when the user wants to upgrade to the *Plus* version. It contains general billing info, which is furthermore processed by a third-party service. In addition to this, it gets declared that third-party services get used and collect data of the device and network specifications. Again it is stated that this information is anonymized. Regarding the authentication security of the application, the functionalities and access are secured by the login data that has been provided at the registration [23].

Atomize

The developed application aimed to keep the user anonymous as far as possible, but still keep the data secure from unauthorized access. Thus the messaging process is also asymmetric end-to-end encrypted, as explained in section 5.2.10. The data stored on the server consists of the messages, the users and the related token for authentication. There is no personal information used, and the messages are stored encrypted. When an attachment is sent with a message, it gets saved in the public/files folder on the server. However, this data is a pure text file and consists only of the encrypted image data. Contact data is only stored locally on the device in the application database. For the authentication process, a simple bearer token authentication is established to protect the access to specific server functionalities. In section 5.1.3 this part is described in more detail.



(a) (b)

Figure 6.2: Comparison of the contact exchange in the applications, the Confide contact exchange (a) and the Atomize contact exchange (b).

6.2.3 Exchange of contact data

This evaluation topic refers to which methods are provided for exchanging the contact data, how they are implemented and how the access to the other contacts is handled.

Confide

There exist three different options for adding a new contact in Confide. An overview of them is shown in figure 6.2. When access to the local contact book of the device is granted, it can either be searched for specific persons to add them or for already registered contacts in Confide. Additionally, in the settings of the application, it can be activated to connect with Facebook and allow other users to find oneself through this service. Another option is to search by email or mobile number for a specific person. The last alternative is to add a contact locally, which also is implemented by using a generated QR-code and barcode-scanner.

Atomize

In Atomize, two options are implemented for the contact exchange as described in the application flow in section 5.3.5. The local variant is the QR-code scanner that can be

used within the application. The remote option is copying the id to the clipboard and then exchanging it via a remote channel.

6.2.4 Data transience

Concerning the data transience, each application is evaluated by its functionality for sending ephemeral messages and additional provided options for keeping the data transient.

Confide

Confide also provides higher privacy by establishing ephemeral messages. In the application, every message gets deleted from the server after the receiver has read it. If a message stays unread for at least seven days, it gets removed automatically from the server. In addition to this, there is a default setting in the application that keeps the chat history of the own sent messages stored locally on the device. Therefore they can be read afterwards. This option only works for text messages, and the history is also removed automatically when the user receives and reads a new message or after 48 hours. Additionally, this option can also be deactivated in the configuration settings. In the *Plus* version, there exists the extra option to retract a message, if the receiver has not read it.

Atomize

As described in section 5.3.2, the messages get removed from the server after each request for fetching the messages and get stored locally on the device. Then, as illustrated in section 5.3.4, when a message is opened, its content gets loaded, and it gets immediately removed from the application database. On the server, there is an extra cron job implemented that removes messages from the remote database, if they have not been fetched in the last seven days.

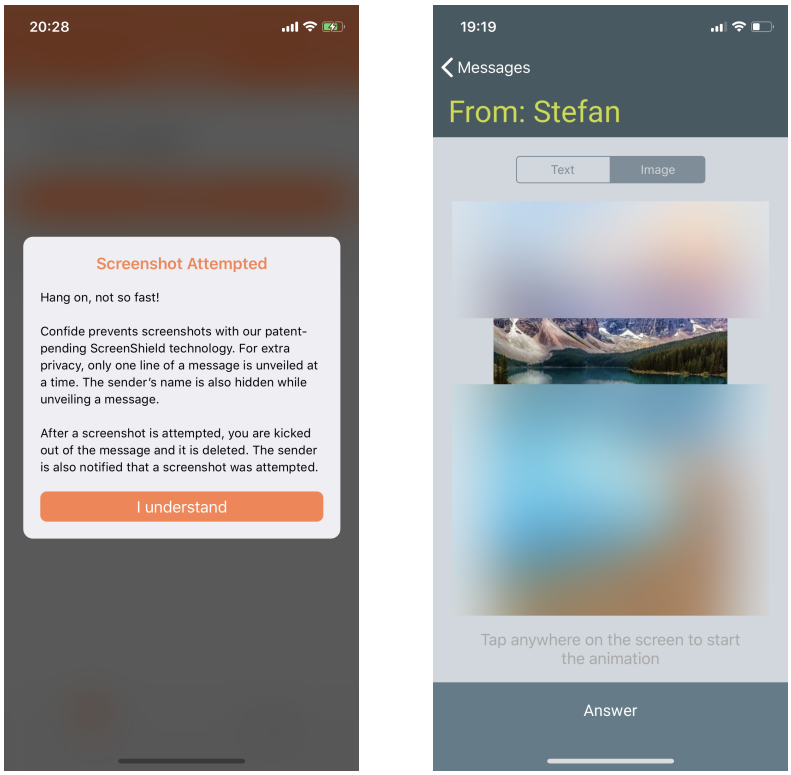
6.2.5 Prevention of persistence threats

Hereafter the methods implemented by the several applications that correlate to the prevention of additional persistence threats are evaluated.

Confide

For the screenshot detection, Confide developed an own Screenshield Kit[1] technology. Its purpose is to recognize if a screenshot is taken and then manipulating the picture with image processing to cover the sensitive parts. In addition to this, the sender of the message automatically receives a notification when the receiver has tried to take a screenshot. This technology has been introduced in Jan. 2018 and is only supported by the newest version of the application. As extra protection of the sent messages, two methods that use a different kind of reading experience are implemented. One is for the text, where a colored block covers each word. When wiping on the screen from the top

[1] https://screenshieldkit.com/



(a) (b)

Figure 6.3: Comparison of the screenshot prevention methods in the applications, the Confide screenshot prevention (a) and the Atomize screenshot prevention (b).

to the bottom of a message, each line gets revealed one by another. The other procedure is for images sent, where the full picture is blurred and again by wiping over the screen always a small stripe gets disclosed. An example of how a screenshot that has been taken looks like, can be viewed in figure 6.3. File types like videos get blurred and have to be pressed on when the content should be revealed. If the finger leaves the screen, it is blurred again.

Atomize

In the application, the screenshot prevention is realized by changing the reading experience through two different animations, as described in section 5.2.15. For the text, each word is revealed one by another while the other words stay invisible. The images are screenshot-protected by moving blurred layers in a certain speed over the picture so that always only a small stripe of the image gets visible. Both animations can be started by touching the screen or stopped if the user wants the specific part to stay longer visible.

	Atomize	Confide
Registration data	-	Name, email or mobile number
Full anonymity	✓	-
Data stored	Registration data	Contact, address book, message data
Third party services	-	✓
Authentication	Bearer token authentication	Username password authentication
Contact exchange options	QR-Code (local), id exchange (remote)	QR-Code (local), contact exchange via address book or manual input (remote)
Ephemeral messages	✓	✓
Attachment limitation	-	✓ (*Standard* version)
Attachment types	Images only	Images, videos, documents, voice messages
Screenshot detection	-	✓
Screenshot prevention technique	Animations controlled by user	Image processing controlled by user

Table 6.2: Overview of the application comparison.

6.2.6 Discussion

After the several aspects have been reviewed and compared, it gets discussed which application has its advantages and disadvantages in the specific cases. In table 6.2 an overview of the comparison, separated in the most significant parts, is displayed.

Traceability of users

Regarding the traceability of users, it is noticeable that in Confide anonymity is not provided, because of the personal information that has to be entered from the user at the registration. Moreover, the privacy of users is not highly guarded, by phone number or email address a person can be found. The contact data is accessible from any external person who knows at least one of these two things. If the user connects the Facebook account with the application, it is also possible to find a person through his profile. Therefore it can be stated that Atomize has better established this part since no connection to the real person is obtainable.

In addition to this, Confide saves such personal contact data so that all relations between users are traceable to its origin. Even if the messages are encrypted, there is always a sender-receiver relation, and the sender is revealed. On the one hand, this is an advantage, and it is always clear who the real person behind a sent message is. On the other hand, the privacy gets decreased, and also no possibility exists to send or receive messages to/from anonymous users. In Atomize the sender-receiver information also gets stored in the message information to associate the related users, though again there is no relation to the real persons.

Data security

In both applications, the messages are asymmetric end-to-end encrypted. Therefore the communication is kept secure and can only be read by the meant receiver. Regarding the verification of the sender, it is not evident if Confide uses digital signatures too.

Confide stores many different pieces of information on their servers for analytical research and provides this data to third-party services. Even if most of the data gets stored in a hashed or anonymized form, again the personal contact information is processed and transmitted transparently and therefore can be easily inspected.

Regarding the authentication security, it is clear that Confide has a safer approach by protecting the application functionality and user data with a login process that adopts a username and password authentication.

Contact exchange

The method for locally adding a new contact is implemented quite similar in both applications. When the exchange should be processed remotely, Confide has the advantage that it is executable within the application without any further effort. However, this is also only possible, if access to the contact book of the device is granted or if the mobile number/email address is known beforehand. In Atomize, the only method to exchange the own contact remotely is to copy and send the id via a third channel. This process causes further effort but also establishes a more anonymized and untraceable approach. Moreover, if a new user should be added to Confide without knowing the email/mobile number, the exchange also has to take place on a third channel.

Since Confide uses personal data, the address book contacts on the smartphone, if access was granted, or also other Facebook users, if the Facebook account is connected, each user is visible to the outside and can be found immediately. This also somehow represents a risk for personal privacy.

Data transience

Each application has similar approaches for keeping the message data in a transitory stage. In both applications the messages get deleted, either if a user reads the last received one, or if the message persisted for too long on the server.

Confide provides additionally some chat history, where the by oneself last sent messages can be read afterward, to be able to keep some context of the conversation. Concerning security, it can be discussed that this concept is not very safe, because if any person gets access to the smartphone, it is possible to reproduce parts of the conver-

sation. Otherwise, it brings benefits with it because it makes it easier for the user to understand the new messages. Also, this feature is enabled by default and has to be deactivated conscious, which would be better if this was reversed.

Prevention of persistence threats

The methods implemented to prevent the users from persisting the data by screenshotting it are quite similar in both applications. Still, it can be argued that Confide uses a better approach by providing a better solution for controlling the reading experience. In addition to this, the Screenshield Kit implements an improved technique for recognizing a taken screenshot and deleting the content within it by adding a gray layer over the image.

6.3 Limitations

From the further illustrated parts, the several limitations that have been found for implementing higher privacy and security in messaging applications get explained.

6.3.1 Unintended access to device

The one not avoidable threat is that another person somehow gets access to the device. This can happen either when the device is stolen or lent to somebody. When this happens, it is still possible to read the last received messages and retrieve the contact data. In the worst case, the application could be misused for pretending to be the person that owns the device. Thankfully this option is improbable for the reason that nowadays every smartphone provides some very secure authentication technology like three-factor authentication, fingerprint or code input to protect the access to it.

6.3.2 Unintended access to server data

There exists the possibility that attackers might get access to the database of the server. By collecting the contact data and therefore the public ids and keys, it is theoretically possible to fetch the messages of the users on their behalf or sending data to them.

6.3.3 Contact exchange

When the exchange of the public id is performed via a third channel, the chance that a user could be traced back to the real person behind is increased. This always depends on the technology or system that has been used for the exchange and how secure it is.

6.3.4 Additional persistence threats

Although some methods for preventing the users from persisting the messages have been implemented, there exist still some other persistence threats. In particular, these include each possibility of taking videos or screen recording the messages.

6.3.5 File type handling

Another limitation can be found in the persistence prevention of several file types sent in messaging applications. There is no possibility to implement a proper screenshot protection method for videos and also voice calls can be recorded by other devices. When documents like PDFs are sent, there needs to be a functionality that enables the user to open the file and scroll through it. Therefore no useful persistence prevention has been found to enable this.

6.3.6 Group chats

This topic is not directly a limitation, but also should be mentioned, because it poses a potential security risk. In group chats, several persons share the same key for encryption and decryption of the messages. Therefore it is less complicated for an attacker to get the key data. Also adding a new user to the group chat contains its threats. It is possible by appending the new person into the chat group on the server or by manipulating this data on the way. So it becomes easier for an attacker to gain access to the chat and read out the messages.

6.4 Possible improvements

Now that the comparison has been drawn and the limitations have been clarified some possible improvements for the application can be determined.

6.4.1 Authentication process

In the progress of this work, it has become clear that the authentication used for the application is not secure enough. Therefore it would have been better to adopt another user verification process that at least requires some password protection.

6.4.2 Key creation/storage

Another improvement lies in the creation and storing of the asymmetric key pair. At the moment, there exists one private key that the public key gets created from. Furthermore, this pair is used to retrieve the symmetric key for the communication with an individual user. The new idea is to add an extra layer of security, by implementing another private key stored on the device and is then encrypted by the key from the secure enclave. Therefore the messaging process will be only asymmetric encrypted, and the advantage is given that the connection between two users is even more secure, because it is not comprehensible any longer, even if the device falls into the false hands.

6.4.3 Screenshot prevention

For the screenshot prevention, some improvements that would have supported a better reading experience have been found. That would be for example to give the user more control over the animations. Another approach could be to work with gray colors, for displaying the message content and is technically not possible to be captured by other cameras.

6.4.4 Additional features

In addition to the improvement of the screenshot prevention, another security upgrade would be to implement a method for detecting if a screenshot has been taken. By this the possibility would be given, to notify the sender of a message that the receiver tried to persist the content.

Chapter 7

Conclusion

The purpose of this thesis was to resolve new methods for implementing improved privacy and security in messaging applications, especially regarding the topics of anonymity and user traceability. The idea behind is related to the evidence that nowadays every person uses messaging applications for their very personal conversations, but one quickly forgets about the tremendous impact on our privacy by sending our data via digital communication channels. Thus, more secure approaches for mobile messaging fulfilling the demand of complete privacy for a person using such a service had to be found.

First of all, comprehensive research has been carried out on general techniques that support higher privacy and security in digital communication. Thus the most reliable methods should be clarified and adapted for the further work. Additionally, other messaging applications have been analyzed and tested on their security features to obtain an idea of how their approaches for safer conversations look like. After that, a concept including all the necessary parts for protecting the privacy of a user has been defined. In consideration of the research result, these parts were mainly related to the anonymity and traceability of a user. Thus it became clear that the main topics for implementing a private and secure messaging application concern the traceability of users, the exchange of contact data, the transience of messages and the data persistence threats. Furthermore, an architecture plan including a structure that serves the purpose of the defined requirements has been developed, and the methodology for building the messaging application has been determined. After that, the implementation of the client and the server has been realized and thoroughly tested. Thereby it turned out that the combination of several methods that either keep the privacy of a person or make the communication more secure lead to the optimum results. Finally, an evaluation of the implementation has been performed by comparing it to another already existing messaging application which implements quite similar approaches. The comparison was based on the criteria points determined in the conception. In addition to this, the several limitations that occurred when developing the application and some possible improvements have been outlined to inspire other persons for doing further research in this area.

The significant findings of this thesis were the new methods leading to the anonymity of persons when using messaging applications for their conversations. Furthermore, the findings are related to how data is secured, even when exchanged via digital communication channels. Especially the part for the registration of a user leads to high privacy because no personal data is required. Corresponding to this a huge advantage is also

given by using only generated ids for identifying a user. Therefore no connection to the real person is obtainable, and relations between persons are not verifiable. Even if someone gets unintentional access to the server database, there is no possibility to find out any personal information about the users. Moreover by the highly secure encryption of the messages and the private keys stored only on the secure enclave of the device, the content of a message can only be revealed by the intended receiver. In addition to this, a secure verification of the sender of a message is possible by sending an extra signature. Therefore the option is given to find out if a message has been altered, which again makes the communication more secure. Also, the transience of the message data has a significant influence on the privacy by ensuring that nothing sent is saved. Therefore again no relation between users can be obtained, nor the danger exists that confidential data gets persisted. This aspect is further supported by the screenshot prevention methods preventing users from saving the message content on their devices.

By combining all these approaches, the traceability of users and their data is minimized to a small level, and the demand for complete privacy for a person using such a messaging application is fulfilled. It is highly recommendable to adopt these methods for making conversations via messaging more private, secure and even anonymous.

7.1 Further research

Based on the topics in the sections 6.3 and 6.4, there exist some problems for which further research has to be performed. These include primarily the remaining security issues that have not been implemented or were not possible to realize in the scope of this thesis. They consist of the further development of the user data models stored on the server, the contact exchange, the screenshot prevention/detection, the handling of dynamic files and group messaging.

Regarding the data model stored in the server database for identifying a user, an improvement could be made by not saving the public key directly with the identifier. Due to the direct relation between the key and the id, any person that gets unintentional access to the data has the opportunity to transmit messages to the specific user. By implementing a method that separates the key from the other data, the possibility for attackers to encrypt and send unintended messages to a user would be minimized.

For the remote contact exchange, a more straightforward approach has to be found. The solution to exchange the id via another channel is cumbersome for the user and brings some security risks with it. This would be for example the arising danger when the id gets persisted while it is sent via another not secured communication way. Thus a relation to the real person can be drawn. Therefore another more secure option that can be invoked in the application and works in the distance would be a useful feature.

The topic of screenshot prevention is also highly remarkable for doing further research. So far, there exist some already very stable techniques that make the screenshots of messages unusable, but there are still some possibilities to persist small parts of the content. By developing further methods that refine the already existing approaches, improved data security could be provided.

Related to the additional data persistence threats new ways should be discovered to handle and protect dynamic files sent through the application. This topic is critical and one of the most difficult to find better solutions and improvements. Until now the

possibility is still given to persist the message content through recording it with a second device. Perhaps there could be found some solutions that use advanced image processing so that the cameras of other devices cannot recognize the screen display and thus make the recording useless.

Finally, a necessary part to investigate concerns the group messaging option. Every person benefits from the advantages to quickly send the same message simultaneously to various persons without any further effort. Therefore it is of high importance to find new solutions to eliminate the threats that come with group messaging. These include implementing new methods for the encryption of the data and securing a higher level of access to the group.

Appendix A

CD-ROM/DVD Contents

A.1 Project

- **Project/src:** Project files for *Atomize*
  - **/backend:** Project files for the backend (XCode Version 9.4, macOS 10.10)
  - **/frontend:** Project files for the frontend (XCode Version 9.4, iOS 11.3)

A.2 Thesis

- **/Thesis.pdf:** Thesis (this document)
- **/images:** Original raster and vector images
- **/literature:** Copies of the online literature resources

References

Literature

[1] Rajdeep Bhanot and Rahul Hans. "A Review and Comparative Analysis of Various Encryption Algorithms". *International Journal of Security and Its Applications* 9.4 (2015), pp. 289–306 (cit. on pp. 11, 12).

[2] Brian Duckering. "10 Years of (Hacking) iOS - Mobile Threat Intelligence Report". *Skycure* 2017 (2017) (cit. on p. 6).

[3] A. K. Jain and D. Shanbhag. "Addressing Security and Privacy Risks in Mobile Applications". *IT Professional* 14.5 (Sept. 2012), pp. 28–33 (cit. on p. 6).

[4] Burt Kaliski. "A Survey of Encryption Standards". *IEEE Micro* 13.6 (1993), pp. 74–81 (cit. on p. 13).

[5] Soheila Omer AL Faroog Mohammed Koko and Dr.Amin Babiker A/Nabi Mustafa. "Comparison of Various Encryption Algorithms and Techniques for improving secured data Communication". *IOSR Journal of Computer Engineering (IOSR-JCE)* 17.3 (Jan. 2015), pp. 62–69 (cit. on pp. 11, 13).

[6] Tole Sutikno et al. "WhatsApp, viber and telegram: Which is the best for instant messaging?" *International Journal of Electrical and Computer Engineering* 6.3 (2016), pp. 909–914 (cit. on p. 14).

[7] N. Unger et al. "SoK: Secure Messaging". In: *2015 IEEE Symposium on Security and Privacy*. May 2015, pp. 232–249 (cit. on pp. 3, 7).

Software

[8] *Apple AVFoundation Framework*. 2018. URL: https://developer.apple.com/av-foundation/ (cit. on p. 45).

[9] *Apple Security Framework*. 2018. URL: https://developer.apple.com/documentation/security (cit. on p. 41).

[10] *Auth Module*. 2018. URL: https://github.com/vapor/auth (cit. on p. 31).

[11] *Fluent Module*. 2018. URL: https://github.com/vapor/fluent (cit. on p. 31).

[12] *Fluent Provider*. 2018. URL: https://github.com/vapor-community/fluent-provider (cit. on p. 31).

[13] *Mongo Driver*. 2018. URL: https://github.com/vapor-community/mongo-driver (cit. on p. 31).

[14] *Mongo Provider*. 2018. URL: https://github.com/vapor-community/mongo-provider (cit. on p. 31).

[15] *MongoKitten Framework*. 2018. URL: https://github.com/OpenKitten/MongoKitten (cit. on p. 31).

[16] *Realm Framework*. 2018. URL: https://realm.io (cit. on p. 44).

[17] *Vapor Framework*. 2018. URL: https://github.com/vapor/vapor (cit. on p. 30).

[18] *Vapor-APNS*. 2018. URL: https://github.com/matthijs2704/vapor-apns (cit. on p. 31).

Online sources

[19] Natasha Aidinyantz. *A Glossary of Cryptographic Algorithms*. Nov. 2017. URL: https://www.globalsign.com/en/blog/glossary-of-cryptographic-algorithms/ (cit. on p. 11).

[20] Apple Inc. *APNs Overview*. 2018. URL: https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html (cit. on p. 31).

[21] Apple Inc. *iOS Security*. 2018. URL: https://www.apple.com/business/docs/iOS_Security_Guide.pdf (cit. on p. 22).

[22] Apple Inc. *UIApplicationDelegate*. 2018. URL: https://developer.apple.com/documentation/uikit/uiapplicationdelegate?hl=et (cit. on p. 41).

[23] Confide. *Privacy Policy*. May 2018. URL: https://getconfide.com/privacy (cit. on pp. 16, 56, 57).

[24] Dust. *Privacy Policy*. June 2016. URL: https://www.usedust.com/privacy-policy (cit. on p. 16).

[25] Infosec Institute. *CIA Triad*. Feb. 2015. URL: http://resources.infosecinstitute.com/cia-triad/#gref (cit. on p. 10).

[26] Bordan Orlov. *iOS Architecture Patterns*. 2015. URL: https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52 (cit. on p. 8).

[27] Pedro Henrique Peralta. *iOS Project Architecture: Using VIPER*. Apr. 2016. URL: https://cheesecakelabs.com/blog/ios-project-architecture-using-viper/ (cit. on p. 25).

[28] Margaret Rouse. *HTTPS (HTTP over SSL or HTTP Secure)*. URL: https://searchsoftwarequality.techtarget.com/definition/HTTPS (cit. on p. 14).

[29] Margaret Rouse. *Transport Layer Security*. URL: https://searchsecurity.techtarget.com/definition/Transport-Layer-Security-TLS (cit. on p. 14).

[30] Snapchat. *Privacy Policy*. May 2018. URL: https://www.snap.com/en-US/privacy/privacy-policy/ (cit. on p. 16).

[31] Telegram. *Privacy Policy*. Apr. 2018. URL: https://telegram.org/privacy (cit. on p. 15).

[32] Viber. *Privacy Policy*. May 2018. URL: https://www.viber.com/terms/viber-privacy-policy/ (cit. on p. 16).

[33] WhatsApp. *Privacy Policy*. Apr. 2018. URL: https://www.whatsapp.com/legal/#privacy-policy (cit. on p. 14).

[34] WhatsApp. *WhatsApp Security-Whitepaper*. Dec. 2017. URL: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf (cit. on p. 14).