FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

A Cryptographic Concept for the Secure Storage and Transmission of Medical Images on iOS Devices

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medizinische Informatik

eingereicht von

Michael Niszl, BSc
Matrikelnummer 0826391

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuer: Ao.Univ.Prof. Dipl.-Ing. Mag.rer.nat. Dr.techn. Rudolf Freund

Wien, 21. Januar 2016

(Unterschrift Verfasser) (Unterschrift Betreuer)

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

A Cryptographic Concept for the Secure Storage and Transmission of Medical Images on iOS Devices

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Medical Informatics

by

Michael Niszl, BSc
Registration Number 0826391

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Mag.rer.nat. Dr.techn. Rudolf Freund

Vienna, January 21, 2016 (Signature of Author) (Signature of Advisor)

Statement by Author

Michael Niszl, BSc
Meidlinger Hauptstraße 63/12, 1120 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

I hereby declare that I am the sole author of this thesis, that I have completely indicated all sources and help used, and that all parts of this work – including tables, maps and figures – if taken from other works or from the internet, whether copied literally or by sense, have been labelled including a citation of the source.

(Place, Date) (Signature of Author)

Acknowledgements

I would like to thank everyone who was involved in the development process of my thesis.

First and foremost I want to thank Rudolf Freund and Stefan Kuschnigg for their expertise, time and patience. Furthermore, I would like to thank Michael Schafferer who provided advice concerning security and cryptographic topics. I appreciate the scientific discussions and the valuable consultations which led to the final outcome of this thesis.

Most importantly, none of this would have been possible without the support and patience of my whole family. I would like to express my heart-felt gratitude to my parents Ingrid & Manfred and my brother Florian who encouraged and supported me during my entire academic studies. I also want to thank my grandparents Theresia & Franz and Anna & Michael who were a constant source of energy and always provided an environment of mental recreation for me.

This thesis is dedicated to all who have accompanied me throughout my endeavor.

Abstract

Smartphones have not only become indispensable in people's lives - they are also progressing into valuable assets in the medical sector. Considering the sensitivity of medical data, security and proper integration are major factors regarding an ubiquitous use of mobile health (mHealth). This thesis focuses on the design and implementation of a cryptographic concept for secure storage and transmission of medical images. The used reference application is part of a medical skin imaging and analysis software, which is employed during clinical trials, targeting patients suffering from radiation induced dermatitis (RID). The goal of this thesis is to create and implement an encryption architecture, which protects sensitive data even if an attacker breaks the security features of the native Operating System (OS).

At first, Austrian laws and medical guidelines like Integrating the Healthcare Enterprise (IHE) profiles are reviewed and security ground rules are derived. Then the current native iOS security mechanisms are evaluated and best practice methods are gathered. As a next step, a threat model, based on the reference application's specification, is created and the gained knowledge is used to design the encryption architecture. After the implementation of the cryptographic concept, using the iOS Software Development Kit (SDK), the prototype is analyzed. At first, theoretical threats are identified and recommendations are specified. Following this, mobile penetration testing frameworks and security assessment tools are utilized to analyze the prototype. At last, the results and exposed weaknesses are used to improve the designed concept.

The analysis showed that the majority of the native OS security mechanisms can be bypassed with elevated privileges on a jailbroken iOS device. Only the iOS Data Protection mechanism could not be bypassed on the passcode locked device. The raised privileges also allowed to annul the implemented certificate pinning mechanism, which furthermore enabled the undetected interception of the transmitted network traffic. Due to the additionally implemented security layer, the transmitted data was still cryptographically inaccessible. However, the runtime analysis disclosed that even method calls can be traced on a jailbroken device. This allowed the recording of the password as part of the Password Based Key Derivation Function (PBKDF).

Recent aspirations to legally bypass encryption mechanisms fortify the results that native OS security mechanisms are not sufficient to protect sensitive data on a mobile device. It is also not recommended to solely rely on basic specifications required by law or medical guidelines. State of the art algorithms and methods need to be utilized, evaluated and updated on a regular basis to provide an adequate level of security. Mobile Device Management (MDM) systems are a valuable asset to detect runtime manipulation. An additional layer of security, utilizing state of the art cryptographic mechanisms, and components securing the runtime are crucial for the safety of sensitive medical data and help to mitigate or even prevent threats.

Keywords

mobile, application, mobile app, iOS, Apple, mobile health, mHealth, biomedical skin imaging, security, threat modeling, cryptography, encryption, best practice, security analysis, penetration testing, vulnerability analysis

Kurzfassung

Smartphones sind nicht nur ein unabkömmlicher Alltagsgegenstand geworden, sondern entwickeln sich auch zu einem wertvollen Bestandteil im medizinischen Sektor. In Anbetracht der Sensitivität von medizinischen Daten sind die Sicherheit und die korrekte Integration von mobilen Geräten ein wichtiger Faktor von mHealth. Der Fokus dieser Diplomarbeit liegt auf dem Design und der Implementierung eines kryptographischen Konzepts, um medizinische Bilder sicher zu speichern und zu übertragen. Die Referenzapplikation ist Teil einer Hautanalyse-Software, welche in klinischen Studien bei Radiodermatitis-Patienten eingesetzt wird. Das Ziel ist es, eine Verschlüsselungsarchitektur zu implementieren, welche sensible Daten selbst dann schützt, wenn die nativen Sicherheitsmechanismen des Betriebssystems außer Kraft gesetzt werden.

Zu Beginn werden Gesetze und medizinische Richtlinien, wie Integrating the Healthcare Enterprise (IHE) Profile, überprüft und grundlegende Sicherheitsstandards abgeleitet. Dann werden die aktuellen iOS Sicherheitsmechanismen evaluiert und „Best Practice“-Methoden erfasst. Im nächsten Schritt wird ein Gefahrenmodell erstellt. Anhand des gewonnenen Wissens wird die Verschlüsselungsarchitektur konstruiert und der implementierte Prototyp analysiert. Zuerst werden theoretische Bedrohungen identifiziert und entsprechende Empfehlungen spezifiziert. Darauf folgend werden Sicherheitsanalyse-Tools verwendet, um den implementierten Prototypen zu evaluieren. Zuletzt werden die Ergebnisse genützt, um das Konzept zu verbessern.

Die Analyse zeigte, dass die Mehrzahl der nativen Sicherheitsmechanismen des Betriebssystems mittels erhöhten Systemrechten umgangen werden können. Nur die iOS Data Protection-Funktionalität konnte auf einem mit Passcode gesperrten Gerät nicht umgangen werden. Weiters erlaubten die angehobenen Systemrechte auch das unbemerkte Abhören des Übertragungskanals. Durch die zusätzlich implementierte Verschlüsselungsschicht waren die Daten allerdings weiterhin kryptographisch unzugänglich. Die Laufzeitanalyse offenbarte jedoch, dass selbst Methodenaufrufe auf einem Gerät mit angehobenen Systemrechten protokolliert werden können. Das erlaubte die Aufzeichnung des Passworts als Teil der Password Based Key Derivation Function (PBKDF).

Aktuelle Bestrebungen, Verschlüsselungsmechanismen mittels gesetzlichem Beschluss zu umgehen, bekräftigen die Ergebnisse der Analyse, dass native Sicherheitsmechanismen nicht ausreichend sind, um empfindliche Daten auf einem mobilen Gerät zu schützen. Weiters ist es auch nicht empfehlenswert, sich ausschließlich auf Spezifikationen, welche durch Gesetze und medizinische Richtlinien vorgeschrieben werden, zu verlassen. Algorithmen und Methoden nach dem aktuellen Stand der Technik müssen verwendet, evaluiert und regelmäßig aktualisiert werden, um ein adäquates Sicherheitsniveau zu gewährleisten. Mobile Device Management-Systeme helfen dabei, Risiken wie eine Laufzeitmanipulation zu entschärfen. Eine zusätzlich implementierte Sicherheitsschicht und Komponenten, die die Laufzeitumgebung der Applikation absichern, sind ausschlaggebend für die Sicherheit von sensitiven medizinischen Daten.

Schlüsselwörter

mobil, Applikation, mobile App, iOS, Apple, mobile Gesundheit, mHealth, biomedizinische Bildverarbeitung, Sicherheit, Bedrohungsmodellierung, Kryptographie, Verschlüsselung, bewährte Verfahren, Sicherheitsanalyse, Penetrationstest, Gefährdungsanalyse

Contents

List of Figures

List of Tables

List of Abbreviations

AEAD Authenticated Encryption with Associated Data. 64, 67, 72

AES Advanced Encryption Standard. 4, 18, 23, 25, 41, 42, 47, 53, 55, 57–60, 64, 67, 68, 72, 95, 107

AHIMA American Health Information Management Association. 6

API Application Programming Interface. 1, 2, 16, 26, 31, 35, 38, 69, 72, 76, 78, 79, 94, 95

ARM Advanced Reduced Instruction Set Computing (RISC) Machines. 16, 28

ASLR Adress Space Layout Randomization. 28

ATNA Audit Trail and Node Authentication. 19

ATS App Transport Security. 96

BetrVG Betriebsverfassungsgesetz. 17

BMWi Bundesministerium für Wirtschaft und Technologie. 12

BSI Bundesamt für Sicherheit in der Informationstechnik. 98

BYOD Bring Your Own Device. 12, 13, 17, 19

CA Certificate Authority. 22, 69

CBC Cipher Block Chaining. 18, 62

CCC Common Criteria Certification. 96

CCM Counter with CBC-MAC. 23

CDMA Code Division Multiple Access. 32

CIA Central Intelligence Agency. 98

CISA Cybersecurity Information Sharing Act. 98

CsfC Commercial Solution for Classified. 96

CTR Counter. 18

CVE Common Vulnerabilites and Exposures. 3, 96

DER Distinguished Encoding Rules. 41

DHIS2 District Health Information Systems. 6

DoD Department of Defense. 55, 58, 73, 95

ECDHE Elliptic-Curve Diffie-Hellman. 47, 63, 64

eHealth Electronic Health. 7

EU European Union. 11

FBI Federal Bureau of Investigation. 98

FIPS Federal Information Processing Standard. 96

FOIA Freedom of Information Act. 98

GCM Galois/Counter Mode. 47, 64, 67, 68, 72, 95

GID Group Identifier. 23

GOe Global Observatory for eHealth. 6

GPS Global Positioning System. 5, 6

GSM Global System for Mobile Communications. 32

HFS Hierarchical File System. 72

HMAC Keyed-Hash Message Authentication Code. 41, 42, 53, 55, 57–60, 107

HSTS HTTP Strict Transport Security. 61, 64

HTTP Hypertext Transfer Protocol. 32, 47, 61, 62, 70

HTTPS Hypertext Transfer Protocol over TLS. 61, 69

ICSECD Infrastructure-Centric Security Ecosystem with Cloud Defense. 17

IDS Intrusion Detection System. 13, 16

IETF Internet Engineering Task Force. 61, 63, 64

IHE Integrating the Healthcare Enterprise. iii, iv, 18, 19, 99, 100

IPS Intrusion Prevention System. 13

IT Information Technology. 11, 13

ITI Information Technology Industry Council. 99

ITI IHE IT Infrastructure. 19

ITU International Telecommunication Union. 5

IV Initialization Vector. 41, 42, 45, 53, 55, 57, 79, 107–110

LLB Low-Level Bootloader. 22

LTE Long Term Evolution. 32

MAC Media Access Control. 83

MAC Message Authentication Code. 41, 42, 53, 64, 67, 72

MDFFP2 Mobile Device Fundamental Protection Profile v2.0. 96

MDM Mobile Device Management. iii, iv, 17, 27, 51, 76, 100

mHealth Mobile Health. iii–v, viii, 2, 3, 5–10, 12, 19

MITM Man-In-The-Middle. 3, 39, 62, 63, 68–72

NIST National Institute of Standards and Technology. 25

NSA National Security Agency. 98, 99

OECD Organisation for Economic Co-operation and Development. 11

openMRS Open Medical Record Systems. 6

OS Operating System. iii, 12, 14, 17, 73, 76, 99

OWASP Open Web Application Security Project. 30, 31, 36, 38

PBKDF Password Based Key Derivation Function. iii, iv, 57, 76, 79, 95

PBKDF2 Password-Based Key Derivation Function 2. 3, 26, 29, 41, 42, 57

PGP Pretty Good Privacy. 4

PPs Protection Profiles. 96

PRF Pseudorandom Function. 57

QR Quick Response. 32

REST Representational State Transfer. 31, 32

RID Radiation Induced Dermatitis. iii, 1

RISC Reduced Instruction Set Computing. 16

RNG Random Number Generator. 23

ROM Read-Only Memory. 22

RSA Rivest-Shamir-Adleman cryptosystem. 41–45, 47, 58–60, 96

SDK Software Development Kit. iii, 79, 94, 95

SHA Secure Hash Algorithm. 25, 41, 42, 64, 67, 72, 96

SIM Subscriber Identity Module. 12

SMS Short Message Service. 6, 13, 14

SNI Server Name Indication. 64

SQL Structured Query Language. ix, 38, 50, 85, 86, 94

SSH Secure Shell. 4, 73

SSL Secure Sockets Layer. 4, 39, 41, 42, 50, 61, 74, 110

STRIDE Spoofing Identity, Tampering with Data, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege. x, 30, 37–39

SWG Secure Web Gateway. 17

SWOT Strengths, Weaknesses, Opportunities, Threats. 6

TDEA Triple Data Encryption Algorithm. 18

TF Technical Framework. 19

TLS Transport Layer Security. 19, 47, 61, 62, 70, 96, 110

UDID Unique Device Identifier. 83

UID Unique Identifier. 23, 25, 26

UML Unified Modeling Language. 2

UN United Nations. 5, 11

URI Uniform Resource Identifier. ix, 85, 86, 94

UUID Universally Unique Identifier. 36

VM Virtual Machine. 16, 32

VPNIPSecPP1.4 VPN IPSecPP 1.4 Client Protection Profile. 96

WHO World Health Organisation. v, viii, 5–7, 9

XDS Cross Document Sharing. 18, 19, 99

XML Extensible Markup Language. 32, 38, 50

XN Execute Never. 28

1 Introduction

1.1 Motivation & Problem Statement

As smartphones have made an entry in the professional medical sector and are becoming a useful asset in various operational areas, the proper integration in the medical environment and data security are crucial subjects [48].

This thesis focuses on the creation of a practical cryptographic concept for the secure storage and transmission of medical images on iOS devices in context of a biomedical skin imaging mobile application. The application is part of a medical skin imaging and analysis software which was developed during a clinical *phase - I. trial* in cancer patients suffering from Radiation Induced Dermatitis (RID).

The mobile iOS application integrates with the web application by providing functionality for acquiring and uploading patient image data. It allows the study team to scan QR-codes for patient identification, to use the device camera to collect patient images and to upload the acquired images to the web application. The app also automatically whitebalances the image after scanning a color marker placed on the patient's skin.

The application is intended to be classified as class *I. measure (I.m)* medical product based on the guidance document for "Classification of medical devices" by the European Comission [31].

Recorded images may show the patients bare skin from different body parts and may also display the patients face or other body parts by which the patient can be identified.

Due to the possibility of a lack of connectivity to the server these images have to be persisted to the device storage and are therefore exposed to threats like malware, unintended data leakage, jailbreaking and more.

The goal of this thesis is to create and implement a cryptographic concept which secures the mobile iOS applications data on the device and during the transmission to the server even if the operating systems native security mechanisms are broken [11].

A concept will be designed considering cryptographic theories, state of the art standards and the operating systems hard- and software limits.

1.2 Expected Results

For this master thesis the following hypotheses are defined:

- It is hypothesized that an iOS application's data can be secured even if the operating system's native encryption mechanisms are broken or bypassed.
- It is hypothesized that only iOS crypto Application Programming Interface (API) functionalities are needed to secure the application data in a state of the art manner.

This thesis answers the question whether application data can be secured even if the smartphone's operating system's native security mechanisms are broken. It also shows if data is still safe when the smartphones rights management is bypassed by jailbreaking/rooting. Furthermore the connection between the smartphone and the server is reviewed to see whether the data is secure if the connection is compromised.

First, the hypotheses are proposed and the architecture is designed, then the designed concept is implemented and the assumptions are verified. At last, the results are presented and substantiated with the corresponding theoretical background.

The goal is to use the iOS APIs cryptographic functionality to its full extent and to create a prototype which can be integrated into the described medical image analysis platforms architecture.

Ultimately, the thesis objective is to design a cryptographic concept, implement a prototype and to review the security characteristics from a theoretical and practical point of view.

1.3 Methodological Approach

This thesis starts off with an initial literature review to outline the current situation of iOS security mechanisms. This will include general information and the native system security including the Secure Boot Chain and the Secure Enclave. Furthermore the encryption and data protection and app security, including the keychain and Data Protection Classes are looked into [11] [69].

Also information about best practice scenarios, manipulation mechanisms and security guidelines regarding the iOS operating system will be gathered [76].

After a theoretical analysis of the iOS security features and the identification of the best suitable available encryption algorithms the next step is the creation of a threat model.

The threat modelling process includes the application architecture, the description of the processed data, the identification of potential threat agents, the possible methods of attack and the control mechanisms [30] [71].

After completing the threat model and identifying possible attacks and counter measurements, a cryptographic concept and an encryption architecture are designed. A terminology definition and UML-flowcharts are used to describe the architectural characteristics and the cryptographic work flow.

Following the creation of the software prototype of the designed concept, forensic approaches (see [68] [77] [39]) are utilized to inspect the implemented setup. The implemented prototype is also reviewed from a theoretical cryptographic point of view. The findings are used to refine the hypotheses, to expose theoretical weaknesses and to give an outline for an improved theoretical concept.

In the event that information from the medical image analysis software is not available, assumptions are made to make the software as secure as possible. If it isn't possible to connect the applications current backend solution with the mobile prototype in a reasonable amount of time, a provisional backend is used during the prototypical setup and evaluation. On the occasion that the application can not be implemented to the full extent in the specified time span, the evaluation of the cryptographic concept will be carried out from a mainly theoretical point of view.

1.4 State of the Art

A global survey of the WHO shows that mHealth is occurring in many member states but mainly in an experimental manner rather than in form of strategic implementations. Despite the fact that mHealth initiatives are growing throughout different countries the percentage of evaluated projects is very low (12%). The report also shows that data security and privacy are crucial areas of mHealth technologies and need proper protection and attention [48].

The native iOS hardware and software protection mechanisms and architectures are thoroughly analyzed by different researchers [69] [17]. Most researchers, though, neglect the possibilities of broken built-in security mechanisms and the customized protection of application data.

Until 1st of September, 51 vulnerabilities regarding iOS have been registered in the Common Vulnerabilites and Exposures (CVE) database for the year 2014.[1]

For example, based on CVE-2014-1266[2] the SSLVerifySignedServerKeyExchange function did not verify the signature in a TLS Server Key Exchange message. This enabled Man-In-The-Middle (MITM) scenarios where an attacker was able to trick a SSL server and use a wrong private key for the signing step or skip the step at all [52].

The vulnerability with the CVE-id CVE-2014-1276[3] allowed the Fireeye team to create a background monitoring application on a non-jailbroken iOS 7 device. They were able to record touch/-press events on the screen, home button press, volume button press, and TouchID press. The team also managed to send all logs to a remote server with the background application on the non-jailbroken iPhone 5s [60].

Another TLS based iOS vulnerability is CVE-2014-1295[4] which is described as a triple handshake attack and allowed MITM attackers to modify the session and obtain sensitive data. The team of the miTLS[5] research project has given more details on the attack and the core weaknesses in TLS in their paper [20].

A severe vulnerability (CVE-2014-1359[6]) in iOS 7.1 was discovered on 1st of July 2014 which allowed attackers to execute arbitrary code through a vulnerability in the launchd function. This attack didn't require any authentication and was a threat to confidentiality, integrity and availability.

The last example of vulnerability with the id CVE-2014-1348[7] is the false advertising of Data Protection for mail attachments. Mail in Apple iOS before 7.1.2 stored attachments in clear text although thorough Data Protection was communicated to the user. Attackers could therefore easily access mail attachments by mounting the data partition [50].

Zdziarski (see [76]) shows a number of different concepts which can be implemented in an iOS application to enhance its security qualities, like the usage of a master key for symmetric encryption, the possibilities of key derivation functions e.g. Password-Based Key Derivation Function 2 (PBKDF2) and the public key cryptography. He also describes the proper implementation and limitations of secure memory- and file deletion on an iOS device [76].

The work "iOS 7 Programming Pushing the Limits" by Napier (see [62]) outlines a best practice implementation of hybrid cryptography on iOS 7. Napier implemented a library where he tries to eliminate common mistakes and chose encryption algorithms, message authentication codes and architectural patterns based on the consultation of the Apple security team. [8]

[1] http://www.cvedetails.com/vulnerability-list/vendor_id-49/product_id-15556/year-2014/Apple-Iphone-Os.html - Serkan Oezkan - CVEdetails.com the ultimate security vulnerability data source (visited on 09/01/2014)
[2] http://www.cvedetails.com/cve/CVE-2014-1266/ - Serkan Oezkan - (visited on 09/01/2014)
[3] http://www.cvedetails.com/cve/CVE-2014-1276/ - Serkan Oezkan - (visited on 09/01/2014)
[4] http://www.cvedetails.com/cve/CVE-2014-1295/ - Serkan Oezkan - (visited on 09/01/2014)
[5] http://www.mitls.org/ - INRIA, Microsoft Research, IMDEA software institute - (visited on 09/01/2014)
[6] http://www.cvedetails.com/cve/CVE-2014-1359/ - Serkan Oezkan - (visited on 09/01/2014)
[7] http://www.cvedetails.com/cve/CVE-2014-1348/ - Serkan Oezkan - (visited on 09/01/2014)
[8] http://rncryptor.github.io/ - Napier, Robert. RNCryptor - Crossplatform Advanced Encryption Standard (AES) data format and implementations (visited on 09/01/2014)

In matters of a secure connection between the iOS device and the web service, the bettercrypto.org team has written a guideline which is currently in draft status and represents *"...an updated, solid, well researched and thought-through guide for configuring Secure Sockets Layer (SSL), Pretty Good Privacy (PGP), Secure Shell (SSH) and other cryptographic tools..."* [22]. The section *3.2 Cipher suites* outlines the best practice possibilities for the choosing of the key exchange protocol, the authentication mechanism, the cipher and the message authentication code for a SSL secured connection [22].

1.5 Reference to the Chosen Field of Study

The mobile application is part a of cloud based biomedical skin imaging and analysis software and therefore represents a prime example of a medical informatics appliance.

This master thesis engages with different parts of my chosen field of studies such as medical imaging, clinical trials and mobile health. It also focuses on the technical areas of my academic studies like mobile platform development, cryptography, system architecture and IT-security.

My goal is to make use of my acquired skills and knowledge and to expand my professional know how in the areas of mobile platform development, medical data management and especially IT-security.

2 mHealth - Mobile Health

2.1 Introduction

Based on a study by the International Telecommunication Union (ITU), about 6 billion mobile phone subscriptions have been registered world wide in 2011. More than 80% of the 660 million new subscribers in 2011 originate from developing countries. It was estimated that by the end of 2014 7 billion people, equal to 96% of the global population, will have a mobile-cellular subscription [42]. This means a higher penetration of mobile phones than of water and sanitation services [16].

Since the 1990s, two decades of plummeting costs led to a more pervasive access to telecommunication technology, enabling low and middle income countries to connect to complex systems via mobile phone [58].

With higher network speed and more powerful handsets a new field of e-health opens up. The spread of mobile technologies leads to a new perspective how health services are delivered, accessed and managed. It enables timely access to general health services as well as emergency services and information [48].

Mobile health is expected to transform the health care industry in a more personalized and participatory sector. mHealth has the potential to counter health care systems shortcomings and improve an individual's health [57].

Also the "always on" status, portability and data transmission functionality are qualities of mobile phones. Therefore, it supports health providers to offer cost effective services by reducing distance, time and cost of information delivery [16].

The mHealth technologies potential is recognized by the United Nations (UN) and the World Health Organisation (WHO). They underline that every person should have access to health services regardless of their financial status. These concepts were defined as global strategies in the WHO's 2010 World Health Report and the UN General Assembly Resolution in 2012. Private sector engagement may provide some basic health services, but innovations are needed to enable unrestricted access to affordable services and to lower the costs for the millions of people who need health care [58].

A crucial driver in health care is the aging population. This leads to an increased demand of continual care due to the upsurge of patients with chronic diseases, especially in developed countries [57]. Mobile applications provide a quick and easy way to collect end user data in the user's own environment. A mHealth application can be used in various fields of operation. It can assist a patient with monitoring or self-reporting the patient's health state or by tracking the adherence of medication or a treatment [28]. Thus, it reduces the need of hospitalization, since non-critical care can be managed in the community, consequently decreasing the cost of caretaking and improving the patients life quality [57].

Modern mobile phones are equipped with a variety of sensors like an accelerometer, a gyroscope, a camera, a microphone or a Global Positioning System (GPS) sensor which can provide additional information about the patients current context. Additional external sensors, which can be carried or worn on the body or the clothes, can enable access to further monitoring scenarios [28].

2.2 Definition

There is no common definition of the term mHealth, but it can be generally viewed as a *"...driving force in transforming health-care delivery, making some elements of health care faster, better, more accessible and cheaper."* [57].

Furthermore the Global Observatory for eHealth (GOe) defines mHealth as a *"..medical and public health practice supported by mobile devices, such as mobile phones, patient monitoring devices, personal digital assistants (PDAs), and other wireless devices."* [48]. This implicates not only the use of mobile phone core components like voice or Short Message Service (SMS), but also more complex features like applications, mobile radio technologies, GPS and wireless technologies [48].

Another definition for mHealth is given by the American Health Information Management Association (AHIMA): *"the use of devices such as smartphones or tablets in the practice of medicine, and the downloading of health-related applications or 'apps' ... [to] help with the flow of information over a mobile network and ... improve communication, specifically between individuals and clinicians."* [70].

2.3 Studies

Global health agencies are starting to use mHealth systems on a countrywide basis. Facility-level medical records are implemented by low-cost open-source products like District Health Information Systems (DHIS2) and Open Medical Record Systems (openMRS). According to Mehl and Labrique's article in 2014 (see [58]), in the past five years the field of mHealth has matured and the enterprise mHealth systems are on the rise. At least three WHO-lead initiatives have been established and several hundred trials, measuring the efficacy of mHealth, have been conducted [58].

Nonetheless, many mHealth systems only address individual problems and neglect taking the complexity of health systems into account. In most cases the quality of the care depends on different preceding layers of enabling conditions, like finding out who is in need and providing the necessary resources at the right time and place [58].

2.3.1 mHealth in Africa

Aranda-Jan, Mohutsiwa-Dibe, and Loukanova analysed the experience of mHealth implementations in Africa (see [16]).

They synthesized the data using a strengths, weaknesses, opportunities and threats (SWOT) analysis and systematically reviewed 44 studies on mHealth projects carried out in Africa between 2003 and 2013. The results were categorized into 'project implementation', 'integration to health systems', 'management process', 'scale-up and replication' and 'legal issues, regulations, standards' [16].

Results showed a regular positive effect of mHealth on health-care. The main aspects of successful mHealth projects are accessibility, acceptance and low-cost of the technology. Critical factors are the adaption to local contexts, strong stakeholder participation and collaboration by the government. mHealth innovations can be a useful asset by helping with problems like unsteady surveillance and reporting systems, poor management of drug stock and lack of other resources. This kind of implementations can become an important part of the health sector, but face threats like dependency on funding, unreliable infrastructure, unclear health-care system responsibilities and lack of evidence [16].

In the case of Africa, the evidence remains poor and the results are project- or setting-specific which provides no solution for health system problems faced in many African countries. Especially scalability, cost-effectiveness and sustainability are issues yet to be addressed [16].

2.3.2 WHO Global Survey

The WHO conducted their second global survey on Electronic Health (eHealth) in 2009 built on the knowledge accomplished by the first survey in 2005. The first survey mainly asked high-level questions at a national level in contrast to the second survey which comprised more detailed questions and was thematically designed [48].

The goal of the mHealth module, of the second survey, was to document the existence and maturity of mHealth, the type of mHealth initiatives, the status of monitoring and evaluation of mHealth initiatives and the barriers to adoption in the member states [48].

The report shows that most member states experiment with technologies rather than build a strategic implementation. Responsible administrators need a certain knowledge to implement initiatives on a large-scale basis. Most of the barriers to mHealth implementation result from a lack of knowledge. Other barriers are the lack of a supporting policy, conflicting health system priorities and legal issues [48].

83% of the 112 member states reported at least one mHealth project, see figure 2.1 on page 7 and figure 2.2 on page 8. Most of the 83% implemented four or more projects [48].

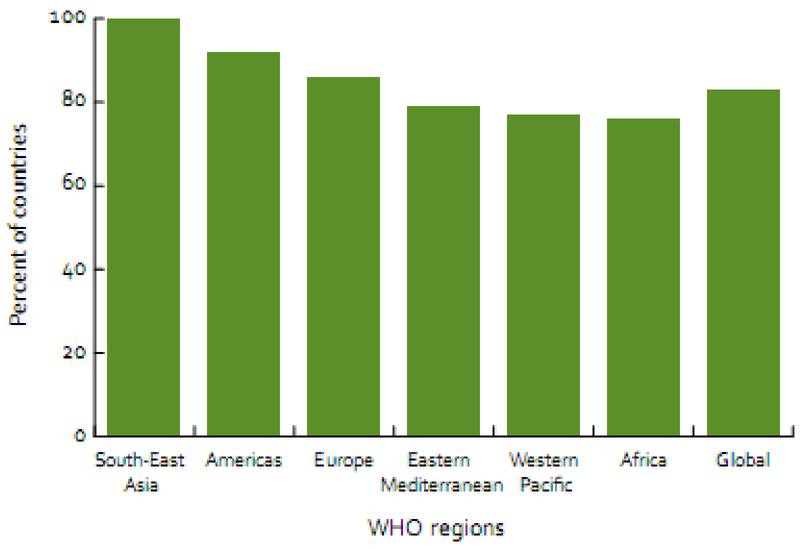


Figure 2.1: Member States reporting at least one mHealth initiative, by WHO region [48]

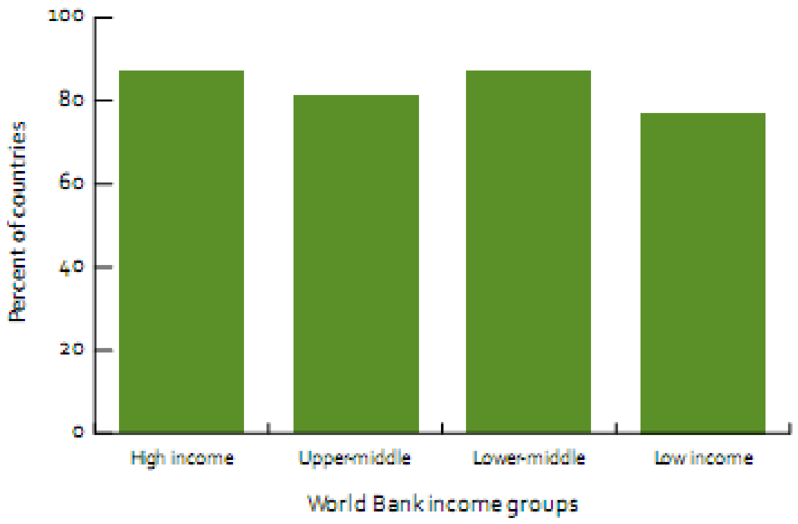


Figure 2.2: Member States reporting at least one mHealth initiative, by World Bank income group [48]

Health call centres (59%), emergency toll-free telephone service (55%), emergency (54%) and mobile telemedicine (49%) are the most frequent mHealth categories deployed in the member states, see figure 2.3 on page 8 [48].

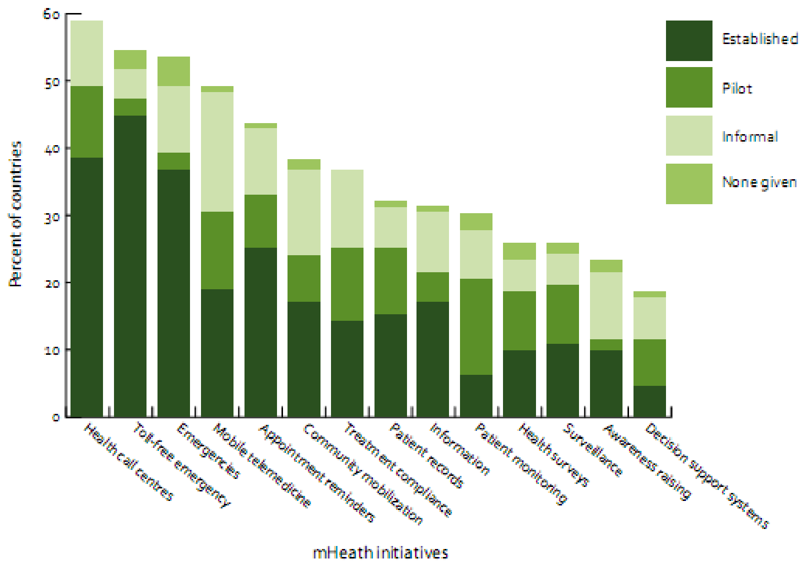


Figure 2.3: Adoption of mHealth initiatives and phases, globally [48]

Except for the African region, the use of health call centres was relatively high. This could be due to the insufficient technological infrastructure, see figure 2.4 on page 9 [48].

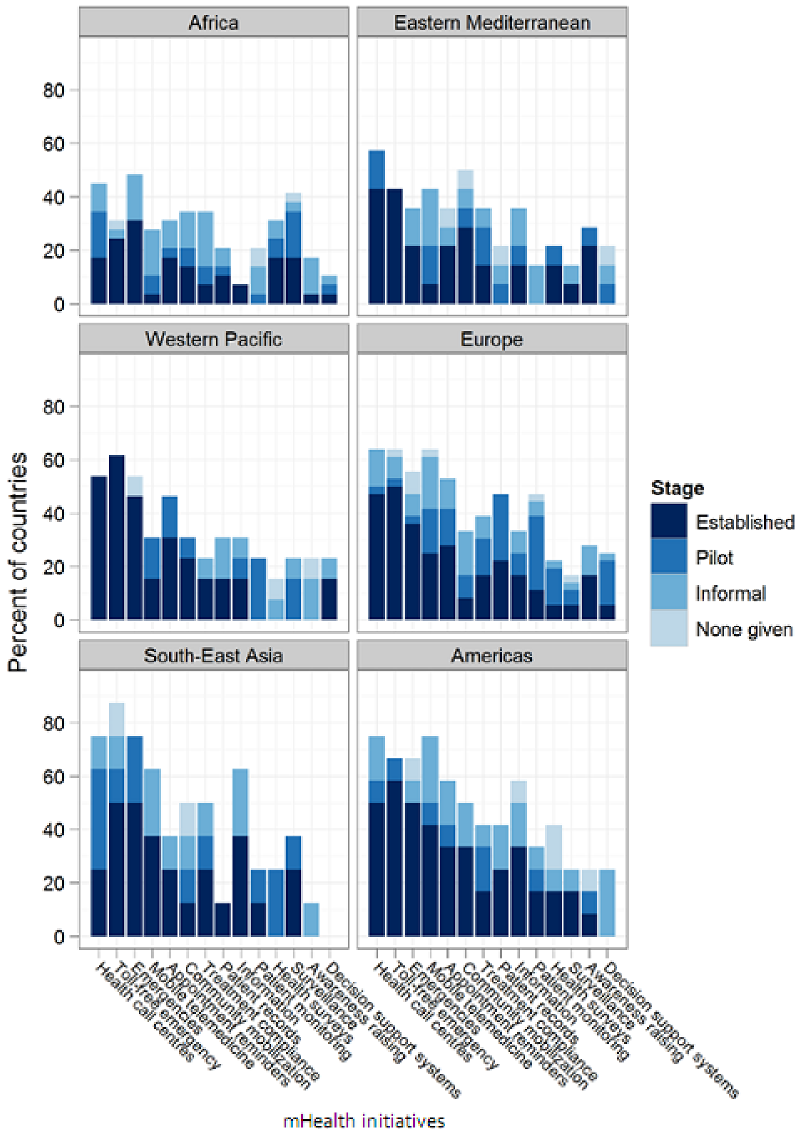


Figure 2.4: Adoption of mHealth initiatives and their phases, by WHO region [48]

The survey also shows that high-income countries have a greater range of initiatives than low-income countries, see figure 2.5 on page 10. Also health survey and surveillance projects were the least reported initiatives by high-income countries, but were relatively often reported in low-income countries. A probable explanation for this phenomenon is that countries in higher income groups have an existing surveillance system and do not need these kind of projects [48].

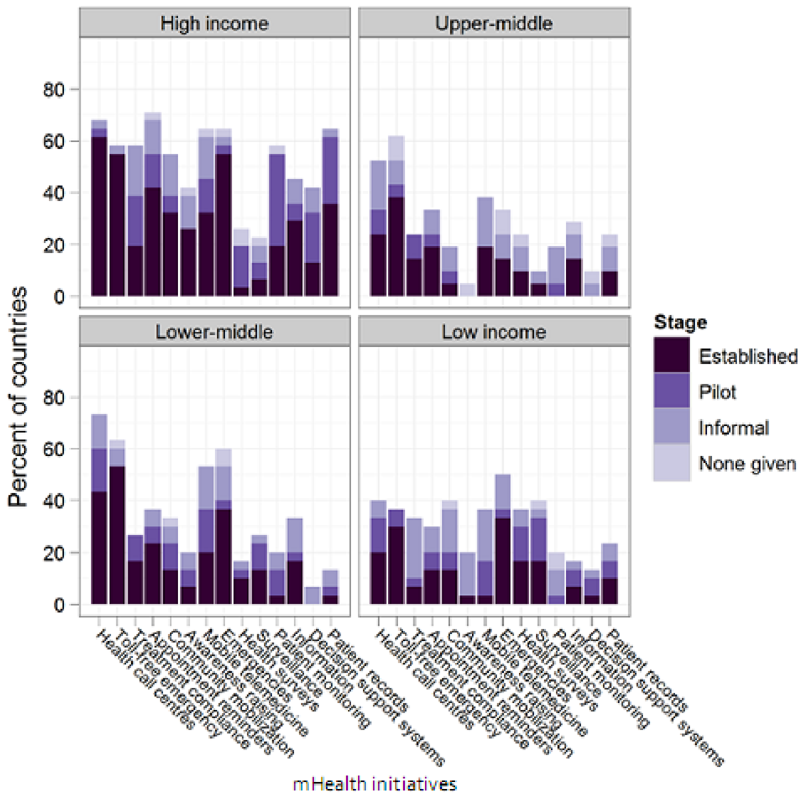


Figure 2.5: Adoption of mHealth initiatives and their phases, by World Bank income group [48]

2.4 Conclusion

There is a lot of activity regarding mHealth, but little attention comes from academic researchers and there is no effort towards strategic planning. Information is found in reports from private research companies, foundations and consultants only [57].

Surveys show that providers lack goals, budgets, plans, business drivers or leadership for mHealth projects [57].

Nevertheless health care is viewed a top three market to drive mobile device growth over the next five years. According to Malvey and Slovensky (see [57]), mHealth has the potential to address four key aspects of the health care sector:

- **Prevention**: public health and lifestyle awareness
- **Monitoring**: pre-disease screening and assessment
- **Treatment**: providing efficient and effective care
- **Support**: for patients along with caregivers

Continual adoption and innovation of mHealth enable a transformation of the health care sector and create new global health care markets [57].

3 IT Security

3.1 Introduction

While in the beginning Information Technology (IT)-Security research was mainly funded by the government and therefore focused on confidentiality and the secrecy of classified information, the definition of the term IT-Security has evolved in the last decades [33].

Since the late eighties IT-Security has been defined by the following parameters [33]:

- **Confidentiality**: The prevention of unauthorized or improper disclosure of information or resources.
- **Integrity**: The prevention of unauthorized or improper changing of data. Assurance that data is a proper physical and semantic representation of information.
- **Availability**: The prevention of unauthorized withholding of data and resources.

According to Dierstein (see [29]) there are two complementary views on IT-Security: the security of the system and the security of the user.

To protect the user's privacy, confidentiality, integrity and availability have to be guaranteed. The best approach to avert abuse of personal data is minimisation or, if possible, avoidance of personal data [33].

The term privacy was first defined by Warren and Brandeis in the article "The Right to Privacy" in the year 1890 (see [74]). Due to the recent development of new forms of technology, Warren and Brandeis defined privacy as *"right to be alone"* [74].

The currently most widespread definition is the one by Alan Westin: *"Privacy is the claim of individual, groups and institutions to determine for themselves, when, how and to what extent information about them is communicated to others"* [75].

Basic privacy principles are required to protect an individuals right of informational self-determination. Most of the following privacy requirements are defined by European Union (EU) Directive on Data Protection (see [32]), the United Nations (UN) Guidelines (see [72]) and the Organisation for Economic Co-operation and Development (OECD) Guidelines (see [65]) [33]:

- **Lawfulness and Fairness**: Personal data is collected and processed in a justifiable and legal manner.
- **Purpose Specification and Purpose Binding**: Processing and collection of personal data has to be specified and consequently limited to these specifications.
- **Necessity of Data Collection and Processing**: Collection and processing is only granted if necessary for the task.
- **Information, Notification and Access Rights**: The subject of the data has the right to information, notification, correction, erasure and blocking of incorrect or illegally stored data.

- **Security and Accuracy**: Proper organizational and technical security mechanisms have to be installed to ensure confidentiality, integrity and availability of the data.
- **Supervision and Sanction**: An independent and designated data protection authority has to be installed to supervise and observe the provisioning of private data.

3.2 Mobile Security

In their 2013 Issue Brief "Networked Medical Device Cyber Security and Patient Safety: Perspectives on Health Care Information Cyber Security", the Deloitte Center for Health Solutions executives described that mHealth devices have high potential in the future of health care, but are also a big factor in exposing organisations and patients to safety and security risks. Especially the personal health information stored on such devices represents a potential target and raises concerns. If personal health information is stored on a mobile device, it is exposed to threats like theft, improper access by unauthorized users, infection with malware or hacking [57].

One fundamental problems is that mobile devices were originally built for the consumer market and therefore it is a challenging task to even generally integrate them in an existing company IT infrastructure, leaving aside the potential security issues [49].

Mobile- and featurephones dominated the market until 2007. Featurephones contain a Subscriber Identity Module (SIM) card like normal mobilephones, but provide various wireless interfaces and more complex features. These features are predefined by the phone manufacturers and can not be modified by the user [49].

After the introduction of smartphones and the increasing market share of mobile OS like iOS and Android, the security situation of mobile devices drastically changed. In contrary to featurephones, smartphones run different kinds of mobile OS, depending on the the manufacturer that enable a much broader range of features and can be extended with mobile applications [49].

A study of the Bundesministerium für Wirtschaft und Technologie (BMWi) (see [23]) showed that in 2012, 88% of small and middle sized companies in Germany used mobile devices as part of their daily business. 44% of these companies even allowed mobile access to company data [49].

According to the market research company Gartner Inc., 50% of the firms worldwide will stop providing mobile devices by 2017 and will require employees to bring their own [54].

The trend of Bring Your Own Device (BYOD), where an employee is allowed to use their personal mobile device in company context, started in 2003 and took of in 2011. A BYOD policy saves the company time and money due to the fact that no adjustment or training on company equipment is required [54].

In 2012 Ovum, a market research firm, held a survey of 3796 consumers in 17 countries which showed that 75% of users in emerging countries and 44% of users in developed countries used their own mobile devices at work, see figure 3.1 on page 13 [54].

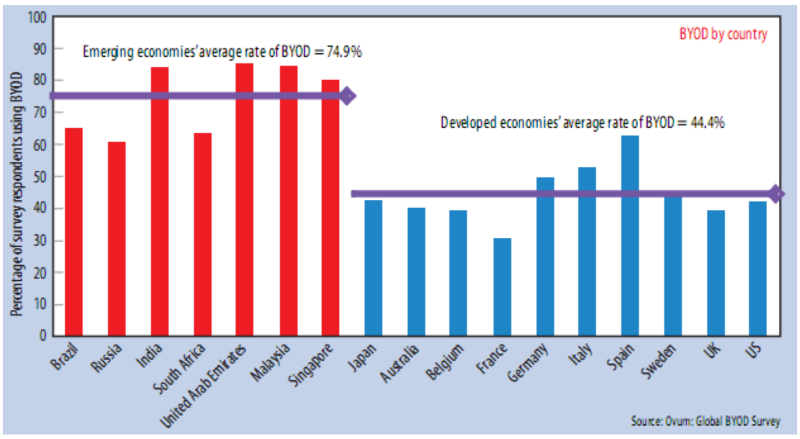


Figure 3.1: Bring Your Own Device (BYOD) usage, by country [54]

3.3 Challenges

3.3.1 General

With the introduction of mobile devices to the company IT infrastructure, the demand for easily accessible IT services is high. This generates new challenges for a company's general security concept. If security architects do not take this demand into account, users will find a way on their own to access certain resources of a company network. Abusive behaviour with mobile devices is very hard to monitor, especially when there are no strict boundaries between private (BYOD) and company owned devices [49].

A benchmark study of 80 organizations, carried out by the Ponemon Institute in 2013, showed that the key problems of information security are the inadequate funding, solutions and expertise [57].

In 2013, in context of their annual mobile security report, Check Point Software Technologies conducted a poll of about 800 IT professionals worldwide. It showed that 80% of the responding corporations had mobile security problems in the past year, where 42% experienced a breach of their security mechanisms costing their companies more than $100.000 USD. About two-thirds of these companies allowed employees to connect their private mobile devices to the company network, but 63% did not monitor the access and usage of corporate-data on these devices [54].

IT professionals have to alter their perception on security mechanisms for mobile devices and realize that conventional mechanisms like Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS), passwords and firewalls are not suitable to handle these kinds of threats. They focus on handling external risks and neglect the fact that BYOD issues evolve from within the organization [54].

The motives of mobile cybercriminals can be separated into three groups [66]:

- **Obtaining financial gain:** This is the most well known motivation. An example is the sending of premium rate SMS. In this scenario the malicious application sends out a SMS to a service provider that charges for the service and hides the confirmation from the user.

Another technique is to collect the device owner's address book and sell it to spam companies.

- **Collect sensitive data**: In this case the cybercriminal uses, for example a key logger, acquires location information or uses other methods to acquire personal data. The data is then used for identity theft, to blackmail the device owner or to sell it on the black market.
- **Access private networks**: This approach is popular especially in companies that have established a BYOD policy. The attacker utilizes the malware to access corporate resources, to steal data or to enable the companies network resources for the use in a malicious botnet.

3.3.2 Mobile Security vs. PC Security

As Rob Bamforth, principal analyst for business communications with the market research firm Quocirca, states, nowadays most security approaches that were used for the desktop environment, are used for mobile security [54].

The security engineers, though, have to keep the major difference between desktop and mobile environment in mind. Desktop or laptop personal computers have the appropriate computing power to run complex antivirus and antimalware software and personal firewalls [55].

Mobile OS are equipped with an application privilege separation model, which means that the OS kernel runs each application in a separate sandbox. Only the user or the kernel can grant access to these sandboxes. A problem of this concept is that the user has to be capable of making good security choices and decide whether he grants an application certain permissions or not. Studies show that criminals take advantage of the gap between the users perception and the reality of the risk. In contrary to the PC software market, where traditional developers are well known, each mobile application developer is a publisher. This allows a developer to distribute his application throughout the application store and attack a large scale of users with a single app. The thriving number of mobile applications makes it impossible for the security industry to analyze, catalog and create signatures for identifying embedded malware. Especially since it is very common to repackage well known malicious and clean applications with an injected malicious code to change the signature [55].

Yet, not only the limited processing and memory resources are critical differences regarding the development of secure solutions for mobile devices, also the limited battery and display size have to be taken into account. It is important that security applications do not take large portions of CPU time to avoid fast drainage of the battery. The multitude of wireless interfaces makes the mobile device prone to eavesdropping, as do the phone's sensors like the microphone which isn't optional and can be used to sniff private data. Due to the "always online" concept of smartphones, attacks can even be carried out when the user is not interacting with the device. Another difference regarding PC and mobile security is that it is more easy to make money with mobile malware, for example with premium rate calls or SMS [51].

In summary there are five key aspects which have to be kept in mind for the creation of mobile security solutions [51]:

- **Mobility**: The mobile device can be used everywhere, therefore it can also be easily stolen and tampered with.
- **Strong personalization**: In most cases, there is only one owner of a mobile device.

- **Strong connectivity**: Smartphones offer many different interfaces which can handle different tasks. This offers an attacker a huge number of different ways to exploit a mobile phone.
- **Technology convergence**: Like personal computers, a modern mobile device consists of different technologies. This diversity of features further extends the possibilities of attacking the device.
- **Reduced capabilities**: Smartphones offer a lot of features, like pocket PCs, but only present limited resources compared to PCs, like limited processing power, limited memory, limited battery life, no keyboards and a limited display size.

3.4 Solution Approaches

3.4.1 General

Mobile devices do not only posses characteristics that are challenging in case of the creation of secure solutions, but also have features that can help developing secure mobile software. For example the context information, which can be derived from the multiple sensors like location, movement, light, sound, and radio, can assist with decision making in regard of enabling the proper security policies. The current location can be taken into account when it comes to deciding which locking mechanism and which timeout should be used. This implies that the used data has to be taken care of and has to be encrypted before stored on the device [19].

Regarding the use of mobile devices in a clinical environment Burgess (see [24]) suggests the following basic ground rules. (1) Encrypt all transmitted data and store it on a dedicated secure server. (2) Establish robust policies, create password requirements and use access control software. (3) Provide a documented strategy for the security of mobile devices. Other useful approaches are the physical access protection of the devices, which means to create secure facilities for equipment and data storage, implement data and media destruction protocols and provide backup and contingency plans. Device access protection should be used. It enforces user authentication on the mobile device, allows remote wiping of data and makes sure that data is only transmitted over secured wireless networks. The monitoring of audit logs is an important part for administration. It enables the possibility to respond to an unauthorized access in a predefined manner. Administrative methods also include the proper training of personnel in security practices [57].

Preparatory training of employees is a step that should not be neglected. Since mobile devices are also used by individuals who have little relation to IT and are also part of their daily routine, they are not perceived as part of information technology any more. Therefore the awareness of potential risks is very low. Many users also don't have the certain knowledge to decide whether an application or website is a threat to their mobile device and their data. It is important to clarify terms like phising or identity theft for the personnel. The importance of the stored data has to be communicated throughout the different levels of the organizational hierarchy. Possible threats and counter measurements should also be part of the training [49].

3.4.2 Security Strategies

Malware detection

The techniques to detect malicious applications can be separated into different categories.

The signature-based technique uses a signature database of known malware and prevents known malicious apps from installing. A big problem with this approach is that it can only detect known applications and even if there is a slight change committed to a known malware it has a new signature and the method fails to detect it [66].

Samsung's newest smartphones provide a built-in security system called KNOX. KNOX consists of three main parts: the Customizable Secure Boot, the Advanced RISC Machines (ARM) Trustzone-based Integrity Measurement Architecture and the kernel with built-in security enhancements. The Customizable Secure Boot ensures that only authorized and verified software is executed. The ARM Trustzone-based Integrity Measurement Architecture monitors the integrity of the kernel and takes action when the boot loader is violated. These three components enable the possibility to separate information by confidentiality and security requirements.
Google partnered with Samsung and brought KNOX key features to the Android Lollipop release. By building on Android's multi-user support, a data separation between corporate and personal data can be created and encrypted using block-level disk encryption. KNOX's APIs represent a superset of Android Lollipop's enterprise API that enables the enforcement of a wide set of policies and restrictions.[1] Manufacture built-in security in context of Apple and iOS is described in more detail in chapter 4 on page 21 [66].

As described in chapter 3.4.1 on page 15, a substantiated knowledge is one of the most effective prevention and detection techniques. Users that are aware of topics like backups, malware and other security related issues are capable of noticing anomalies on their mobile devices and can identify malicious activities [66].

The market of mobile security offers various approaches and solutions to tackle technical attack vectors. Available solutions are: device based Intrusion Detection System (IDS), ontology based firewalls, behaviour based detection and cloud based protection. Some of these systems rely on a server based decision making and threat modelling system and on on-device lightweight clients. Other approaches focus on the mobile OS core and use functionalities like virtualization [55].

A more futuristic malware detection strategy is the anomaly/heuristics based detection. The system constantly monitors apps and tries to detect malicious behaviour. If an application, for example, invokes a collection of API calls which are known to be malicious, the user is alerted. This kind of detection can be done in real time and combined with permission-based detection, where an application is marked as malicious based on its functionality and granted permissions. Another useful asset to this methodology is the monitoring of incoming and outgoing connections to peers and servers that are known to be malicious [66].

According to M. Chandramohan and H. B. K. Tan (see [26]), cloud based detection is the future for fast, efficient and effective mobile security. This includes a system entirely responsible for static and dynamic analysis of malware. Penning et al. have outlined how a cloud based framework could work [66].

The process starts with a user request to download and install an application from an app store. This request is also sent to the known threat and known safe databases. If the application is found

[1] http://android-developers.blogspot.co.at/2014/07/knox-contribution-to-android.html - Srikanth Rajagopalan - Android Developers Blog (visited on 10/04/2015)

in the libraries, the result is returned. If the app is not part of the library, it is passed on to the malware detector. The detector then downloads the application and proceeds with an automated static and dynamic analysis of the app. The application is also deployed in a Virtual Machine (VM) and is tested by a human tester. If a threat is found by any of these testing mechanisms, the application is added to the known threat library, if not, it is added to the safe library. Finally, the user is informed whether the desired application is safe and can be installed or is a threat and has to be blocked. Especially the manual, human testing is a very time consuming task and compromises the practicality of this solution [66].

Device security

MDM systems represent an approach to inventory, monitor, manage and secure mobile devices. The functionalities vary from enforcing policies and disabling sensors to locking the device and wiping data. In contrast to the solutions, which tackle technical attack vectors and try to prevent attacks, MDM only allows reactive security.

From a user's point of view it makes the device more unattractive, reduces productivity and can simply be disabled by turning off the network interfaces. For the multitude of hardware manufacturers and mobile OS, it is very difficult to support and manage all systems. Also MDMs do not provide a universal remedy for BYOD problems and can not prevent the device from being stolen, which is an inevitable problem of mobile devices in general [54].

Although a MDM system's primary goal isn't to monitor the employees location and private data, the separation of corporate and private data is mandatory and the German Betriebsverfassungsgesetz (BetrVG) demands that *"...technical facilities that are intended to monitor the behavior or performance of employees"*[2] (BetrVG §87 Abs. 1) can only be granted by the work council [49].

Another on-device concept is called containerization which basically separates the corporate and personal data. This includes a secured enterprise browser, the isolation of data and applications and the enforcing of strong credentials. The corporate apps function independently of the mobile OS, connections to the corporate network have to be encrypted and all corporate data has to be stored in a separately encrypted storage [54].

3.4.3 Outlook

Even for the desktop systems, which have sufficient resources, the growing number and complexity of threats is staggering. A signature based approach will always be ineffective against zero-day attacks[3] [55].

A possible solution is a combination of cloud based detection and endpoint systems. Due to the resource constraint problems on mobile devices and the vast diversity of hardware and software, everything boils down to an infrastructure centric approach. This kind of system relys on a Infrastructure-Centric Security Ecosystem with Cloud Defense (ICSECD) and makes the mobile client obsolete. The ICSECD has to support in-network application analysis, context based policy definition and dynamic threat assessment. It consists of application proxies, Secure Web Gateways (SWG), data leak prevention engines, antivirus engines, antimalware engines and the functionality for secure interconnections between the components. The key to this kind of a approach is to establish a stable and intelligent application profiling mechanism [55].

[2] *"...technischen Einrichtungen, die dazu bestimmt sind, das Verhalten oder die Leistung der Arbeitnehmer zu überwachen"*

[3] An unknown security flaw, which hasn't been fixed yet.

3.5 Laws & Guidelines

There are various laws and guidelines on a national and international level providing specifications on how to handle medical data.

The Austrian Data Protection Law[4], Article 2, Part 3, §14[5] specifies basic requirements for data security in general. It states that every organizational unit has to take steps to provide data security, depending on the type, amount and purpose of the used data. The data has to be secured from loss, unlawful destruction and unauthorized access. It also has to be ensured that the data is properly used and secured considering the state of technical possibilities and the economical justifiability.

A more specific approach for the protection of medical data provides the Austrian Health Telematics Law[6]. The Health Telematics Law, Part 2, §6[7], defines the arrangements which need to be implemented to provide confidentiality to health care data in transition. The data can be handled over a network, which is protected against unauthorized access by state of the art standards, either by securing the transmission by cryptographic or structural procedures, by providing access to the network infrastructure to a limited user group, or by user authorization. Another way to secure the health care data in transit is the complete encryption of the data using cryptographic algorithms listed at §28 of the Health Telematics Law. According to §28[8] the minister of health has to consult a confirmation department[9] to define a list of cryptographic algorithms which are suitable for a state of the art encryption. The list of encryption algorithms is specified in the health telematics regulation[10] of 2013[11]. The permitted algorithms are:

- Every method described in the appendix of the signature regulation[12] of 2008[13].
- The symmetric encryption algorithms AES with 128, 192 or 256 bit or TDEA with an effective key length of at least 112 bit. Both algorithms in Cipher Block Chaining (CBC) or Counter (CTR) mode.

§8[14] of the Health Telematics Law creates a connection between the Data Protection Law and the Health Telematics Law by defining that health care providers have to keep records of all data security arrangements taken, according to §14 of the Data Protection Law[15]. These documents have to provide information about the proper handling of data access and transmission, as well as the restriction of unauthorized access.

[4] *Datenschutzgesetz*

[5] Bundesgesetz über den Schutz personenbezogener Daten (Datenschutzgesetz 2000 - DSG 2000), StF: BGBl. I Nr. 165/1999, Artikel 2, 3. Abschnitt, §14

[6] *Gesundheitstelematikgesetz*

[7] Bundesgesetz betreffend Datensicherheitsmaßnahmen bei der Verwendung elektronischer Gesundheitsdaten (Gesundheitstelematikgesetz 2012 – GTelG 2012), StF: BGBl. I Nr. 111/2012, 2. Abschnitt, §6

[8] Bundesgesetz betreffend Datensicherheitsmaßnahmen bei der Verwendung elektronischer Gesundheitsdaten (Gesundheitstelematikgesetz 2012 – GTelG 2012), StF: BGBl. I Nr. 111/2012, 5. Abschnitt, §28

[9] *Bestätigungsstelle*

[10] *Gesundheitstelematikverordnung*

[11] Verordnung des Bundesministers für Gesundheit, mit der nähere Regelungen für die Gesundheitstelematik getroffen werden – Gesundheitstelematikverordnung 2013 (GTelV 2013), StF: BGBl. II Nr. 506/2013, Anlage 2

[12] *Signaturverordnung*

[13] Verordnung des Bundeskanzlers über elektronische Signaturen (Signaturverordnung 2008 – SigV 2008), StF: BGBl. II Nr. 3/2008

[14] Bundesgesetz betreffend Datensicherheitsmaßnahmen bei der Verwendung elektronischer Gesundheitsdaten (Gesundheitstelematikgesetz 2012 – GTelG 2012), StF: BGBl. I Nr. 111/2012, 2. Abschnitt, §8

[15] Bundesgesetz über den Schutz personenbezogener Daten (Datenschutzgesetz 2000 - DSG 2000), StF: BGBl. I Nr. 165/1999, Artikel 2, 3. Abschnitt, §14

Concerning international standard procedures, IHE provides guidelines using established standards to improve the communication of digital health care systems. It is driven by health care professionals and the health care industry with the goal to create a secure and seamless way to share and access health care information. Especially the IHE Cross Document Sharing (XDS) Profile (see [40] and [41]) is relevant in relation to the medical imaging application's use cases. The XDS Profile defines standard-base document sharing of any data type. It doesn't dictate specific privacy or security policies, but facilitates the approach to a broad range of solutions.

A secure foundation for XDS can be established by the use of IHE Audit Trail and Node Authentication (ATNA). ATNA establishes security domains by enforcing security policies and procedures, data integrity, user accountability and patient information confidentiality. Data transfer within one domain may omit encryption, while the transfer across domains, depending on the policies and security mechanisms, may require encryption with Transport Layer Security (TLS). ATNA, in the context of XDS, can only be used, if each communication partner provides audit and security facilities [40].

A more detailed description of the XDS Security Environment is specified in the Appendix K of the IHE IT Infrastructure (ITI) Technical Framework (TF)-2x (see [41]). This document implies that almost all threats external to the XDS are handled by policies, technologies and agreements. The XDS Profile only provides support to control mechanisms specified by law or other regulations. Two aspects can be highlighted regarding the reference application's use cases. The Appendix K specifies cryptography by dictating that systems involved *"...shall use standard approved cryptography (methods and implementations) for key management (i.e., generation, access, distribution, destruction, handling, and storage of keys) and cryptographic services (i.e., encryption, decryption, signature, hashing, key exchange, and random number generation services)."* [41]. One of the relevant XDS Profile Component Security Objectives is the use of encrypted channels, which aren't defined by specific standards, but are generally recommended, especially if confidential data has to be transferred over a public network [41].

3.6 Conclusion

When it comes to sharing a patient's information, the patient's privacy is the main concern. It is important that all health-care, clinical and administrative personnel is thoroughly trained and involved in the mHealth process to maximize the utilization of mHealth devices [57].

All mHealth devices have to be included in the overall information security plan and robust protocols like multifactor authentication and the encryption of all data have to be implemented. If BYOD is an issue, a risk assessment has to be carried out. Then policies, which include a list of allowed protocols, continuous monitoring and the consequences of a breach, have to be developed [57].

Laws like the Data Protection Law[16] or the Health Telematics Law[17] specify a demand on how to handle health care data in general. The IHE Profiles (see [40] and [41]) provide guidelines established by professionals of the health care industry and create instruments which support the creation of secure and standardized ways of handling health care information.

The consulted experts in Leavitt's article (see [54]) agree that a multi layered approach is most efficient. The layers consist of a proper device management, containerization and the establishment and enforcement of security policies [54]. In case of malware protection, non of the current ap-

[16] *Datenschutzgesetz*
[17] *Gesundheitstelematikgesetz*

proaches has shown to be entirely effective. A non device centric but cloud based and data centric concept is the most promising one [66].

Attack vectors have readjusted and have outpaced static security mechanisms. In the future, collective intelligence and dynamic detection systems have to be utilized to tackle the pressing issue of mobile security [55].

According to a study of Ponemon in 2014 (see [67]), breaches have averaged \$3.5 million USD per incidence. This led companies to the investment in security insurance and the establishment of risk prevention strategies to decrease the impact of costs and the likelihood of breaches. These insurance policies cover issues like repairs to the company database, costs for notifying the customer and crisis management [57].

In general the approach has to be shifted from “waiting for the attack and then react”, to a more agile and dynamic method which allows to flip the equation [55].

4 iOS Security

4.1 Introduction

The usability in smartphones is embraced by the private and public sector, but for the non-consumer market, security also plays a very vital role. Although encryption is deployed in most current devices, there are a wide range of parameters, use cases and weaknesses, which need to be considered by professional security personnel [69].

This chapter mainly represents an aggregation of Apple's official ”iOS Security Guide” from April 2015 (see [10]), including findings of Teufl et al.'s “iOS Encryption Systems - Deploying iOS Devices in Security-Critical Environments” (see [69]).

Apple's iOS platform combines software features, hardware features and services to maximize security and assure a seamless user experience (see figure 4.1 on page 21). Many security features are enabled by default and cannot be configured. That way, it is assured that an extensive knowledge level is not necessary and no configuration efforts must be taken. [10].

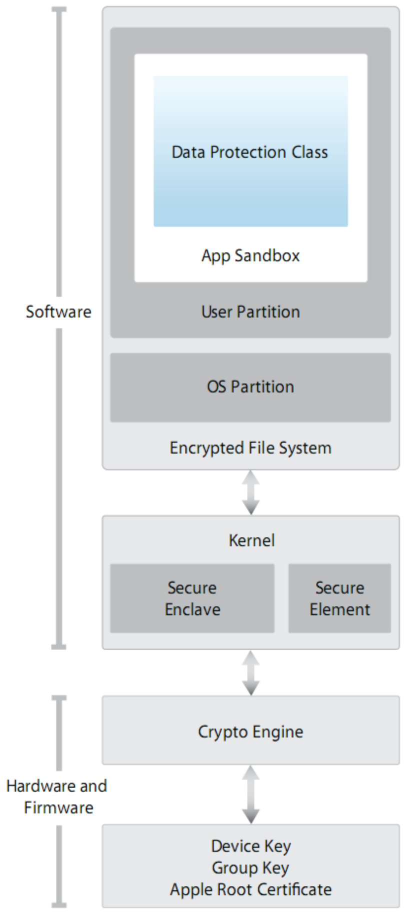


Figure 4.1: iOS architecture diagram [10]

However, different systems can be influenced by the developer and device administrator and need to be configured accurately to reduce the susceptibility to various attacks [69].

These topics are discussed in more detail in the following sections:

- **System Security:** The secure software and hardware for iOS devices.
- **Encryption and Data Protection:** The subsystem that protects user data from unauthorized access.
- **Runtime Environment Security:** The system that assures that applications run securely and without compromising the platform integrity.
- **Backups:** The difference between non-encrypted and encrypted backups.

4.2 System Security

According to the design principles of system security, software and hardware are secure across all components which include the boot-up process, software updates and the Secure Enclave. To assure optimal performance, the hardware and software are tightly integrated and the whole system is analysed from the boot-up process onwards. This allows the validation of each step and ensures that all components of the system can be trusted [10].

4.2.1 Secure Boot Chain

The components used during the start-up process consist of bootloaders, kernel, kernel extensions and baseband firmware. All these items are cryptographically signed by Apple [10].

After an iOS device is powered on, the application processor executes a code from the boot Read-Only Memory (ROM). This ROM is specified during the manufacturing process and is therefore implicitly trusted. This chip contains the Apple Root Certificate Authority (CA) that is used to verify the Low-Level Bootloader (LLB). When the LLB is trusted, it verifies the next step, the iBoot, and loads it. Afterwards the iBoot verifies and runs the iOS kernel, which concludes the boot chain validation process [10].

Devices with a cellular access also run an analogous process for the baseband subsystem. After these boot-up steps it is assured that the iOS system runs on a validated Apple device and that the low level software is not manipulated [10].

iOS devices with an A7 or later A-series processor also use a secure boot-up for their Secure Enclave to make sure that the software is signed by Apple [10].

If one step fails the next one is stopped and the device displays a ”Connect to iTunes” message. To recover the equipment, it has to be connected to iTunes and restored to factory settings [10].

4.2.2 System Software Authorization

The system software authorization architecture prevents devices from downgrading to older iOS versions that lack security updates. On iOS devices with an A7 or later A-series processor, the Secure Enclave coprocessor utilizes this process [10].

Depending on the update mechanism, the device or iTunes sends a cryptographic measurement, an anti-replay value as well as the device’s unique ID (ECID), to Apple’s Servers, for each part of the installation. During boot-up the received data is verified with Apple's signature and the device's ECID is compared with the sent ECID [10].

4.2.3 Secure Enclave

The Secure Enclave is a coprocessor present in all A7 or later A-series devices. It offers cryptographic operations for the Data Protection (see 4.3.2 on page 25) and has its own secure boot procedure. This assures the Data Protection's integrity even if the kernel is compromised [10]. This Enclave also provides a hardware random number generator and an encrypted memory. Each Secure Enclave is fabricated with its own Unique Identifier (UID) that isn't known to Apple and can not be accessed by other parts of the system [10]. During each startup, an empheral key is created and entangled with the UID. This key is used to encrypt the Secure Enclave part of the memory and every data saved by the Secure Enclave. The data storing procedure also includes an anti-replay counter [10].

Matching fingerprints via TouchID (see section 4.2.5 on page 23), enabling access to the device and purchase mechanisms are also part of the Secure Enclaves responsibilities. The whole communication between the TouchID and Secure Enclave is done over a serial peripheral interface bus and is encrypted and authenticated with a session key that is arranged with a shared key mechanism. The session key exchange utilizes AES key wrapping and uses AES-Counter with CBC-MAC (CCM) for transport encryption [10].

4.2.4 Hardware Security Features

Every iOS device contains a dedicated AES 256 crypto engine in the DMA path between the flash storage and the system memory [10].

Each device also possesses two AES 256 keys, the UID and the Group Identifier (GID) that cannot be read by any software or firmware. The UID is fused and the GID compiled into the application processor and into the Secure Enclave. These IDs are used by the Secure Enclave's AES engine to encrypt and decrypt information and only presents the result to other systems. In contrary to the UID, which is unique and not recorded by Apple, the GID is common to all processors of the same class. The GID is only used for non-security critical functions. The UID, on the other hand, enables the system to cryptographically tie data to the specific device [10].

For the creation of other cryptographic keys, the system's Random Number Generator (RNG) algorithm based on `CTR_DRBG` is used. The keys of the Secure Enclave use a hardware RNG based on multiple ring oscillators post-processed with `CTR_DRBG` [10].

Not only the creation and storage, but also the deletion of keys is important. For this purpose every iOS device employs an Effaceable Storage that accesses the underlying storage technology and removes the blocks at a very low level. The ”Erase all content and settings” function erases all keys in the Effaceable Storage which renders all user data cryptographically unattainable [10].

4.2.5 TouchID

TouchID is the name for Apple's finger print sensing system. It allows the user to choose longer and more complex passphrases and still enables a practical locking system. A passcode is mandatory to use the TouchID functionality. Under special circumstances the user is required to type in the passcode:

- When the device has just been turned on, or restarted

- When the device hasn't been unlocked for the last 48 hours
- When a remote lock command is received
- After five unsuccessful login attempts via fingerprint
- During the setup process for new fingerprints

The fingerprint sensor is only active when the capacitive steel ring detects a finger and sends a 88-by-88 pixel, 500-ppi image to the Secure Enclave. This image is temporarily stored in the encrypted memory and vectorized for further analysis. A subdermal ridge flow angle mapping algorithm is used to create an encrypted map of nodes that can only be read by the Secure Enclave [10].

Compared to the unlock process without TouchID, the Data Protection Complete Class keys (see section 4.3.2 on page 25) are not discarded from the Secure Enclave memory. With TouchID the keys are wrapped with a key, generated by the TouchID subsystem, which is only available after a correct fingerprint is provided. These keys are also deleted after five failed fingerprint login attempts, after 48 hours or if the device reboots [10].

4.3 Encryption

iOS's secure boot-chain, code signing and runtime process security are designed to allow only trusted code to be run on the user's device. Even if this part of the architecture is jeopardized, the system architecture provides additional functionality to secure the user's data. This allows the protection of corporate and private data and enables remote wiping features [10].

4.3.1 File-System Encryption

The File-System Encryption is always active and cannot be configured or deactivated by the user or a device administrator. It's not entangled with the passcode, which represents its major weakness, but provides a basic protection and is used for the remote wiping functionality [69].

The keys 0x89B and 0x835 are derived from the UID key and are used to wrap the EMF-key (file-system master encryption key) and D-key (device key) (see figure 4.2 on page 24) [69].

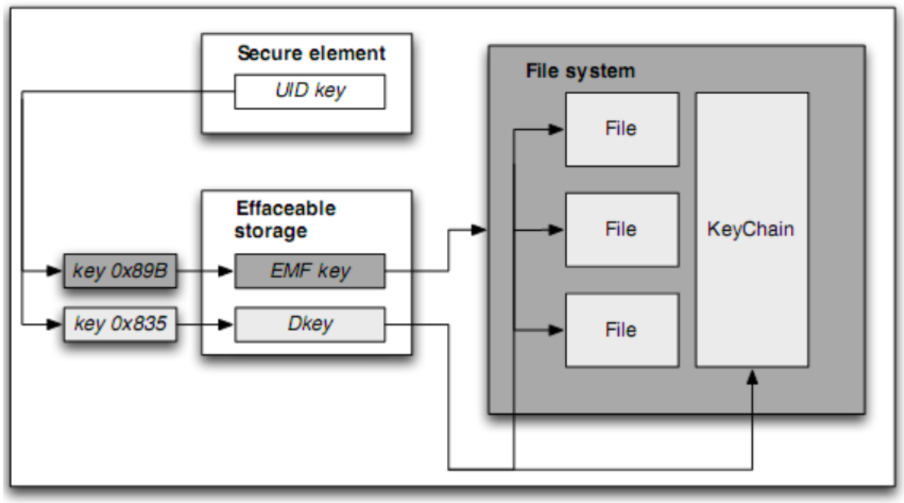


Figure 4.2: File-System Encryption overview [69]

The EMF-key is used for the File-System Encryption and the D-key for the Data Protection encryption (see section 4.3.2 on page 25). Both keys are stored in the Effaceable Storage and are inaccessible from within the file-system [69].

Key Derivation:

The user's passcode isn't involved which is the main weakness, since the security is only based on the UID [69].

Administrator/Developer:

Both need to be aware of the limited protection mechanism and that the File-System Encryption is always-on and cannot be deactivated or configured by either one [69].

4.3.2 Data Protection - File

The Data Protection uses the device's hardware encryption features and is based on a key hierarchy. Therefore each file is assigned to a class which can be controlled on a per-file basis. Depending whether the class key is unlocked, a file is accessible. This enables a high level of encryption as well as it allows the device to respond to events.

Architecture:

For each file a 256 bit per-file key is created and the AES engine encrypts the file using AES-CBC mode before storing it to the flash memory. The IV is calculated with the block offset into the file and encrypted with the Secure Hash Algorithm (SHA)-1 hash of the per-file key. After this, the per-file key is wrapped with the appropriate class key, using the National Institute of Standards and Technology (NIST) AES wrapping (see [44]) and stored in the file's metadata (see figure 4.3 on page 25). When opening a file, the metadata has to be decrypted with the file-system key, this exposes the wrapped per-file key including an information about the used class. Furthermore this key is unwrapped with the class key and used by the AES engine to decrypt the file [10]. The class keys are protected with the UID and some are also entangled with the passcode. This key hierarchy provides performance and flexibility [10].

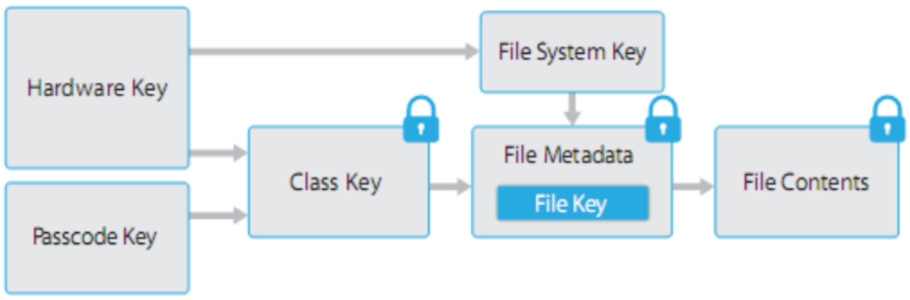


Figure 4.3: Data Protection overview [69]

Passcode:

Since the passcode provides entropy for some encryption keys, Data Protection is only enabled when a passcode is set and the stronger the passcode, the stronger the encryption. The passcode can consist of four-digits or alphanumerics of an arbitrary-length [10].

Due to the entanglement of the passcode and the UID, an off device brute force is not possible. One attempt takes about 80ms, therefore a brute-force attack, iterating all combinations, on a six-character alphanumeric passcode with lower-case letters and numbers would take about 5 1/2 years [10].

With A7 or later A-series devices, the key operations are carried out in the Secure Enclave and a five second delay between repeated failed requests is enforced. The user or the device administrator can also decide the delay length between the failed attempts and whether a device should automatically be wiped after 10 consecutive failed attempts [10].

Protection Classes:

Depending on the used class, different policies apply. Each class has its own key that is used to encrypt and decrypt the unique per-file key [69].

- **No Protection:** Files are only protected by the File-System Encryption [69].
- **Complete Protection:** The class and the per-file key are removed when the file is locked [69]. This class key is protected by the user's passcode and the UID [10].
- **Protected Until First User Authentication:** The class key is decrypted when the device is unlocked the first time after the boot [69]. This is the default class for third-party application data [10].
- **Protected Unless Open:** Data can be written even when the device is locked. An asymmetric elliptic curve cryptography (ECDH over Curve 25519) is used for this method [10]. This class is intended to be used in cases like downloading an e-mail attachment [69].

Key Derivation:

Data Protection uses the PBKDF2 as key derivation function, which is specified in PKCS #5 (see [43]). This function includes the user's passcode, tangled with the UID, a salt and a high iteration count [69].

Developer:

The developer has to choose the appropriate class for the stored data. Data Protection can only be effective when the developer has accurate knowledge of the functionality [69].

Administrator:

The only way to influence the security of Data Protection as an administrator is the enforcement of passcode policies, since the Data Protection's efficiency depends on the entropy strength derived from the user's passcode. Unfortunately there is no prefabricated way to determine the used protection classes as an administrator, although this is very important in detecting whether an application fulfils the defined security requirements [69].

4.3.3 Data Protection - Keychain

The keychain is a SQLite database which allows to store passwords and sensitive data like keys, certificates and login tokens in a secure way. The API calls the securityd daemon which decides whether access to certain items is granted [10].

The Data Protection is similar to the one for files, although there is a difference in available protection classes [69].

Protection Classes:

- **Accessible Always:** like File Protection class "No Protection" [69].
- **Accessible When Unlocked:** like File Protection class "Complete Protection" [69].
- **Accessible After First Unlock:** like File Protection class "Protected Until First User Authentication" [69].
- **Accessible When Passcode Set This Device Only:** Available only when a passcode is set and the items don't sync to the iCloud keychain [10].

All keychain protection classes exist in a "ThisDeviceOnly" version, which doesn't allow a transfer of these protected items off-device. The File Protection class "Protected Unless Open" also doesn't exist for keychain items [69].

Key Derivation:

The same as "Data Protection - File" (see 4.3.2 on page 26) [69].

Administrator/Developer:

Similar to "Data Protection - File" (see 4.3.2 on page 26), plus the developer can decide whether a keychain item is transferable to another device or not [69].

4.3.4 Keybag

iOS uses four different keybags to store and manage keys [10].

System keybag:

This keybag stores the wrapped class keys [10].

Backup keybag:

This keybag is created when an encrypted iTunes backup is used. It holds keys to re-encrypt the backuped-up data [10].

Escrow keybag:

The escrow keybag is used for iTunes sync and MDM. A MDM can remote-clear the passcode and an iTunes backup is possible without entering the passcode. The keybag is split and the parts are stored on the device and the computer [10].

iCloud keybag:

It is similar to the backup keybag, but it is asymmetrically encrypted and allows iCloud backups in the background. The class keys are wrapped with the device UID, like an encrypted iTunes backup [10].

4.4 Runtime Process Security

Third party applications use unique home directories and are “sandboxed” on iOS operating systems. These kind of applications are separated from system files and system applications and can only access outside information via services provided by the operating system [10].

In addition, the operating system partition is mounted in read-only mode and the main part of the iOS system runs as non-privileged user “mobile”. User information can only be accessed via declared entitlements that are used by daemons and system apps [10].

XCode, the iOS IDE, compiles applications with Adress Space Layout Randomization (ASLR) support by default. This feature helps to protect against memory corruption exploits. ARM’s Execute Never (XN) feature makes sure that memory pages are non-executable. Under controlled conditions apps can write to memory pages, but only a single mmap-call can be made [10].

4.5 Backups

iOS differentiates between plaintext and encrypted backups. Backups can be stored to a device via iTunes or to an iCloud account. Files are marked for backup by default and only the developer can change this setting [69].

4.5.1 Standard Backup

All files within a standard backup are encrypted with Data Protection, therefore iTunes cannot access the content of these files without the corresponding protection class keys. At the first connection to iTunes an escrow keybag (see 4.3.4 on page 27) is created that contains all protection class keys encrypted with a random key. This random key is stored on the iOS device with the protection class “Accessible After First Unlock This Device Only”. This allows iTunes to decrypt the escrow keybag and further gain access to the protection class keys, decrypt the files and copy them to the backup location. Keychain items, on the other hand, remain encrypted with the protection class keys and can not be decrypted without access to the iOS device (see figure 4.4 on page 29) [69].

Standard Backup - Key Derivation:

No key derivation is involved in the standard backup process.

4.5.2 Encrypted Backup

For an encrypted backup a backup password has to be defined. This password is used to derive an encryption key to encrypt the files and the iOS keychain. Altough the files are better protected, since they are encrypted with the backup password key, encrypted backups have some drawbacks for the keychain’s security [69]. During an encrypted backup the device’s system keybag is decrypted and encrypted with the backup password key. This allows the transfer and decryption of the system keybag off-device and the strength of the encryption depends on the backup password only (see figure 4.4 on page 29) [69].

Encrypted Backup - Key Derivation:

The derivation of the backup password key is based on the PBKDF2 function. Due to the lack of a Secure Enclave on the backup device, the derivation is based only on the backup password and not entangled to the iOS device.

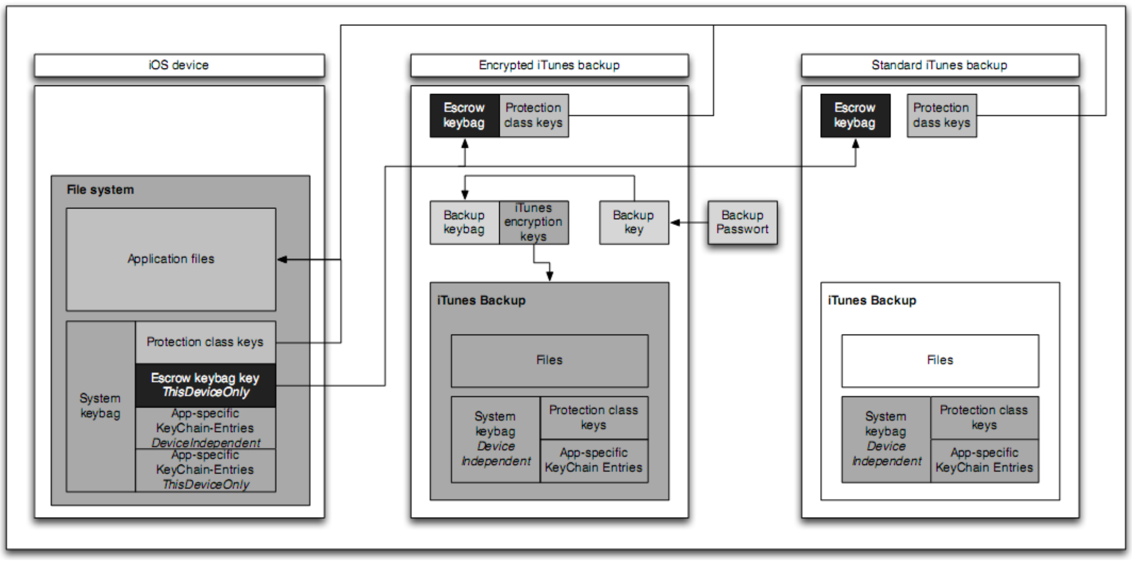


Figure 4.4: Encrypted and standard backup overview [69]

General - Developer:

Files are marked for backup by default. A developer has to decide whether files are security critical and should be excluded from backups.

General - Administrator:

In general backups can be configured using the iTunes configuration. In a managed setup an administrator cannot disable backups but he can enforce encrypted backups. Nevertheless the administrator can not enforce policies regarding the backup password, which is the main factor for the strength of the backup's encryption.

5 Threat Modeling

This chapter describes the threat modeling process for the mobile skin imaging application. The threat modeling aproach is based on the Open Web Application Security Project (OWASP) *"Mobile Threat Model Project"* (see [30]) and is divided into five different areas:

Mobile Application Architecture - Defines the device features used by the application;

Mobile Data - Illustrates the data processed and stored throughout the application's work flow, including flow diagrams, obtained from the app's project documentation;

Threat Agent Identification - Identifies potential threat agents regarding the application;

Methods of Attack - STRIDE (Spoofing identity - Tampering with data - Repudiation - Information disclosure - Denial of service - Elevation of privilege) threat modeling mechanisms are used to identify threats to the applications environment;

Controls - Based on STRIDE, control mechanisms are highlighted to prevent or mitigate the identified threats.

The gathered knowledge and control mechanisms are incorporated into the design process of the cryptographic concept and encryption architecture, described in chapter 6 on page 40.

5.1 Use Case

The application is part of a medical skin imaging and analysis software which consists of the mobile iOS application and a web application. The threat modeling process focuses only on the mobile part of the software suite.

The smartphone application is utilized to acquire and upload patient image data. It specifically allows the study team to scan QR-codes for patient identification, to use the device camera to collect patient images and to upload the acquired images to the web application.

The mobile software is developed during a clinical phase I. - trial in cancer patients suffering from radiation induced dermatitis (RID) and intended to be classified as class *I. measure (I.m)* medical product based on the guidance document for "Classification of medical devices" by the European Commission [31].

Due to the possibility of lack of connectivity to the server the recorded images have to be persisted to the device storage and are therefore exposed to threats like malware, unintended data leakage, jailbreaking and other risks.

5.2 Scope of Work

The main task is to create and implement a cryptographic concept which secures the mobile iOS applications data on the device and during the transmission to the server even if the operating systems native security mechanisms are broken [11].

A concept will be designed considering cryptographic theories, state of the art standards and the operating system's hard- and software limits.

The thesis answers the question whether application data can be secured even if the smartphones operating systems native security mechanisms are broken. It also shows if data is still safe when the smartphones rights management is bypassed by jailbreaking/rooting. Furthermore the connection between the smartphone and the server is reviewed whether the data is secure if the connection is compromised.

At first, the architecture is designed, then the designed concept is implemented and the assumptions are verified. In the thesis the results are presented and substantiated with the corresponding theoretical background.

The goal of this prototype is to use the iOS APIs cryptographic functionality to its full extent and to create a prototype which can be integrated into the described medical image analysis platforms architecture.

5.3 Threat Model

5.3.1 Introduction Statement

"Threat modeling is a systematic process that begins with a clear understanding of the system. It is necessary to define the following areas to understand possible threats to the application." [30]

Mobile Application Architecture - The first section is used to outline the device features used by the application. The features range from transmission protocols and mediums to interaction with other applications and hardware components [30].

Mobile Data - This area describes which data is processed and stored by the application. Furthermore the purpose and workflow of this data are identified [30].

Threat Agent Identification - In this section the threats and threat agents to the application are identified [30].

Methods of Attack - This chapter lists possible attack methods for the application. The attack methods are defined in order to develop control mechanisms which mitigate the attacks [30].

Controls - The last area specifies control mechanisms to prevent or diminish the listed attack methods [30].

The following threat model approach is based on the OWASP *"Mobile Threat Model Project"* (see [30]) which is part of the OWASP *"Mobile Security Project"* (see [30]).

5.3.2 Mobile Application Architecture

Design of the Architecture

- Carrier: Data
- Endpoints: RESTful Web Services
- Wireless interfaces: 802.11 and Cellular Network
- App Layer: Hypertext Transfer Protocol (HTTP)
- Runtime Environment (VM, framework dependencies, etc):
  - Open Source Computer Vision library (opencv) - for image processing
  - XSWI project XML writer
  - MBProgressHUD library for visual user feedback
- OS Platform: iOS 8.0 and higher

Hardware Components

The used hardware components of the mobile device comprise of the following features:

- Cellular Radios (GSM/CDMA/LTE): Connection to the server
- Flash Memory: Storage
- Wireless Interfaces - 802.11: Connection to the server
- Touch Screen: User input
- Camera: Capturing images

Authentication

The authentication is password based and used to authenticate against the RESTful Web Service. The decision process takes place on the remote server.

5.3.3 Mobile Data

Business Function of the Application

- Scan patient QR-codes for convenient patient identification
- Acquire patient images for documentation using the device's camera app
- Upload patient images to server

Data stored/processed by the Application

The following flow diagrams are taken from the mobile app's software specification and describe the data flow of the mobile application.

Figure 5.1 on page 33 shows the basic work flow including scanning the patients QR-code and the image capturing.

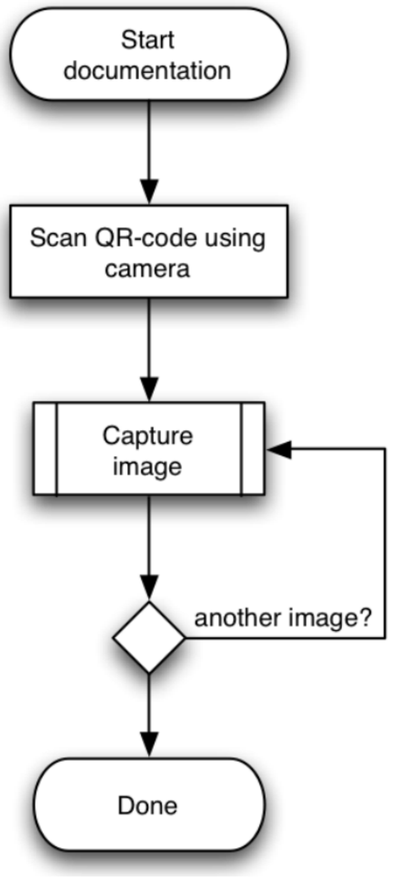


Figure 5.1: Process work flow

The detailed image capturing work flow is shown in figure 5.2 on page 34. After the user opens the camera, the application automatically scans for the marker which is used for the white balancing of the image. If the marker is detected, the “capture image” button is enabled. After the image is taken, the image is transformed to match colors and the user can decide whether he wants to save the image.

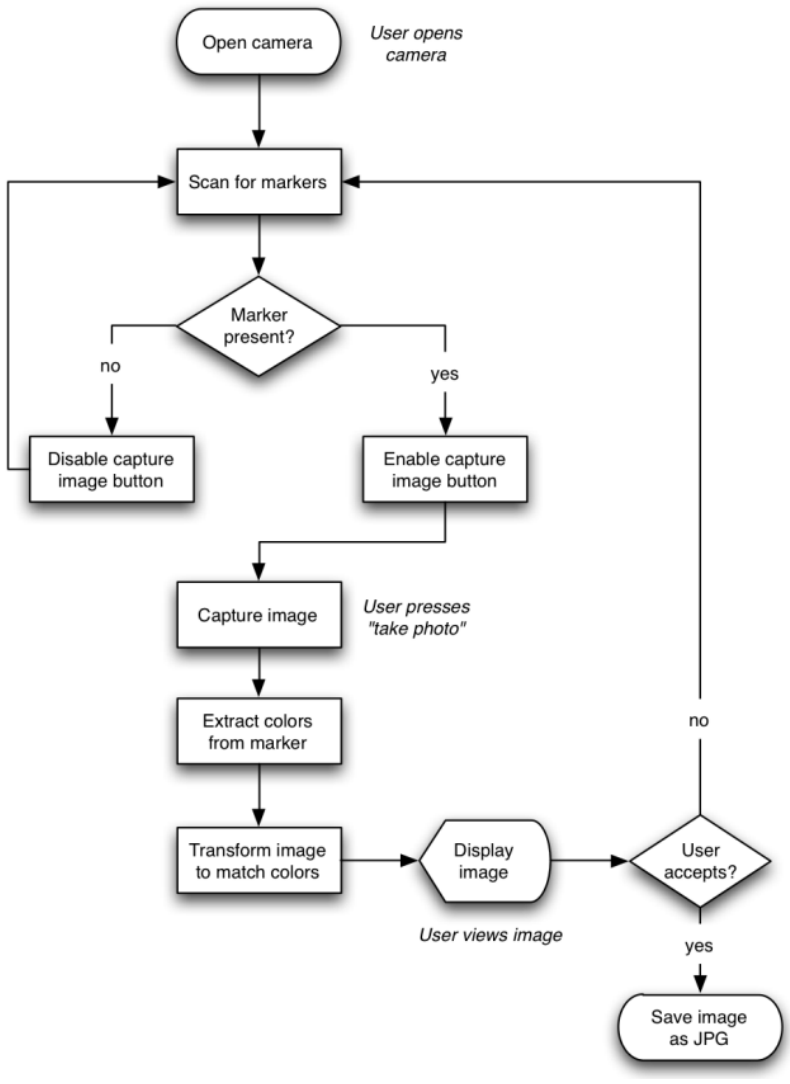


Figure 5.2: Capture image work flow

The stored images can be uploaded to the backend, which is illustrated in figure 5.3 on page 35. After opening the upload view controller the application verifies whether an active session is available. If no active session is found and the user presses the “start upload” button, the user is queried for his login credentials. Then all queued images are uploaded, depending whether the user has permission for the trial or not. The user has the possibility to provide other credentials for the trial or to skip the trial specific images and upload all not skipped data.

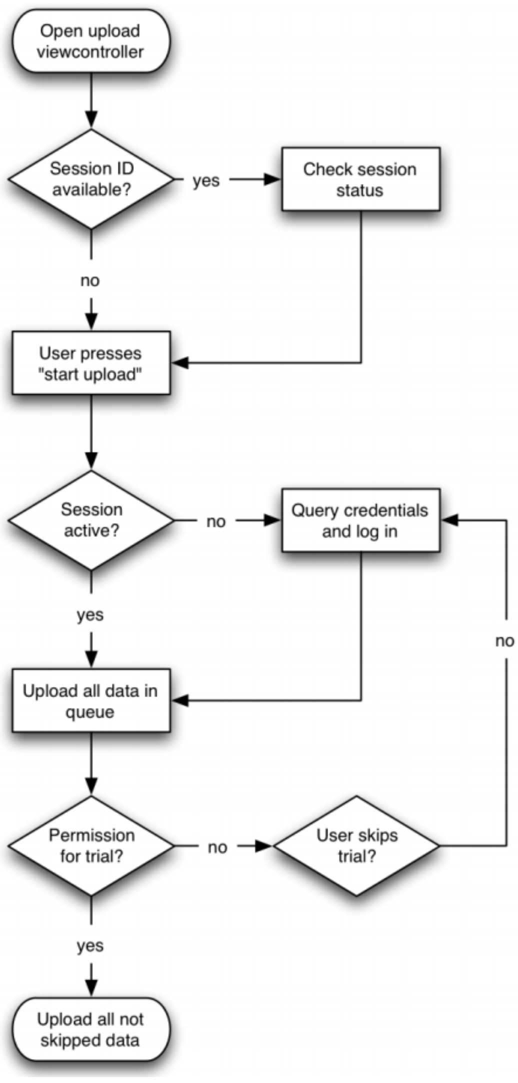


Figure 5.3: Upload images work flow

How is data transmitted between third party API’s and app(s)?

There are no third party APIs involved.

Are there different data handling requirements between different mobile platforms?

The app will be available for iOS only.

Does the app use cloud storage APIs (Dropbox, Google Drive, iCloud, Lookout) for device data backups?

No.

Does personal data intermingle with corporate data?

No.

Is there specific business logic built into the app to process the data?

The opencv library is used to perform a white balancing on the recorded images.

Data Values

Data at rest and in transit:

- Images of the patient's body parts/skin. The patient's face may also be visible.
- Session cookies for the web service authentication
- Documentation-Universally Unique Identifier (UUID)
- Trial-UUID
- Patient-ID
- Date
- Image comment

Third Party Data

What are the privacy requirements of user data?

The images show the patients bare skin from different body parts and may also display the patients face or other body parts by which the patient may be identified. However no basic claims data, like name or social security number, is stored with the images since the patient has to remain anonymous throughout the process of the clinical study.

Are there regulatory requirements to meet specific user privacy?

Since the patient remains anonymous throughout the clinical trial there are no regulatory requirements regarding the user privacy.

Impact on the Application by other Data stored on the Device

The app should not share any data with other installed apps on the device. Since the app uses the device's camera, extraordinary diligence should be exercised to prevent the images from showing in the device's gallery application.

Impact of jailbroken vs. non jailbroken Device and the Affects on Application Data

Since it is possible to access any kind of data stored on a jailbroken device, the data has to be encrypted before it is stored to the device's memory. Also, the encryption key has to be secured on the device. For example by a password based key derivation function.

5.3.4 Threat Agent Identification

The threat agents listed below are common threat agents identified by the OWASP *"Mobile Threat Model Project"* (see [30]).

Human Interaction

- **Stolen Device User**: A user who obtained unauthorized access to the device and who is trying to access sensitive data on the owner's device.
- **Owner of the Device**: A malicious application which is unwillingly installed by the owner.
- **Common WiFi Network User**: This user sniffs the WiFi network used by a victim. This agent collects all the data transmitted by the victim's device and may re-use it to launch further attacks.
- **Malicious Developer**: A malicious user who writes an application which obtains information from the owner's device and sends it back to the malicious user. It is also possible that the application is using all the device resources to perform a denial of service on the device.
- **Organization Internal Employees**: A user who is a member of the company and has the privileges to submit changes to the application.

Automated Programs

- **Malware on the device**: An application which performs malicious activities. It runs in the background parallel to other applications and, for example, sends sensitive user data to a remote server [30].

  If the application is distributed via the official Apple app store [1] it has to undergo a review process. Apple provides "App Store Review Guidelines" [2] to help the developer steer clear of issues which could prevent the app from being submitted to the app store. This process focuses on preventing malware applications being distributed through the Apple app store.
- **Scripts executing at the browser with HTML5**: A script which can access the device on content level. It may run in the browser and transmits browser data or complete device level data [30].

[1] https://itunes.apple.com/en/genre/ios/id36?mt=8 - Apple Inc. - App Store Downloads on iTunes (visited on 10/10/2014)

[2] https://developer.apple.com/app-store/review/guidelines/ - Apple Inc. - App Store Review Guidelines (visited on 10/10/2014)

5.3.5 Methods of Attack

Threat Identification

To identify possible threats the STRIDE threat modelling system is used. It provides a number of categories with corresponding counter measurements (see table 5.1 on page 38) [71].

Type	Description	Security Control
Spoofing	attackers pretend to be someone (or something) else	Authentication
Tampering	attackers change data in transit or at rest	Integrity
Repudiation	attackers perform actions that can't be traced to them	Non-Repudiation
Information disclosure	attackers steal data in transit or at rest	Confidentiality
Denial of service	attackers interrupt a system's legitimate operation	Availability
Elevation of privilege	attackers perform actions they aren't authorized to perform	Authorization

Table 5.1: STRIDE categories [71]

Furthermore every object in the data flow diagrams (see figure 5.1 on page 33, figure 5.2 on page 34 and figure 5.3 on page 35) is analysed considering potential threat categories of the STRIDE model resulting in a list of attack methods [71]. The following methods consist of example attack methods of the "Mobile Application Threat Model" (see [30]) and the "OWASP Mobile Security Project - Top Ten Mobile Risks" (see [37]).

Carrier based methods:

C1. **Man in the middle attack**: targets transmitted data packets

C2. **Hijacking** of the wireless transmission

C3. **SSL proxying** due to lack of certificate inspection

C4. Exploitation of **weak cyphersuits** as a result of weak handshake negotiation

C5. **Jeopardizing of transmitted data** over an unsecured, non-SSL data channel

Endpoint based methods:

E1. Stealing sensitive content using **malware**

E2. Accessing **insecure data storage**

E3. Decrypting insecure or **deprecated cryptography** algorithms: Many cryptographic algorithms and protocols should not be used because they have been shown to have significant weaknesses or are otherwise insufficient for modern security requirements

OS and application level methods:

O1. **Unintended data leakage**: An agent that has access to the device via malicious code will use fully permissible and documented API calls to conduct this attack

O2. **Client side injection**: SQLite Injection, JavaScript Injection, Extensible Markup Language (XML) Injection, Format String Injection

O3. **Session(-Cookie) Hijacking**

O4. **Reverse Engineering**

O5. **Jailbreaking** the phone to access sensitive data from memory

O6. **Stealing keyboard cache** or logs

Once all data flows are analyzed each threat and attack method is revisited and it is determined how to resolve or prevent these threats (see table 5.2 on page 39) [71].

Attack	STRIDE Type	Security Control
C1. MITM attack	Tampering	Integrity
C2. Wireless hijacking	Tampering	Integrity
C3. SSL proxying	Tampering	Integrity
C4. Exploitation of weak cyphersuits	Information disclosure	Confidentiality
C5. Jeopardizing of insecurely transmitted data	Information disclosure	Confidentiality
E1. Stealing sensitive content	Information disclosure	Confidentiality
E2. Accessing insecure data storage	Information disclosure	Confidentiality
E3. Decrypting insecure cryptography algorithms	Information disclosure	Confidentiality
O1. Unintended data leakage	Information disclosure	Confidentiality
O2. Client side injection	Tampering	Integrity
O3. Session(-Cookie) Hijacking	Elevation of privilege	Authorization
	Spoofing	Authentication
O4. Reverse Engineering	Tampering	Integrity
	Information disclosure	Confidentiality
O5. Jailbreaking	Elevation of privilege	Authorization
	Information disclosure	Confidentiality
O6. Stealing keyboard cache	Information disclosure	Confidentiality

Table 5.2: Mapping of attack to security control mechanism

The implementation of the security controls are described in the architectural concept including references to the attack method IDs (see section 6.3 on page 50).

6 Cryptographic Concept

After completing the threat model and identifying possible attacks and counter measurements a cryptographic concept and an encryption architecture are designed.

For a better understanding a terminology is defined, describing the purpose, the location and the properties of each cryptographic component. Cryptographic work flows are created, based on the use cases extracted from the medical skin imaging mobile application. Each use case consists of a flow diagram and a step-by-step description of the process and the components involved. Additionally, an architectural snapshot is presented and practical implementations of control mechanisms, based on the threats and controls identified in chapter 5.3.5 on page 38, are outlined.

6.1 Terminology

The following table (see table 6.1 on page 41) gives an overview of the keys, salts, passwords and certificates used in the proposed concept. Every item is defined by a name, a type, a short description and a location. The location shows which component of the architecture has access to the corresponding information.

Name	Type	Description	Location
M-key	AES-256	master-key	client and server
HM-key	HMAC SHA-256	master HMAC-key	client and server
MHM-key	AES-256 + HMAC SHA-256	concatenated M-key with HM-key	client and server
CPR-key	RSA-4096	client private-key	client
CPUB-key	RSA-4096	client public-key	client and server
SPR-key	RSA-4096	server private-key	server
SPUB-key	RSA-4096	server public-key	client and server
KDF-key	AES-256	PBKDF2 generated AES-key	client
HKDF-key	HMAC SHA-256	PBKDF2 generated MAC-key	client
KDF-salt	8 Bytes	salt for PBKDF2	client
HKDF-salt	8 Bytes	salt for PBKDF2	client
App password	minimum of six characters	application password	client
IV	16 Bytes	random initialization vector per picture	client and server
PIN-cert	X509, DER format	root-cert for SSL pinning	client

Table 6.1: Terminology overview

Each cryptographic component is defined by its purpose in the global concept.

- **M-key:** Encryption and decryption of the pictures and the clients private key.
- **HM-key:** Message authentication code after the encryption of the pictures and the clients private key with the M-Key.
- **MHM-key:** Concatenation of the two master keys, the M-key and the HM-key.
- **CPR-key:** Decryption of the servers public key during the initial key exchange and signing of the MHM-key and the pictures.
- **CPUB-key:** Checking the signature of the signed pictures and the signed MHM-key.

- **SPR-key:** Decryption of the client public key and decryption of the MHM-key during the key exchange.
- **SPUB-key:** Encryption of the clients public key and encryption of the MHM-key during key exchange.
- **KDF-key:** Encryption of the MHM-Key. It is created with the PBKDF2-function, KDF-salt and the app password.
- **HKDF-key:** MAC of the encrypted MHM-key. It is created with the PBKDF2-function, HKDF-salt and the app password.
- **KDF-salt:** Salt for PBKDF2 to generate the KDF-key.
- **HKDF-salt:** Salt for PBKDF2 to generate the HKDF-key.
- **App password:** Application password used for PBKDF2 functions.
- **IV:** Random initialization vector for the M-key and for each separate image.
- **PIN-cert:** Client root certificate for SSL pinning.

For a better understanding four prefixes are used in the explanation of the architecture, which can occur in connection to a cryptographic item.

- **mac-:** The message authentication code for the corresponding data is calculated.
- **enc-:** The corresponding data is encrypted with a symmetric encryption key.
- **sign-:** The corresponding data is signed with a private key.
- **asymenc-:** The corresponding data is encrypted with a public key.

6.2 Cryptographic Workflow

During the initial setup of the app a password (*app password*) is set. Furthermore an AES-256 key (*M-key*), a SHA-256 HMAC key (*HM-key*) and a RSA-4096 key pair (*CPR-key & CPUB-key*) are generated.

The PBKDF2-function with 10.000 iterations, salt (different salt for each key) and the app password as parameters are used to generate an AES-256 key (*KDF-key*) and a SHA-256 HMAC key (*HKDF-key*). The *M-key* and the *HM-key* are concatenated (*MHM-key*) and encrypted with the *KDF-key* and maced with the *HKDF-key*. The encrypted and maced *MHM-key* (*mac-enc-MHM-key*) and both PBKDF2 salts (*KDF-salt, HKDF-salt*) are stored to the device memory.

In the course of every *app password* change (or initialization) the dependant keys (*KDF-key, HKDF-key*) and salts (*KDF-salt, HKDF-salt*) are also renewed and therefore the *MHM-key* is re-encrypted with the new set of keys and again stored to the device memory.

The clients RSA private key (*CPR-key*) is encrypted with the M-key and maced with the *HM-key* (*mac-enc-CPR-key*) and also stored to the device memory.

The master keys (*M-key, HM-key*) are only available if the user supplies the correct application password (*app password*) and therefore submits the valid parameters to the PBKDF2-function.

When the user closes or locks the app, the keys are removed from the devices random access memory, hence no data which is encrypted with the master keys (*M-key, HM-key*) can be decrypted.

For an illustration of the initial setup workflow, see figure 6.1 on page 43.

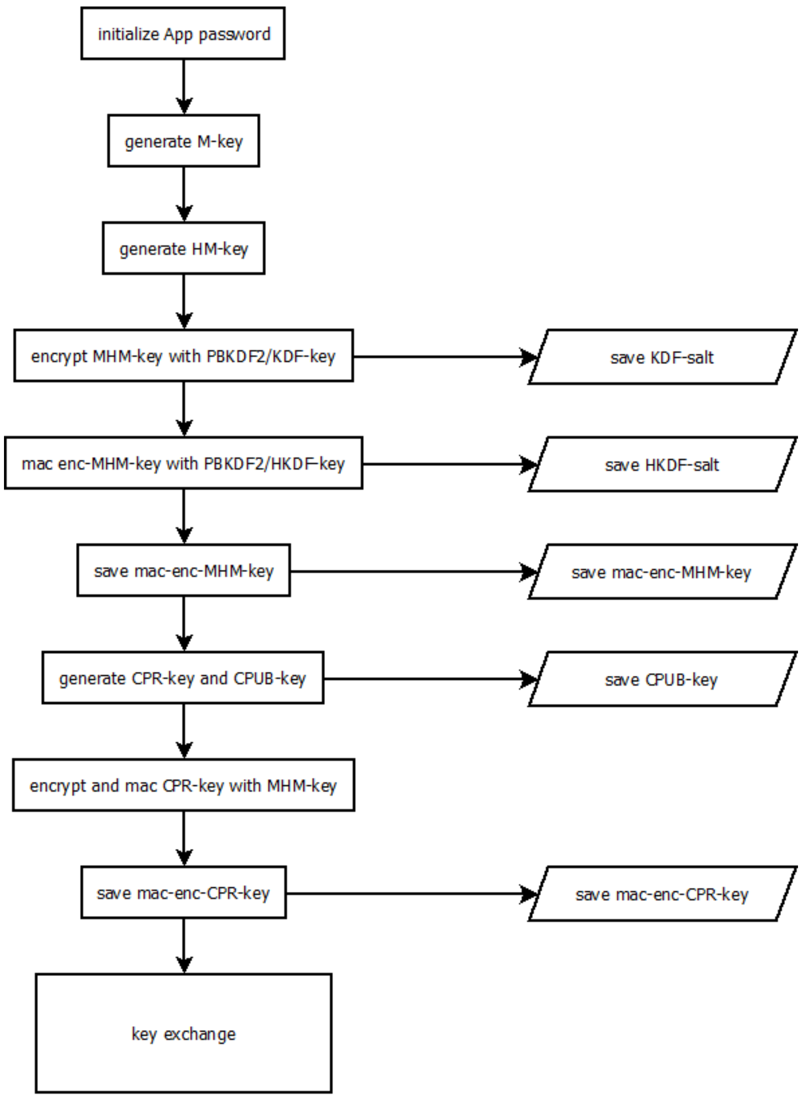


Figure 6.1: Initial Setup

A crucial part of the initial setup is the key exchange (see figure 6.2 on page 45). During the key exchange a connection between the client and the server is mandatory.

The server's RSA public key (*SPUB-key*) which is included in the application package is utilized to encrypt the clients RSA public key before sending it to the server. The server decrypts the encrypted *CPUB-key* (*asymenc-CPUB-key*) with its own private RSA key (*SPR-key*). In the next step the server encrypts its RSA public key (*SPUB-key*) with the clients RSA public key (*CPUB-key*) and sends the encrypted *SPUB-key* (*asymenc-SPUB-key*) to the client.

The iOS client decrypts the encrypted *SPUB-key* (*asymenc-SPUB-key*) with the RSA private key (*CPR-key*) and compares the received *SPUB-key* with the local *SPUB-key* from the application package. If both keys match, the client continues with the key exchange and encrypts the concatenated *M-key* and *HM-key* (*MHM-key*) with the servers RSA public key (*SPUB-key*). Before sending the encrypted *MHM-key* (*asymenc-MHM-key*) to the server, the client signs the key with the client RSA private key (*CPR-key*).

After receiving the signed and encrypted *MHM-key* (*sign-asymenc-MHM-key*) from the client, the server checks the signature with the clients RSA public key (*CPUB-key*). If the signature is valid the *MHM-key* is decrypted with the server RSA private key (*SPR-key*) and saved to the servers storage.

After successfully completing the process, the server sends a positive key exchange message to the client. The client accepts the message and checks whether the key exchange was successful or not. If successful, the client stores a key exchange timestamp.

A renewal of the devices cryptographic key set (*M-key, HM-key, CPR-key & CPUB-key*) renders all previously taken pictures invalid since the old cryptographic keys are all deleted beyond recall. Every key renewal initiates an online key exchange with the server to guarantee that the server can decrypt new pictures. Only if the key exchange was successful the clients key set is updated.

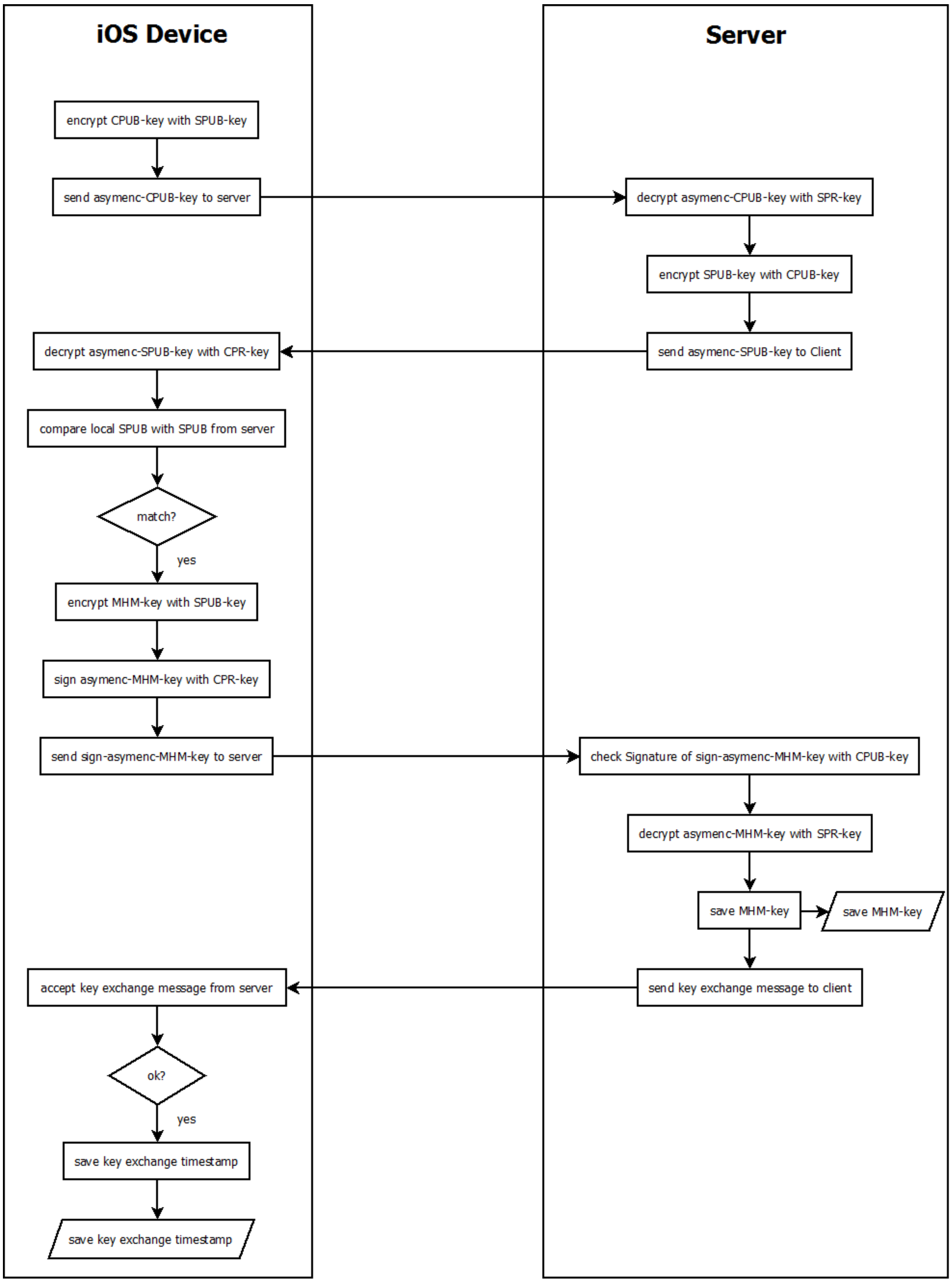

iOS Device
Server
encrypt CPUB-key with SPUB-key
send asymenc-CPUB-key to server
decrypt asymenc-CPUB-key with SPR-key
encrypt SPUB-key with CPUB-key
send asymenc-SPUB-key to Client
decrypt asymenc-SPUB-key with CPR-key
compare local SPUB with SPUB from server
match?
yes
encrypt MHM-key with SPUB-key
sign asymenc-MHM-key with CPR-key
send sign-asymenc-MHM-key to server
check Signature of sign-asymenc-MHM-key with CPUB-key
decrypt asymenc-MHM-key with SPR-key
save MHM-key
save MHM-key
send key exchange message to client
accept key exchange message from server
ok?
yes
save key exchange timestamp
save key exchange timestamp

Figure 6.2: Key exchange

After a picture was successfully taken, a random initialization vector (*IV*) is generated. Before storing the picture to the device, the *M-key* and the *IV* are used to encrypt the picture and the *HM-key* to generate the message authentication code for the picture. After that the maced and encrypted picture (*mac-enc-picture*) is signed with the clients RSA private key (*CPR-key*), the individual (*IV*) is stored in the images header and the image is saved to the device storage (see figure 6.3 on page 46).

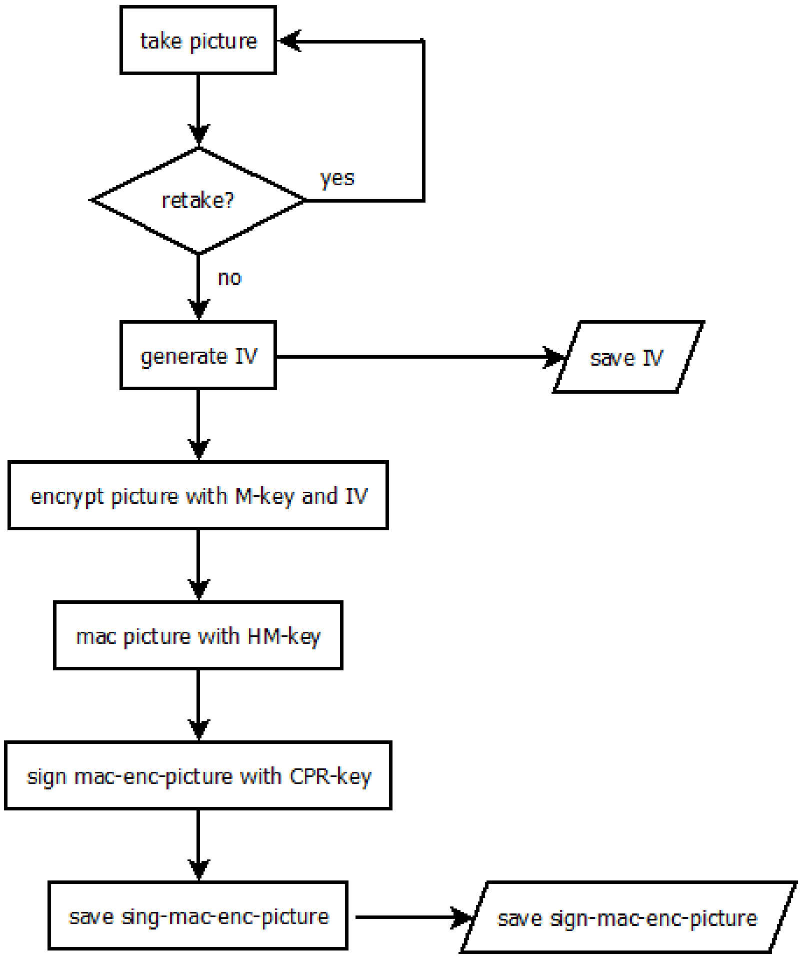

take picture
retake?
yes
no
generate IV
save IV
encrypt picture with M-key and IV
mac picture with HM-key
sign mac-enc-picture with CPR-key
save sing-mac-enc-picture
save sign-mac-enc-picture

Figure 6.3: Image saving

To upload the encrypted pictures stored on the device an active session is required. If no session is active, the users login credentials for the server are queried. Right after submitting valid credentials the server checks whether the user has permission to submit pictures to the trial. The user may skip all pictures of trials for which he has no permission. The rest of the not skipped, signed, maced and encrypted pictures (*sign-mac-enc-pictures*) are uploaded to the server (see figure 6.4 on page 47).

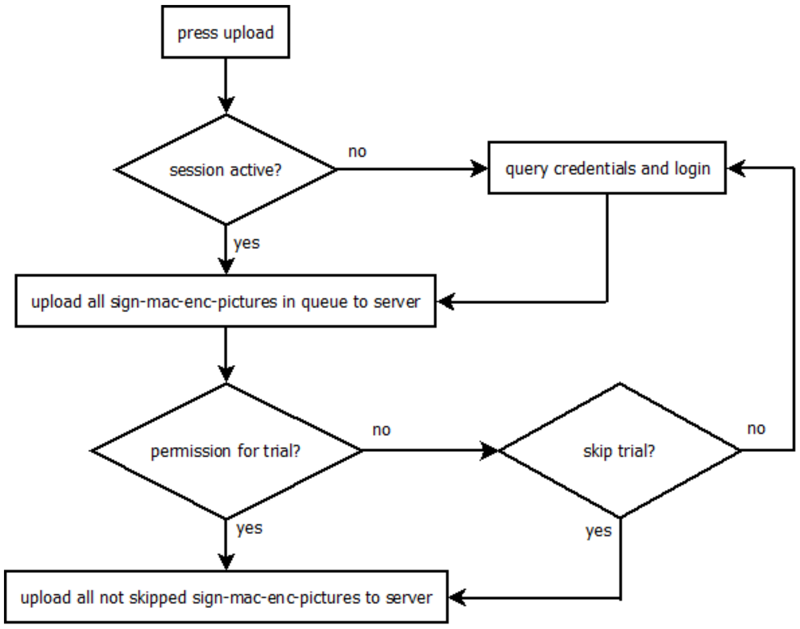

press upload
session active?
no
query credentials and login
yes
upload all sign-mac-enc-pictures in queue to server
permission for trial?
no
skip trial?
no
yes
yes
upload all not skipped sign-mac-enc-pictures to server

Figure 6.4: Image uploading

The following figures (see figure 6.5 on page 48 and figure 6.6 on page 49) give an architectural snapshot of a device state, a server state and the transmission of a picture.

The iOS device contains the *M-key* and the *HM-key* encrypted with the *KDF-key* and maced with the *HKDF-key*. Besides the clients RSA private key (*CPR-key*) which is encrypted with the *M-key*, the rest of the keys, the *CPUB-key* and the *SPUB-key*, and the *PIN-cert* certificate are not encrypted. The client side also illustrates a stack of pictures which are all separately encrypted with the *M-key*, maced with the *HM-key* and signed with the *CPR-key*.

The server holds its own RSA key pair (*SPR-key* & *SPUB-key*) and for each iOS device a separate container with its corresponding *M-key*, *HM-key* and *CPUB-key*.

The pictures are transmitted over a TLS secured HTTP channel with a strict set of cypher suits. TLS in version 1.2 is mandatory. The key exchange mechanism is Elliptic-Curve Diffie-Hellman (ECDHE). The authentication mechanism utilizes RSA certificates and the encryption cypher is AES-256 in Galois/Counter Mode (GCM) which also takes care of the authentication. In addition, the client implements certificate pinning with the *PIN-cert*. All uploaded images are encrypted with the *M-key*, maced with the *HM-key*, signed with the *CPR-key* and transmitted over the TLS secured HTTP channel.

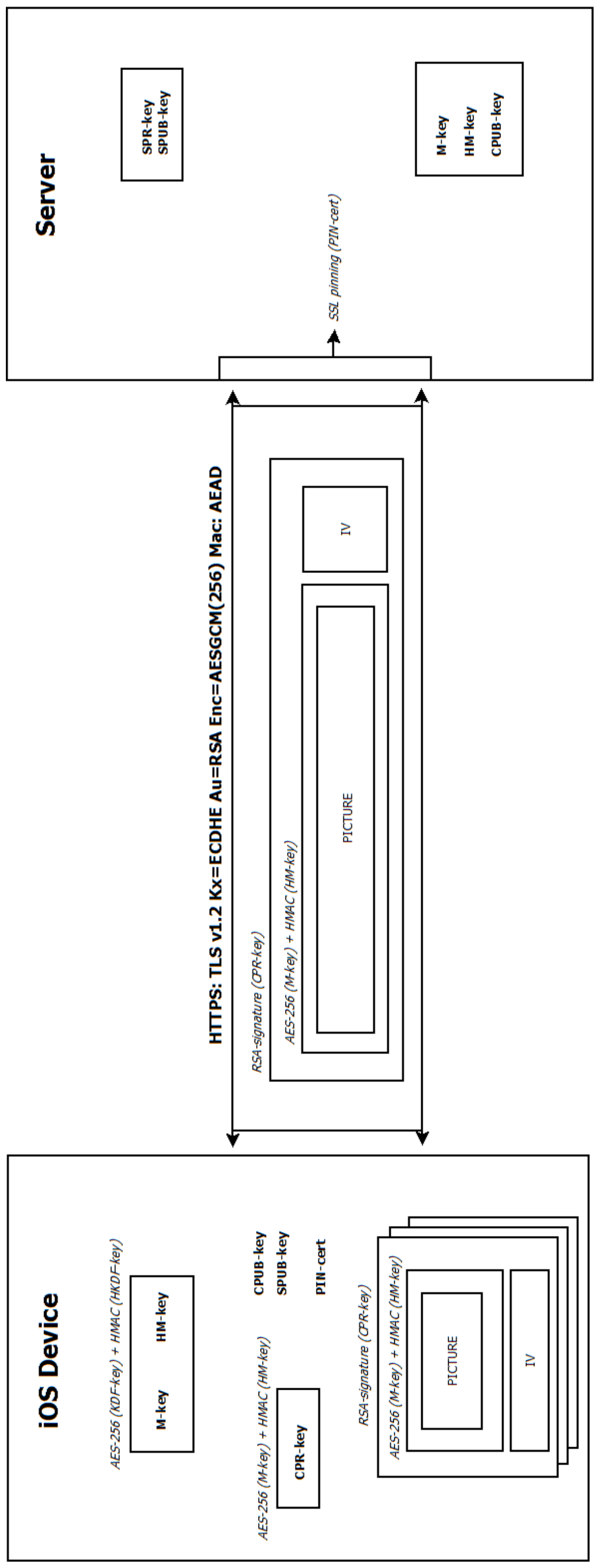

Server
SPR-key
SPUB-key
M-key
HM-key
CPUB-key
SSL pinning (PIN-cert)
HTTPS: TLS v1.2 Kx=ECDHE Au=RSA Enc=AESGCM(256) Mac: AEAD
RSA-signature (CPR-key)
AES-256 (M-key) + HMAC (HM-key)
IV
PICTURE
iOS Device
AES-256 (KDF-key) + HMAC (HKDF-key)
M-key
HM-key
CPUB-key
SPUB-key
PIN-cert
AES-256 (M-key) + HMAC (HM-key)
CPR-key
RSA-signature (CPR-key)
AES-256 (M-key) + HMAC (HM-key)
PICTURE
IV

Figure 6.5: Architecture snapshot - technologies

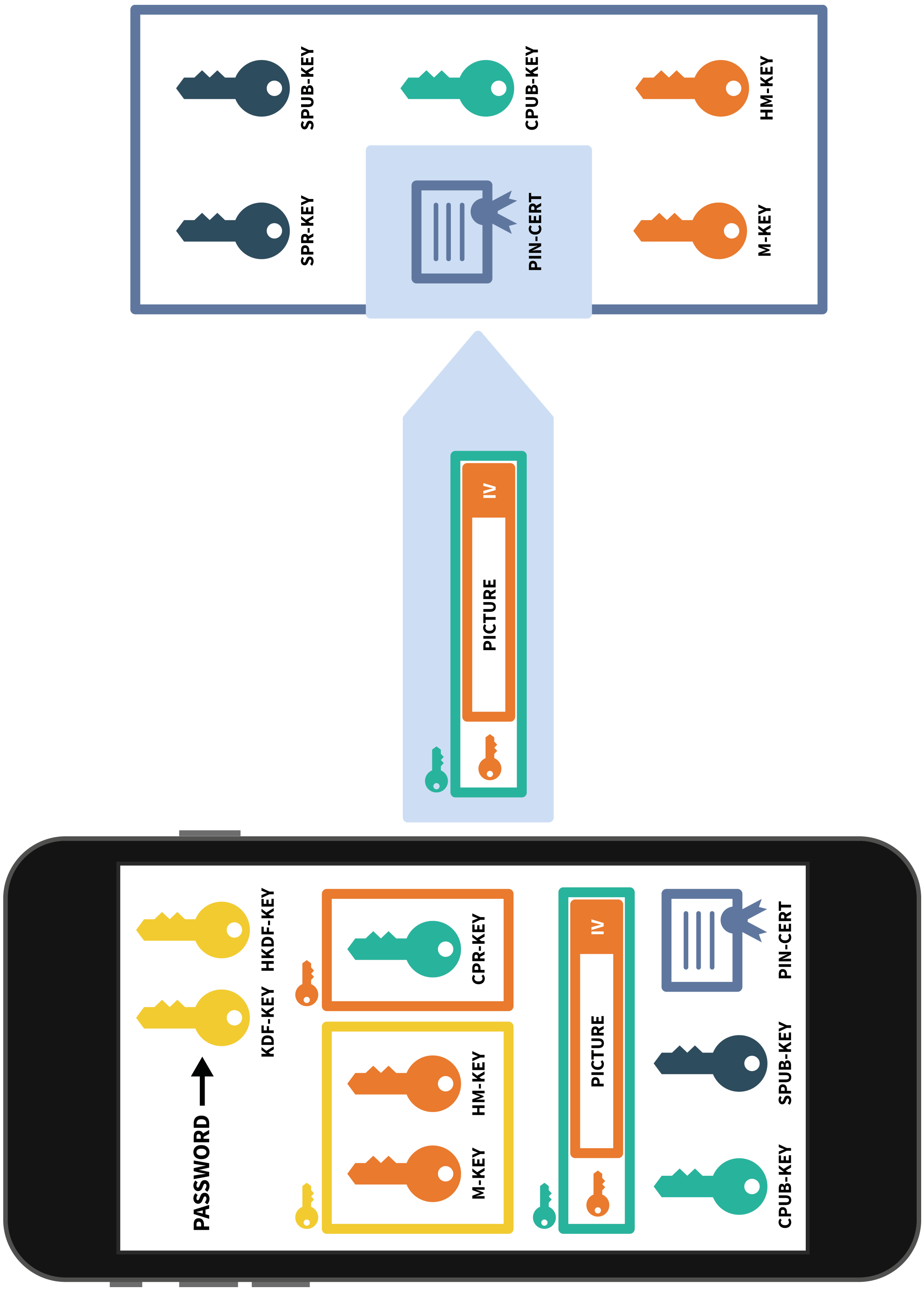


Figure 6.6: Architecture snapshot - graphic

6.3 Security Control Mapping

The following control mechanisms prevent, challenge or mitigate the possible attack methods identified in section 5.3.5 on page 38.

The attacks ***C1 to C5***, categorized as ***"carrier based methods"*** focus on targeting the connection between the client and the server. The main goal to prevent these kind of attacks is to establish a secure connection as described in section 6.2 on page 47, and to make sure that every data exchanged between both peers is sent over this channel.

The problem of ***SSL proxying (C4)*** is tackled with the SSL pinning method using the *PIN-cert* as mentioned in the connection definition on page 47.

The ***"endpoint based methods" (E1 - E3)*** represent a breach of confidentiality by allowing the perpetrator to access sensitive data.

The first step to avoid making sensitive data accessible is to reason whether a certain data set has to be stored in the device storage or can also be temporarily held in memory.

Encryption is mandatory, if a sensitive data record has to be stored to the device memory. The *M-key* and the *HM-key* are the master keys responsible for the encryption and generation of message authentication codes for the stored data. These two master keys (*M-key, HM-key*) are only held in memory after the user supplied a valid *app password*. After closing or locking the app the decrypted keys are wiped from the temporary memory and are only present in an encrypted version in the device storage. Since wiping encryption keys from memory is crucial, approaches for secure file and memory wiping are described in "Hacking and Securing iOS Applications" by Zdziarski (see [76]).

The third attack category ***"OS and application level methods"*** can not be tackled with a general security control approach. Each attack method has to be analysed separately.

To prevent ***O1. unintended data leakage*** it is recommended to stick to best practice programming guidelines and to check default values which may extend the applications attack vector or unintentionally expose sensitive data. The control mechanisms for ***"endpoint based methods"*** also help to mitigate the possible damage by unintended data leakage.

The attack method ***O2. client side injection*** can be split in different categories: SQL Injection, JavaScript Injection, XML Injection, Format String Injection, etc.

A method of resolution for injection in general is input validation. A secure parser has to be used all the time. XML is an exception and has to be handled differently to ensure protection against common XML attacks. Strong typing is also mandatory to guarantee that the input parameter has a certain value and type. Especially for POSTing or PUTing new data a type should never be assumed and it should also be checked that the Content-Type header and the content are of the same type. Also responses have to be validated and checked whether the Accept header matches the response content [64].

It is important that the data confirms to a expected length, range, type and format. An example for this kind of vulnerability is the *Format String Vulnerability* (see [73]) which could be exploited with a racoon configuration file and was fixed by a software update from Apple on 07.03.2012 (*APPLE-SA-2012-03-07-2 iOS 5.1 Software Update*).

The attack method ***O3. Session(-Cookie) Hijacking*** utilizes the users cookies to spoof the attackers identity and therefore elevates his privileges. By design, cookies are not shared among applications in iOS (see [9]). Every application stores its persistent cookies in the default directory *[AppDirectory]/Library/Cookies/Cookies.binarycookies* (see [76]). Since the binarycookies-format is already decoded, (see [61]) an attacker can access and decode the cookie file on a jailbroken phone.

Therefore cookies should not be persisted and used for one session only. All cookies, including the cookie storage, should be removed when locking or closing the application. If persistent cookies are mandatory, the server should handle the session timeout and the validity of the cookie. All cookies have to be sent via an encrypted channel only (secure flag).

To prevent ***O4. Reverse Engineering*** the following methods should be applied. Tamper response mechanisms should be implemented, which wipe sensitive data like encryption keys when the attacker tries to tamper with the application. Other possible tamper responses are enabling logging, reporting home or disabling network access. It is also possible for an attacker to debug the application. As a security mechanism the application can monitor the flag signifying that the application process is being traced. Also a flag can be specified which blocks debugging in general. To complicate disassembly of the application, various methods can be applied, like optimization flags, stripping the binary or unrolling loops. All these methods are described in "Hacking and Securing iOS Applications" by Zdziarski in the section *Securing the Runtime* (see [76]).

A very crucial attack method is the ***O5. Jailbreaking*** which allows the perpetrator to elevate his privilege and to access and manipulate the whole device memory. There are various methods to detect if an iOS device is jailbroken which include a sandbox integrity check, file system tests, checking the size of /etc/fstab, the evidence of certain symbolic links and the page execution check (see [76]). If a jailbreak test is positive the device has to report it to the server where further actions are taken by the administrator. In order to reduce the risk of attackers being able to jailbreak the device, it has to be provided with the latest iOS security updates as soon as possible.

Another solution to prevent and detect the jailbreaking of a device, is the implementation of a Mobile Device Management (MDM) system. It can detect altered mobile devices and unapproved OS configurations and can respond with predefined actions. The MDM administrator is able to define security policies and monitor and control installed applications as well as the transfer of information between mobile devices [25]. Therefore MDM systems are an useful asset to prevent and detect a variety of threats.

The last attack method of the ***"OS and application level methods"*** is ***O6. stealing the keyboard cache***. *"The keyboard cache contains a cache of data typed into the keyboard from anywhere in any application, unless specifically disabled by turning off a text field's autocorrect feature or making the field a secure password field"* [76].

It is also possible to disable the copy/paste functionality for certain text fields to reduce the risk of unintentional data leakage since data is cached in plain text when it is copied to the clipboard (see [76]). In consideration of that the keyboard cache can not be deleted by the application, it should be considered to write a customized keyboard class for the application.

With the release of iOS 8, Apple introduced the possibility to create third party keyboard applications. Custom keyboard applications also represent a possible threat although full access is disabled by default. Some keyboards may require a full access permission to enable special features. This enables keyboards to send keystrokes to external servers, which can lead to an unwanted data leakage. Even if the developers intentions are not maleficent, the sensitive data is also vulnerable to attacks from other sources [56].

Secure text entry fields, which are used for password inputs, are an exception. Custom keyboards are not eligible to type into password fields. Therefore, when a user taps in a secure text field, the OS temporarily switches to the systems native keyboard and automatically switches back when the user taps into a non secure input field [7].

7 Implemented iOS Application

The "Implemented iOS Application" chapter details the use cases, derived from the reference mobile imaging application that are implemented in the application prototype. The extracted functional requirements and their transformation into user interface views are outlined. Further, the setup, including the development environment and the used frameworks, is specified in this chapter. The designed and implemented classes are illustrated by a domain model and separately described. For a thorough insight into the implementation of the designed cryptographic architecture, the implementation of the use cases, using the designed classes, is elaborated.

7.1 Use Cases

The system level use cases, shown in figure 7.1 on page 52, are derived from the medical skin imaging application's requirements and implemented in the application prototype.

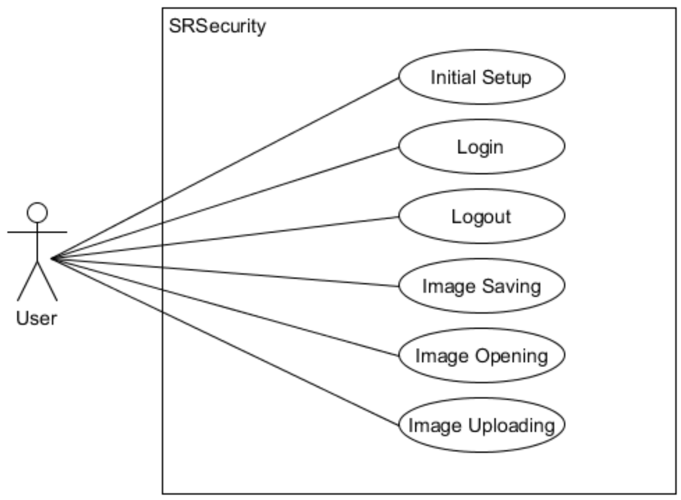


Figure 7.1: System use cases

Each use case is described separately, based on the following schema, adapted from Grechenig et al.'s book "Softwaretechnik: Mit Fallbeispielen aus realen Entwicklungsprojekten" (see [36]).

Use case schema description:

1. **"Identification summary"**
    - ***Title:*** The use case title
    - ***Scope:*** The use case package
    - ***Level:*** User-, system-, or enterprise-level
    - ***Actors:*** The involved actors

- ***Summary:*** A short summary of the use case

2. **“Scenarios”**

   - ***Pre-Conditions:*** A pre-condition for the successful execution of the use case
   - ***Main-Scenario:*** A precise description of the use case
   - ***Error-Scenario:*** Consequences and system state in case of an error-scenario
   - ***Alternative-Scenario:*** Alternative scenario, if possible
   - ***Post-Conditions:*** The system state after a successful execution of the use case

3. **“Nonfunctional Constraints”**

   - Remarks and nonfunctional constraints

For demonstration purposes the use case “Initial Setup” is subsequently described. The complete list of use cases and descriptions can be found in the Appendix (see A.1 on page 106).

Initial Setup:

1. **“Identification summary”**

   - ***Title:*** Initial Setup
   - ***Scope:*** Start Screen
   - ***Level:*** System Goal
   - ***Actors:*** User
   - ***Summary:*** The user has entered a valid user name and password. Symmetric and asymmetric keys are created.

2. **“Scenarios”**

   - ***Pre-Conditions:*** A valid user name and password are supplied.
   - ***Main-Scenario:*** The user enters a user name and password. If they fulfill the predefined conditions an asymmetric key pair is generated (*CPR-key* and *CPUB-ke*y). Then a symmetric AES key (*M-key*) and a symmetric Keyed-Hash Message Authentication Code (HMAC) key (*HM-key*) are created. The *CPR-key* is encrypted and maced with the *M-key*, *HM-ke*y and a random *Initialization Vector (IV)* and stored in the keychain. The *M-key* and *HM-key* are concatenated to create the *MHM-key* which is also encrypted by the keys derived from the user’s password and stored in the keychain. Finally the *CPUB-key* is also stored in the keychain.
   - ***Error-Scenario:*** If the user name and password do not match the criteria, the initial setup is not executed and an error message is shown to the user.
   - ***Alternative-Scenario:*** No alternative scenario.
   - ***Post-Conditions:*** The user is logged in and the user’s *M-key*, *HM-key*, *CPR-key* and *CPUB-key* are available to the user.

3. **“Nonfunctional Constraints”**

   - Depending on the hardware, the key generation takes a variable amount of time, therefore no timeout is set for the key generation. It should not take longer than 2 minutes to finish the setup.

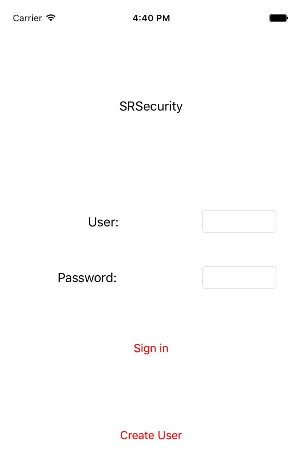
7.2 Views

Based on the use cases from section 7.1 on page 52, views are implemented in the prototype application. Each view is defined by the functional requirements extracted from the biomedical skin imaging reference application.

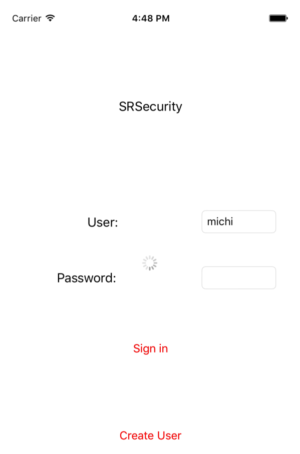
The following list and screen shots (see figure 7.2 on page 54 and figure 7.3 on page 55) give an exemplary impression of a view's requirements and the corresponding user interface.

Welcome View

- Show input fields for user name and password (see figure 7.2a on page 54)
- Login User with user name and password supplied
- Create User with user name and password supplied
- An activity indicator is shown during the "initial setup" (see figure 7.2b on page 54)
- An error message is displayed when the password is not valid (see figure 7.3a on page 55)
- An error message is displayed when the user name is already taken (see figure 7.3c on page 55)
- An error message is displayed when the user name and password combination do not match (see figure 7.3b on page 55)
- An error message is displayed when the user name does not exist (see figure 7.3b on page 55)
- An error message is displayed when "initial setup" fails (see figure 7.3b on page 55)

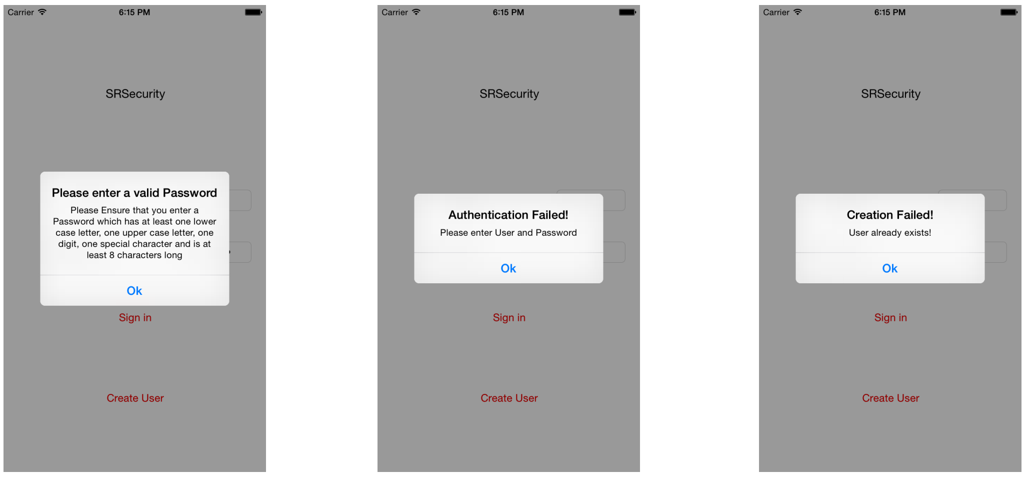


(a) Default



(b) Generating keys

Figure 7.2: Welcome View - General



Please enter a valid Password
Please Ensure that you enter a Password which has at least one lower case letter, one upper case letter, one digit, one special character and is at least 8 characters long

Authentication Failed!
Please enter User and Password

Creation Failed!
User already exists!

(a) Password not valid **(b)** General error **(c)** User already exists

Figure 7.3: Welcome View - Error

A full list of the views, including the individual functional features and screen shots for each required scenario can be found in the Appendix section A.2 on page 111.

7.3 Setup

The prototype application, SRSecurity, is written in objective-C and partly in C for iOS version 8.4. The application is developed on a Macbook Pro (Retina, 13-inch, Mid 2014) with OS X 10.10.5 Yosemite and Xcode 6.2 as IDE. The test device is a jailbroken iPhone 4S 32GB (Model MD242D/A) with iOS 7.1.1.

RNCryptor is the only external library used in the application. RNCryptor (see [62]) is developed by Robert Napier and implements features which are available in the standard SDK only (CommonCrypto, see [6]). RNCryptor is a CCCryptor (AES encryption) wrapper for iOS and provides feature to encrypt and decrypt NSData. It includes AES-256 encryption with CBC mode, Password stretching with PBKDF2, Password salting, Random IV and Encrypt-then-hash HMAC. Another external code fragment used, is the file wiping mechanism by Jonathan Zdziarski described in his book “Hacking and Securing iOS Applications” (see [76]) chapter 11, secure file wiping. This C code enables to open a file and overwrite it byte per byte. Specifically this method recommended by the US Department of Defense (DoD) (see page 274 of [76]) to destroy classified data, is implemented in the application prototype. The sample code from Apple’s PhotoPicker is used to take pictures from the device library or the device camera (see [13]).

7.4 Implementation

7.4.1 Domainmodel

The following figure (see figure 7.4 on page 56) represents the domain model of the prototype application and gives an overview of the implemented classes.

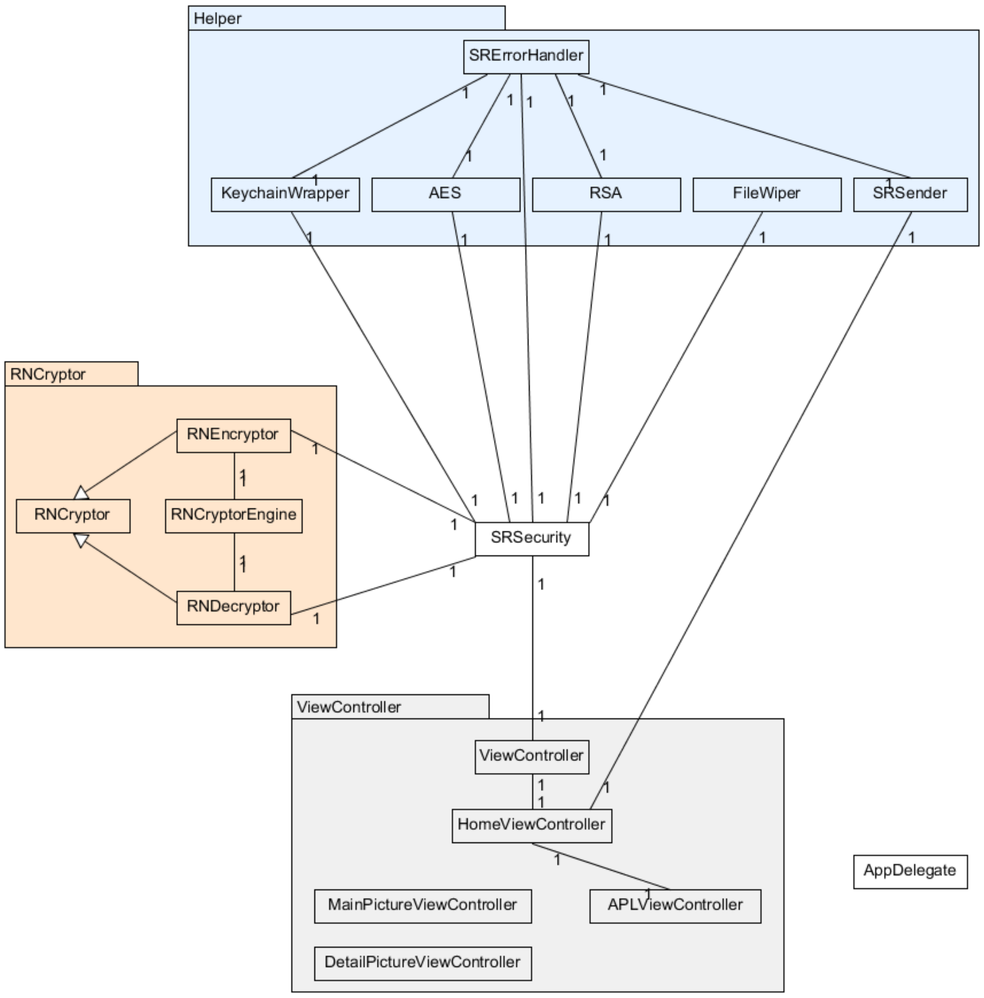


Figure 7.4: Domainmodel SRSecurity

7.4.2 Classes Definition

Main - Package

AppDelegate:

This is the delegate class for the main UIApplication. XCode automatically generates the AppDelegate class for every new project. The AppDelegate provides features like responding to the change of the application state, responding to notifications from outside the app or assists in the preservation and restoration process of the application. For more detailed information see [15].

ViewController - Package

ViewController:

The ViewController for the view “Welcome View” is described in Appendix A.2 on page 111.

HomeViewController:

The ViewController for the view “Home View” is described in Appendix A.2 on page 112.

MainPictureViewController:

The ViewController for the view “Show Picture View” is described in Appendix A.2 on page 116.

DetailPictureViewController:

The ViewController for the single picture of “Show Picture View” is described in Appendix A.2 on page 116.

APLViewController:

The ViewController for the view “Take Picture View” is described in Appendix A.2 on page 115.

RNCryptor - Package

This subsection gives an overview of the RNCryptor library implemented by Robert Napier (see [62]).

RNCryptor:

It is an abstract class for RNEncryptor and RNDecryptor and provides the method to generate a key given a password and salt using PBKDF with specified settings. The settings include the resulting key size, the salt size, the PBKDF algorithm, the Pseudorandom Function (PRF) algorithm and the number of rounds for the PBKDF. The class also provides a synchronous result for the encryption- or decryption-operation. For more details see [62].

RNCryptorEngine:

RNCryptorEngine is initialized with a CommonCrypto operation, settings, a key and an IV. This class executes the encryption or decryption operation utilizing the CommonCrypto library. It includes the methods CCCryptorCreate, CCCryptorUpdate and CCCryptorFinal (see [6]). For more details see [62].

RNEncryptor:

This class is a derivation of the RNCryptor class and allows the user to encrypt data with a provided password, using a PBKDF2 function in the background. The user has to provide settings to define the used AES-algorithm, the AES-blocksize, the IV size, padding options, the HMAC-algorithm and the HMAC-length. The settings also include the specification for the AES and HMAC key, like the key size, the salt size, the PBKDF algorithm, the PRF and the rounds for the PBKDF. RNEncryptor which also enables the user to decide whether he wants to use a password, with a

PBKDF, or provide an AES-key and a HMAC-key. For more details see [62].

RNDecryptor:

This class has basically the same features like the RNEncryptor class, but it allows the user to decrypt data. This can be done by providing a password or an AES-key and a HMAC-key. The main difference is that the settings are optional, since the RNEncryptor class stores this information in the header of the encrypted data. Although the user can also specify settings manually, which commands the RNDecryptor class to ignore the settings stored in the header and use the ones provided by the user. For more details see [62].

Helper - Package

RSA:

The RSA class holds the private key (*CPR-key*) and the public key (*CPUB-key*) of the user currently logged in. It allows to generate a key pair, which is invoked during the initial setup (see Appendix A.1 on page 106). The Rivest-Shamir-Adleman cryptosystem (RSA) class also features methods to encrypt or sign data with a public key and to decrypt or verify data with a private key.

AES:

This class stores the AES-key (*M-key*), the HMAC-key (*HM-key*) and the concatenation of both (*MHM-key*) after the login. It is used to generate an AES- and HMAC-key pair and allows to concatenate them into the *MHM-key* or split the *MHM-key* into the *M-key* and the *HM-key*.

KeychainWrapper:

The KeychainWrapper class is used to interact with the iOS keychain. This class allows the handling of keys of the type SecKeyRef or NSData. These two types of keys can be added, removed or retrieved from the keychain.

SRErrorHandler:

SRErrorHandler initializes an NSError object with a user-defined domain and error message. It is used in different classes throughout the project to create custom errors.

SRSender:

The method sendDatatoURL of SRSender is used to send the encrypted and signed pictures of a user to the remote server. It creates an NSURLSession with a HTTP POST method. Delegate methods are implemented to handle different events throughout the process. The delegate method didReceiveChallenge is implemented to perform a certificate pinning with the pre-packaged server certificate (*SPUB-key*).

FileWiper:

The FileWiper performs methods recommended by the US DoD (see page 274 of [76]) to destroy classified data. This class is used to wipe the users stored pictures. The original implementation in C by Jonathan Zdziarski can be found in the book “Hacking and Securing iOS Applications - Stealing Data, Hijacking Software, and How to Prevent It” on page 274 (see [76]).

SRSecurity - Package

SRSecurity:

SRSecurity is the class which implements the different use cases and functionalities for one logged in user. It holds the user name and (temporarily) the password for the user. A KeychainWrapper instance, an AES instance and a RSA instance are part of this class. It is also responsible for the memory cleanup when the user logs out of the application.

7.4.3 Use Cases

Initial Setup

Before processing the "initial setup" a SRSecurity object with the user name and the password is created, the password is validated and the keychain is checked whether the user already exists.

The "initial setup" creates a RSA instance and generates the RSA key pair (*CPR-key*, *CPUB-key*). After that, an AES instance is created and an AES-key (*M-key*) and a HMAC-key (*HM-key*) are produced.

Then a KeychainWrapper is instantiated. To get a NSData-type key a SecKeyRef-key has to be temporarily added to the keychain. The retrieved NSData of the *CPR-key* is encrypted with the RNEncryptor with settings for AES-256, the *M-key* as AES-key and the *HM-key* as HMAC-key. This encrypted *CPR-key* NSData (*mac-enc-CPR-key*) is added to the keychain.

During the next step the *M-key* and *HM-key* are concatenated to create the *MHM-key*. This *MHM-key* is encrypted with the RNEncrypter, with settings for AES-256 and the user's password. The *mac-enc-MHM-key* is also added to the iOS keychain.

The last step of the "initial setup", is saving the *CPUB-key* in the keychain. Finally, the user password is wiped from memory, by overwriting it with 0-values in memory and setting it to nil. For the definition of the "initial setup" use case see Appendix A.1 on page 106.

Login

Before processing the "login" use case, like before the "initial setup", a SRSecurity object with the user name and the password is created, the password is validated and the keychain is checked whether the user already exists.

At first, a KeychainWrapper instance is created and the *mac-enc-MHM-key* for the supplied user name is fetched from the keychain. Next, the RNDecryptor tries to decrypt the *mac-enc-MHM-key* with the provided password. If it fails, the user has provided a wrong user name and password combination and is represented with an error message.

If the decryption is successful, the *MHM-key* is split into the *M-key* and the *HM-key*. Then the *mac-enc-CPR-key* is retrieved from the keychain and decrypted with the *M-key* and the *HM-key*. To get a SecKeyRef key, the decrypted NSData *CPR-key* has to be temporarily added to the keychain.

At last, the SecKeyRef *CPUB-key* is restored from the iOS keychain and the user password is wiped from memory, by overwriting it with 0-values in memory and setting it to nil. For the definition of the "login" use case see Appendix A.1 on page 106.

Logout

When a user is logged in a SRSecurity object, including a RSA, AES and KeychainWrapper instance, is already available. These objects hold the user name, the *M-key*, the *HM-key*, the *CPR-key* and the *CPUB-key*, which are necessary for encryption and decryption procedures. Throughout the logout process the memory is cleaned from sensitive information like keys and captured images stored in memory.

At first, the SecKeyRef-type RSA key pair, *CPR-key* and *CPUB-key*, are released from memory (CFRelease) and set to nil. Then the AES keys, *M-key* and *HM-key*, are overwritten with 0-values in memory and set to nil. Although the capturedImages array should be empty after the images are not open and are stored encrypted to the device storage, all items are removed from the array and the array is set to nil.

Finally, the user name and password, which should be already wiped after the login, are wiped from memory by overwriting it with 0-values in memory and setting it to nil. Then the RSA, AES, KeychainWrapper and SRSecurity objects are set to nil. For the definition of the "logout" use case see Appendix A.1 on page 107.

Image Saving

The captured picture is encrypted using the RNEncryptor with AES-256 settings, the *M-key* as AES-key and the *HM-key* as HMAC-key. The encrypted and maced image is signed with the *CPR-key* and stored to the Documents directory of the application. For the definition of the "image saving" use case see Appendix A.1 on page 108.

Image Opening

The users signed, maced and encrypted images are fetched from the Documents directory. Then the images signature is validated with the *CPR-key*. If the validation fails, the image is skipped and the next image is processed. After a successful signature verification, the image is decrypted using the RNDecryptor, the user's *M-key* and the user's *HM-key*. The successfully decrypted image is temporarily added to the capturedImages array or skipped if the decryption fails.

The decrypted, in-memory stored images are then listed in the "Show Picture View". After closing this view, the capturedImages is cleared and the decrypted images are removed from memory. For the definition of the "image opening" use case see Appendix A.1 on page 108.

Image Uploading

First, the user's signed, maced and encrypted images are read from the device storage. The next step is the creation of a SRSender object, which implements delegate methods that handle the certificate pinning process and different events throughout the uploading process. The sendDatatoURL creates a secured connection and uses HTTP POST to send the images to the remote server. For the definition of the "image uploading" use case see Appendix A.1 on page 109.

8 Analysis & Results

Throughout the following chapter the main aspects of the prototype application are analyzed. The main factors can be divided into three categories: transmission, storage and runtime. Each section is separately analyzed and starts with a general description and the determination of possible threats. Furthermore corresponding recommendations are given to eliminate or mitigate these threats. After a thorough analysis of the applications feature, using different approaches and tools, the results for each section are highlighted.

8.1 Transmission Analysis

8.1.1 General

As described in chapter 6.2 on page 47, the app uses HTTP as application protocol for the image uploading process and secures this connection via TLS. The following sections describe the threats and exploits faced by former and current TLS/SSL versions and presents recommendations for protocol versions and cipher suits, which mitigate these threats. Furthermore the Hypertext Transfer Protocol over TLS (HTTPS) connection, implemented in the application prototype, is analyzed.

8.1.2 Threats

The TLS protocol is extensively used to secure the communication between applications over a network. In the last years various serious attacks have occurred on the standard's modes of operation and ciphers. The following section represents an overview of the attacks as described in IETF's "RFC 7457 - Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)" (see [46]).

SSL Stripping

The attacker pursues the removal of the TLS layer. This attack can only be attempted if the client initially connects to a server using an insecure connection [46].

Mitigation: The usage of HTTP Strict Transport Security (HSTS) mitigates this kind of attacks [46].

STARTTLS Command Injection Attack (CVE-2011-0411)

Numerous protocols use the STARTSSL tag to open a secure connection. In consequence of an improper implementation, the input buffer is retained. Therefore, data received before the creation of the secure TLS connection, was executed after the negotiation. The usage of STARTSSL can also lead to a downgrade attack, which just removes the tag and prevents the creation of a TLS connection [46].

Mitigation: The usage of HSTS mitigates this kind of attacks [46].

BEAST (CVE-2011-3389)

The BEAST attack takes advantage of an issue with the TLS 1.0 Cipher Block Chaining (CBC) implementation. It allows the decryption of specific crypto data, like HTTP cookies [46].

Mitigation: The usage of a newer version than TLS 1.0 mitigates this kind of attacks [46].

Padding Oracle Attacks

Examples for this kind of attacks are the Lucky Thirteen attack (CVE-2013-0169) or the POODLE attack (CVE-2014-3566). It allows the decryption of arbitrary crypto data. This attack is an after effect of the MAC-then-encrypt design [46].

Mitigation: The usage of authenticated encryption like AES-GCM or encrypt-then-MAC mitigates this kind of attacks [46].

Attacks on RC4

It is known that RC4 nowadays does not provide a certain level of security [46].

Mitigation: The usage of ciphers other than RC4 avoids this kind of attacks [46].

Compression Attacks: CRIME, TIME, and BREACH

The CRIME, TIME and BREACH attacks allow the decryption of data like encrypted cookies. The first two use a weakness in the TLS-level compression in comparison to the latter which exploits the compression on HTTP-level [46].

Mitigation: The CRIME and TIME attack can be mitigated by disabling TLS-compression. The BREACH attack allows no mitigation on TLS-level, but on application level [46].

Certificate and RSA-Related Attacks

There are different attacks concerning RSA certificates, since those are the most commonly deployed ones [46].

Mitigation: Depending on the attack, there are different mitigation mechanisms, for example the Bleichenbacher attack is mitigated in TLS 1.0, while the Klima attack is only mitigated in TLS 1.1 [46].

Recently, the IT security research team of the Austrian company RISE GmbH disclosed a “Key Compromise Impersonation” attack. The TLS protocol holds various key agreement and authentication methods which are vulnerable to this attack. Even non-ephemeral Diffie-Hellman key exchange mechanisms with fixed Diffie-Hellman client authentication are vulnerable. The attack allows the impersonation of a trusted server and furthermore the execution of a MITM attack [38].

Mitigation: To mitigate this attack, it is proposed to disable non-ephemeral Diffie-Hellman or at least fixed Diffie-Hellman client authentication. It is also recommended to evaluate whether X509 Key Usage Extensions should be considered [38].

Theft of RSA Private Keys

In most cases, a stolen server private key allows the decryption of all transmitted cipher data if a non Diffie-Hellman cipher suite is used [46].

Mitigation: Cipher suits which enable forward secrecy should be used. If this is not possible, the server private key should be better protected [46].

Renegotiation (CVE-2009-3555)

All current TLS versions suffer from an attack on the renegotiation mechanism [46].

Mitigation: The attack can be mitigated with the extension described in RFC 5746 [46].

Triple Handshake (CVE-2014-1295)

This attack allows the attacker to create two TLS connections which share key material. This enables for example MITM attacks [46].

Mitigation: The client should make sure that all certificates of a connection are valid for the server [46].

Logjam

The, in May 2015 discovered, logjam attack allows a MITM attack and furthermore a downgrade attack to create a TLS connection with 512-bit export-grade cryptography, which can be exploited to read and decrypt the sent data. About 8,4% of the top one million HTTPS servers are vulnerable to this Diffie-Hellman key exchange attack. The risk is present for DH groups until 1024-bit which are in range of decryption for state-level attacks [1].

Mitigation: The export cipher suits, which are no longer supported by modern browsers, need to be disabled. The use of (ephemeral) Elliptic-Curve Diffie-Hellman (ECDHE) is recommended which avoids all known feasible crypto attacks. Elliptic-Curve Diffie-Hellman is also shorter and faster than standard Diffie-Hellman. Also, strong Diffie-Hellman groups need to be used. If 1024-bit groups are necessary, it is suggested to avoid fixed groups. 2048-bit groups provide a sufficient level of security [1].

8.1.3 Recommendations

Protocol

Considering the steps recommended by the Internet Engineering Task Force (IETF) (see [47]), the IT security research team of RISE GmbH (see [38]) and Adrian et al. (see [1]), the following steps need to be implemented to mitigate the attacks on a TLS protocol level.

- Stop using SSLv2 and SSLv3.
- TLS 1.0 is not recommended, since it does not hold modern strong ciphers.
- TLS 1.1 is an improvement to TLS 1.0, but does not implement certain strong ciphers.
- TLS 1.2 holds strong ciphers, including the ones recommended by the IETF.

- If clients need to fallback to a lower version, they must not fall back lower than SSLv3.
- HSTS and TLS only clients should be preferred to avoid attacks like SSL stripping.
- TLS-compression should be disabled to mitigate compression attacks on TLS-level.
- To avoid TLS handshake attacks, ciphers which provide forward secrecy and the refusal of a certificate change during renegotiation are recommended.
- The TLS Server Name Indication (SNI) extension must be supported.
- Evaluate whether X509 Key Usage Extensions should be considered.

Cipher Suites

The following TLS cipher suits are recommended by the IETF (see [47]), the IT security research team of RISE GmbH (see [38]), Adrian et al. (see [1]) and Breyha et al. of bettercrypto.org (see [22]).

- NULL encryption and RC4 ciphers must not be used.
- Cipher suits with less than 112 bits must not, and cipher suits with less than 128 bits should not be used.
- Instead of static RSA cipher suits, cipher suits providing forward secrecy must be used.
- Disable non-ephemeral ECDHE, or at least disable fixed (EC)DH authentication.
- Strong Diffie-Hellman groups with at least 2048-bit should be used.
- ECDHE is shorter and faster than standard Diffie-Hellman.
- Authenticated Encryption with Associated Data (AEAD) algorithms like AES-GCM are recommended.

Conclusion

These recommendations conclude the following cipher string:

`TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`

TLS v1.2 is the used protocol, since it (currently) is the only protocol version providing these cipher suits. The key exchange protocol is Elliptic-Curve Diffie-Hellman (ECDHE) which provides forward secrecy. With forward secrecy it is not feasible for an attacker to obtain the servers private key, since a different key is generated for every communication session. The server and client hold Diffie-Hellman parameters to calculate the session key and only use the long-termin private key to encrypt the session keys [47]. The used cipher for this cipher string is AES-256. The GCM parameter indicates that a AEAD algorithm is used which takes care of encryption as well as authentication. If GCM is not available, a SHA-384 Message Authentication Code (MAC) is calculated for authentication purposes.

8.1.4 Analysis

As described in chapter 7.4.2 on page 58, the image transmission functionality is implemented in the SRSender class. It implements delegate methods for the NSURLSession class to send the signed and encrypted pictures, via HTTP POST message, to the remote server.

The testsetup (see figure 8.1 on page 65) consists of a jailbroken iPhone[1] 4S 32GB (Model MD242D/A) with iOS 7.1.1[2] as client and a Raspberry Pi[3] Model B Revision 2.0 with Debian 8.0[4] as server. The server runs a plain webserver with TLS and poses as a connection endpoint only. No further application logic is implemented on the server side. The server runs an Apache webserver[5] (version 2.4.10) with openssl[6] (version 1.0.2d). Tcpdump[7] (version 4.6.2) is installed on the server side to capture incoming packages.

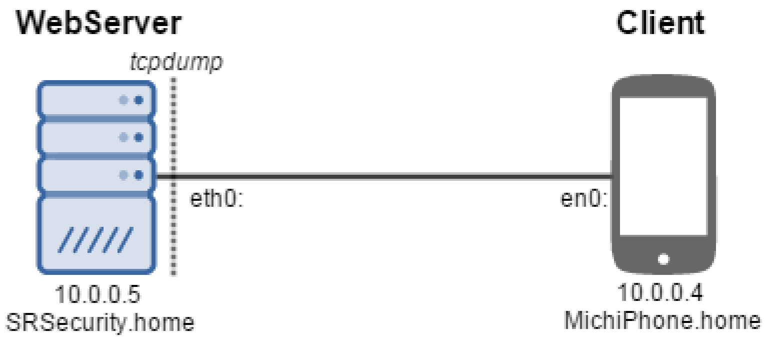


Figure 8.1: Testsetup

Following the recommendations elucidated in chapter 8.1.3 on page 64 and the configuration guide of bettercrypto.org for Apache (see [22] chapter 2.1.1 on page 11) these custom Apache SSL configuration parameters can be derived (see listing 8.1 on page 65):

```
#TLS Protocol Version 1.2
SSLProtocol TLSv1.2

#The Server's cipher preference will be used, instead of the
↪ client's
SSLHonorCipherOrder On

#TLS compression is disabled
SSLCompression Off

#HSTS is enabled
Header always set Strict-Transport-Security "max-age=15768000"

#Custom Diffie-Hellmanparameters
#Created with command: openssl dhparam -out dhparams.pem 2048
SSLOpenSSLConfCmd DHParameters "/etc/apache2/dhparams.pem"
```

[1] http://www.apple.com/iphone
[2] http://www.apple.com/ios
[3] https://www.raspberrypi.org
[4] https://www.debian.org
[5] http://httpd.apache.org
[6] https://www.openssl.org
[7] http://www.tcpdump.org

```

#Cipher-Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
SSLCipherSuite 'EECDH+aRSA+AESGCM'
```

Listing 8.1: Apache 2.4.10 SSL recommended configuration

The allowed cipher suits for the configured cipher string are (see table 8.1 on page 66):

OpenSSL Name	Version	KeyEx	Auth	Cipher	MAC
ECDHE-RSA-AES256-GCM-SHA384	TLSv1.2	ECDH	RSA	AESGCM(256)	AEAD
ECDHE-RSA-AES128-GCM-SHA256	TLSv1.2	ECDH	RSA	AESGCM(128)	AEAD

Table 8.1: EECDH+aRSA+AESGCM cipher suits

The following tcpdump command is executed on the server (see listing 8.2 on page 66) and a signed and encrypted picture is sent from the client to the server.

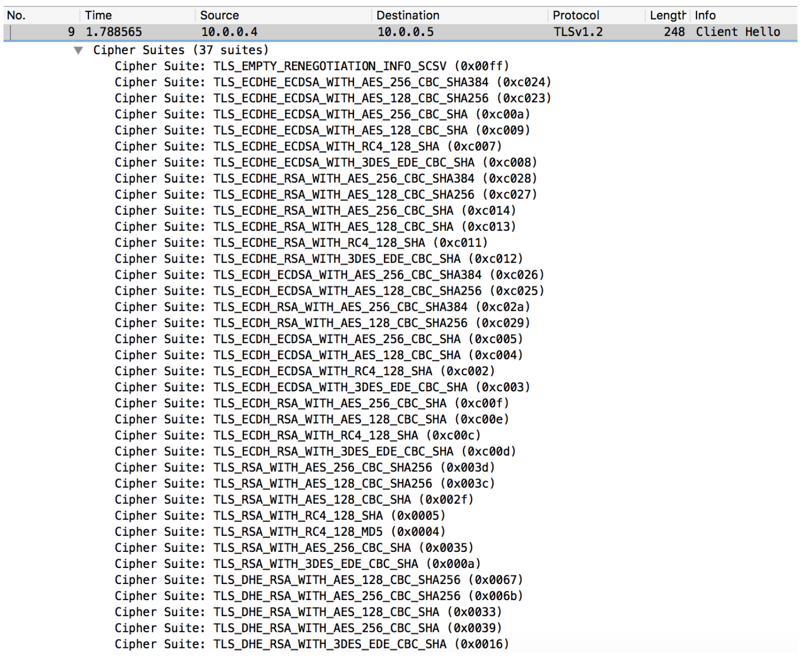
```
$ tcpdump -i eth0 -vvv -w caputre.cap
```

Listing 8.2: tcpdump command executed on server

Enabling this strict cipher set only, it produces an error ("*NSURLConnection/CFURLConnection HTTP load failed (kCFStreamErrorDomainSSL, -9824)*") on the client side. The iOS application developed in iOS 8.4 does not support the recommended strict cipher set, therefore another strict cipher set providing a nearly similar level of security has to be found.

After opening the file containing the, via tcpdump, captured packages with Wireshark[8] (version 1.99.9), the TLS Client Hello Message can be extracted. According to RFC 5246 chapter 7.4.1.2 (see [45]) the Client Hello message contains a cipher suit list, which holds cipher suits supported by the client in order of the client's preference. The extracted message is displayed in figure 8.2 on page 67.

[8] https://www.wireshark.org



```
No.   Time        Source      Destination   Protocol   Length  Info
    9 1.788565    10.0.0.4    10.0.0.5      TLSv1.2       248  Client Hello
  ▼ Cipher Suites (37 suites)
      Cipher Suite: TLS_EMPTY_RENEGOTIATION_INFO_SCSV (0x00ff)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA (0xc007)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc008)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
      Cipher Suite: TLS_ECDHE_RSA_WITH_RC4_128_SHA (0xc011)
      Cipher Suite: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (0xc012)
      Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 (0xc026)
      Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 (0xc025)
      Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 (0xc02a)
      Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 (0xc029)
      Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xc005)
      Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xc004)
      Cipher Suite: TLS_ECDH_ECDSA_WITH_RC4_128_SHA (0xc002)
      Cipher Suite: TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc003)
      Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xc00f)
      Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (0xc00e)
      Cipher Suite: TLS_ECDH_RSA_WITH_RC4_128_SHA (0xc00c)
      Cipher Suite: TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA (0xc00d)
      Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003d)
      Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c)
      Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
      Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
      Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
      Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
      Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
      Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (0x0067)
      Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (0x006b)
      Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
      Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
      Cipher Suite: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x0016)
```

Figure 8.2: TLSv1.2 Client Hello: Cipher Suits List

Taking the suggestions from chapter 8.1.3 on page 64 into account, the strongest recommended cipher, which can be derived from the list in figure 8.2 on page 67, is:

`TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384`

Changing the web server's configured cipher from "*EECDH+aRSA+AESGCM*" to "*EECDH+aRSA+AES256!SHA1*" allows the client to establish a secure connection and changes the allowed ciphers to the following (see table 8.2 on page 67):

OpenSSL Name	Version	KeyEx	Auth	Cipher	MAC
ECDHE-RSA-AES256-GCM-SHA384	TLSv1.2	ECDH	RSA	AESGCM(256)	AEAD
ECDHE-RSA-AES256-SHA384	TLSv1.2	ECDH	RSA	AES(256)	SHA384

Table 8.2: EECDH+aRSA+AES256!SHA1 cipher suits

A successful encrypted connection can be found in the servers log file and the used cipher suite, "*TLSv1.2 ECDHE-RSA-AES256-SHA384*", is displayed.

The difference between the recommended and the working cipher suit is the Message Authentication Code (MAC) method. Compared to the recommended cipher, which only allows AES with GCM mode and therefore an Authenticated Encryption with Associated Data (AEAD) Message Authentication Code (MAC) method, the working cipher suit uses SHA-384 to create the Message Authentication Code. Although Apple itself uses AES with GCM in iOS, as described in [10], tests show that neither NURLConnection nor NSURLSession support these cipher suits in iOS 8.4.

The listing 8.3 on page 67 represents an updated working configuration for Apache 2.4.10.

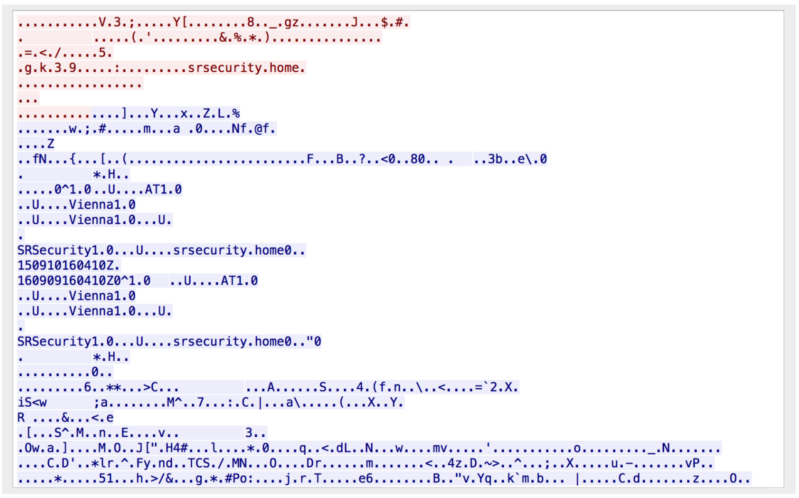
```
#TLS Protocol Version 1.2
SSLProtocol TLSv1.2

#The Servers cipher preference will be used, instead of the client's
```

```
SSLHonorCipherOrder On

#TLS compression is disabled
SSLCompression Off

#HSTS is enabled
Header always set Strict-Transport-Security "max-age=15768000"

#Custom Diffie-Hellmanparameters
#Created with command: openssl dhparam -out dhparams.pem 2048
SSLOpenSSLConfCmd DHParameters "/etc/apache2/dhparams.pem"

#Cipher-Suite: TLS_ECDHE_RSA_WITH_AES_256_SHA384
SSLCipherSuite 'EECDH+aRSA+AES256!SHA1'
```

Listing 8.3: Apache 2.4.10 SSL updated recommended configuration

While writing this thesis, iOS 9 was released. According to the document "iOS Security for iOS 9.0 or later" of September 2015 (see [12]), "App Transport Security" was introduced, which provides default connection requirements and supports AES with GCM mode. Therefore it might be reasonably assumed that the first recommended configuration (see 8.1 on page 65) works with iOS 9. Considering the defined testsetup, these assumptions could neither be tested, nor confirmed. An overview of the significant changes and improvements with iOS 9 are provided in chapter 9 on page 94.

Due to the nature of forward secrecy (see 8.1.3 on page 64 for a brief description), enabled by the used cipher, the TCP stream can not be decrypted, even with the server's private key. The encrypted TCP stream is shown in figure 8.3 on page 68.



```
...........V.3.;.....Y[........8.._.gz.......J...$.#.
.          .....(.'.........&.%.*.)...............
.=.<./.....5.
.g.k.3.9.....:.........srsecurity.home.
.................
...
..............]...Y...x..Z.L.%
.......w.;.#.....m...a .0....Nf.@f.
....Z
..fN...{...[..(........................F...B..?..<0..80.. .  ..3b..e\.0
.          *.H..
.....0^1.0..U....AT1.0
..U....Vienna1.0
..U....Vienna1.0...U.
.
SRSecurity1.0...U....srsecurity.home0..
150910160410Z.
160909160410Z0^1.0  ..U....AT1.0
..U....Vienna1.0
..U....Vienna1.0...U.
.
SRSecurity1.0...U....srsecurity.home0.."0
.          *.H..
..........0..
.........6..**...>C...        ...A......S....4.(f.n..\..<....=`2.X.
iS<w      ;a........M^..7...:.C.|...a\.....(...X..Y.
R ....&...<.e
.[...S^.M..n..E....v..        3..
.Ow.a.]....M.O..J[".H4#...l....*.0....q..<.dL..N...w....mv.....'...........o.........._.N.......
....C.D'..*lr.^.Fy.nd..TCS./.MN...O....Dr......m.......<..4z.D.~>..^...;..X.....u.-.......vP..
.....*.....51...h.>/&...g.*.#Po:....j.r.T.....e6........B.."v.Yq..k`m.b... |.....C.d.......z....O..
```

Figure 8.3: Encrypted TCP stream

As next step, the testsetup is extended (see figure 8.4 on page 69) with a proxy (Burp Suite Free Edition v 1.6.25[9]), which poses as Man-In-The-Middle.

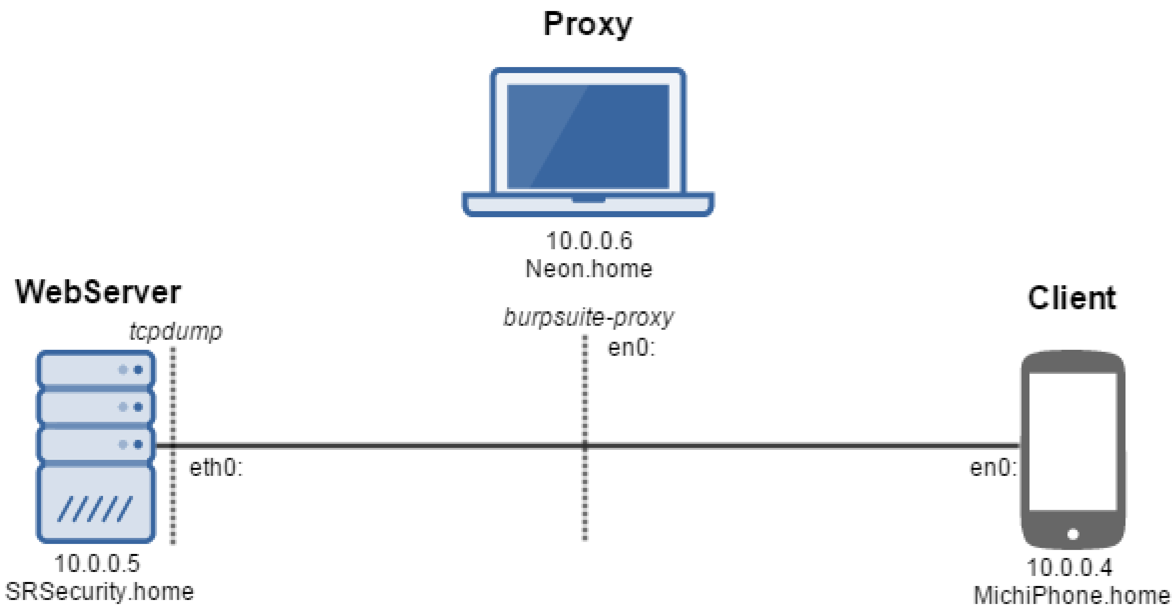
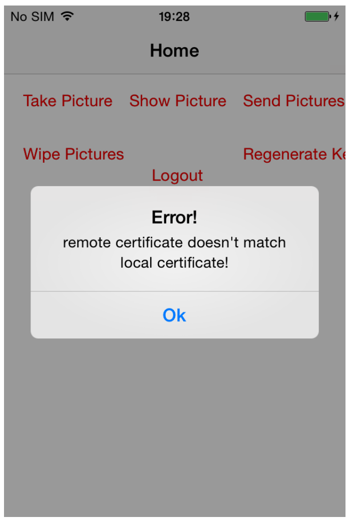


Figure 8.4: Testsetup with proxy

The client is configured to send every request via the proxy server and the proxy server is configured to intercept the traffic. The default behaviour of the Burp Suite Proxy is to generate a SSL certificate, signed by the Burp Suite CA, for every host requested over HTTPS. The prototype application, which implements SSL certificate pinning, recognizes this kind of MITM threat and drops the connection, due to the invalid certificate presented by the proxy. See figure 8.5 on page 69 for the error message.



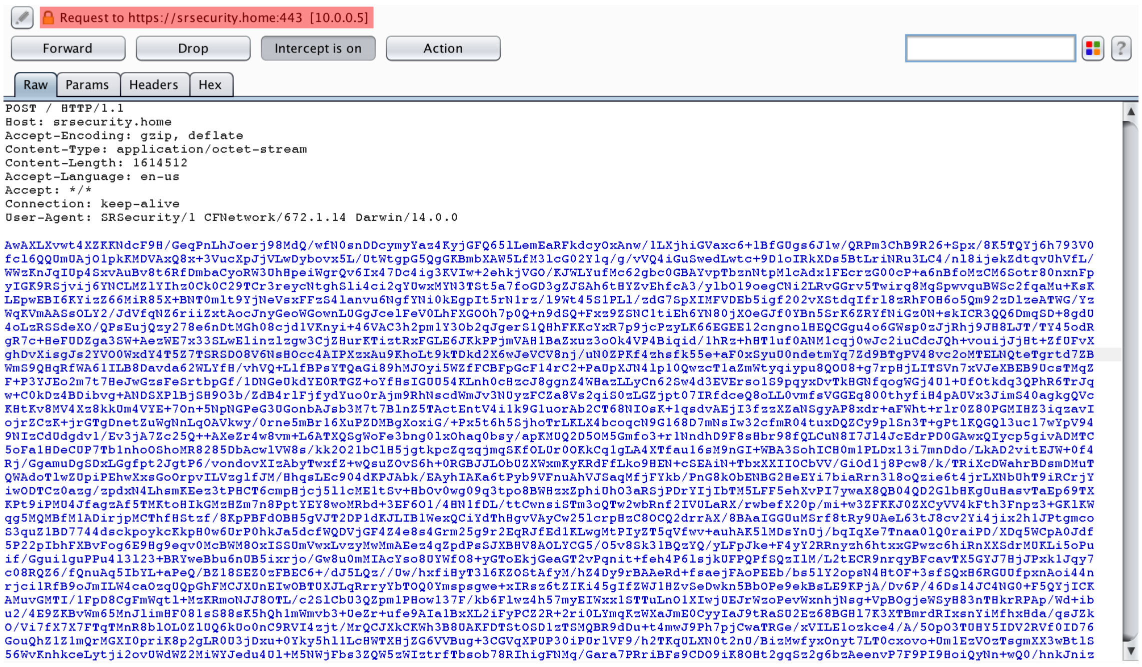
Error!

remote certificate doesn't match local certificate!

Figure 8.5: SSL pinning error

9 https://portswigger.net/burp

By installing the software “SSL Kill Switch 2”[10] on the jailbroken iOS 7.1.1 device the implemented SSL-pinning procedure is annulled. Due to the raised privileges on a jailbroken device, the application is able to patch low level Secure Transport APIs to disable the system’s verification mechanisms. Unfortunately, due to the limitation of the test device (see figure 8.4 on page 69), the exploit could not be tested on the most recent iOS version. The figure 8.6 on page 70 shows that the SSL-pinning is successfully disabled. Furthermore the application accepts the self-signed Burp Suite certificate and the encrypted traffic can be intercepted, without being noticed by the client.



```
POST / HTTP/1.1
Host: srsecurity.home
Accept-Encoding: gzip, deflate
Content-Type: application/octet-stream
Content-Length: 1614512
Accept-Language: en-us
Accept: */*
Connection: keep-alive
User-Agent: SRSecurity/1 CFNetwork/672.1.14 Darwin/14.0.0
```

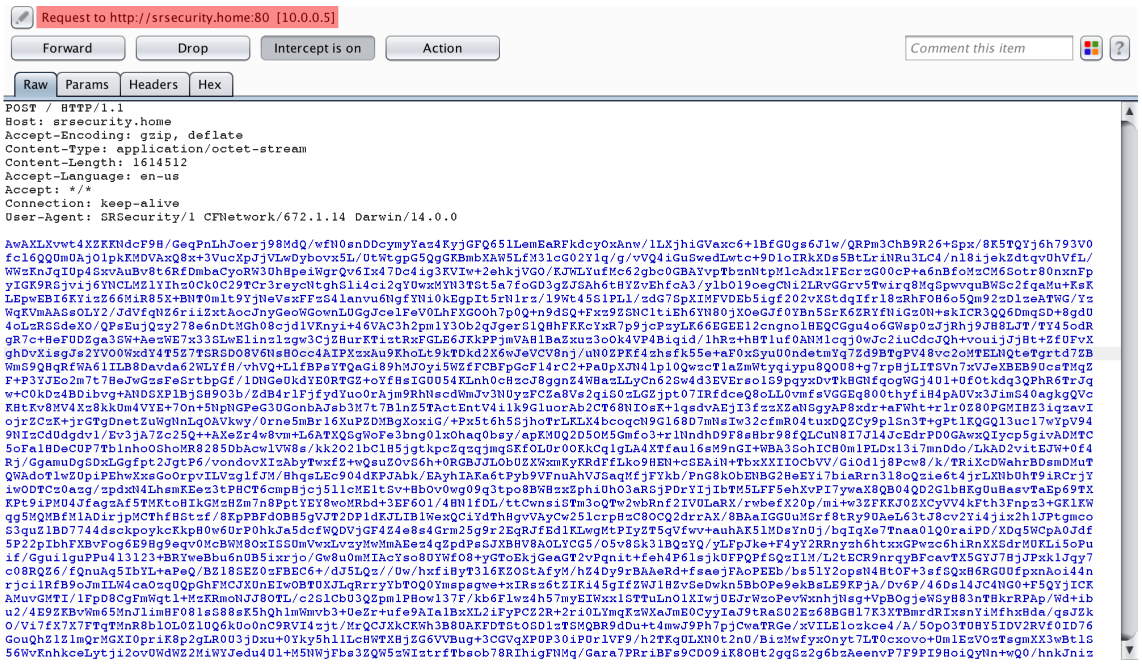
```
AwAXLXvwt4XZKKNdcF9H/GeqPnLhJoerj98MdQ/wfN0snDDcymyYaz4KyjGFQ651LemEaRFkdcyOxAnw/1LXjhiGVaxc6+1BfGUgs6J1w/QRPm3ChB9R26+Spx/8K5TQYj6h793V0
fcl6QQUmUAjO1pkKMDVAxQ8x+3VucXpJjVLwDybovx5L/UtWtgpG5QgGKBmbXAW5LfM3lcG02Y1q/g/vVQ4iGuSwedLwtc+9DloIRkXDs5BtLriNRu3LC4/nl8ijekZdtqvUhVfL/
WWzKnJqIUp4SxvAuBv8t6RfDmbaCyoRW3UhHpeiWgrQv6Ix47Dc4ig3KVIw+2ehkjVGO/KJWLYufMc62gbc0GBAYvpTbznNtpMlcAdx1FEcrzG00cP+a6nBfoMzCM6Sotr80nxnFp
yIGK9RSjvij6YNCLMZlYIhz0Ck0C29TCr3reycNtghSli4ci2qYUwxMYN3TSt5a7foGD3gZJSAh6tHYZvEhfcA3/ylbOl9oegCNi2LRvGGrv5Twirq8MqSpwvquBWSc2fqaMu+KsK
LEpwEBI6KYizZ66MiR85X+BNT0mlt9YjNeVsxFFzS4lanvu6NgfYNi0kEgpIt5rNlrz/l9Wt45SlPLl/zdG7SpXIMFVDEb5igf202vXStdqIfrl8zRhFOH6o5Qm92zDlzeATWG/Yz
WqKVmAASsOLY2/JdVfqNZ6riiZxtAocJnyGeoWGownLUGgJcelFeV0LhFXGOOh7p0Q+n9dSQ+Fxz9ZSNCltiEh6YN80jXOeGJf0YBn5SrK6ZRYfNiGz0N+skICR3QQ6DmqSD+8gdU
4oLzRSSdeXO/QPsEujQzy278e6nDtMGh08cjdlVKnyi+46VAC3h2pmlY3Ob2qJgerSlQHhFKKcYxR7p9jcPzyLK66EGEE12cngnolHEQCGgu4o6GWsp0zJjRhj9JH8LJT/TY45odR
gR7c+HeFUDZga3SW+AezWE7x33SLwElinzlzgw3CjZHurKTiztRxFGLE6JKkPPjmVAH1BaZxuz3oOk4VP4Biqid/1hRz+hHTluf0ANM1cqj0wJc2iuCdcJQh+vouijJjHt+ZfUFvX
ghDvXisgJs2YVO0WxdY4T5Z7TSRSDO8V6NsHOcc4AIPXzxAu9KhoLt9kTDkd2X6wJeVCV8nj/uN0ZPKf4zhsfk55e+aF0xSyuU0ndetmYq7Zd9BTgPV48vc2oMTELNQteTgrtd7ZB
WmS9QHqRfWA61ILB8Davda62WLYfH/vhVQ+LlfBPsYTQaGi89hMJOyi5WZfFCBFpGcFl4rC2+PaUpXJN4lp10QwzcTlaZmWtyqiypu8QOU8+g7rpHjLITSVn7xVJeXBEB9UcsTMqZ
F+P3YJEo2m7t7HeJwGzsFeSrtbpGf/1DNGeUkdYE0RTGZ+oYfHsIGUU54KLnh0cHzcJ8ggnZ4WHazLLyCn62Sw4d3EVErsolS9pqyxDvTkHGNfqogWGj4Ul+Uf0tkdq3QPhR6TrJq
w+C0kDz4BDibvg+ANDSXPlBjSH9O3b/ZdB4rlFjfydYuo0rAjm9RhNscdWmJv3NUyzFCZa8Vs2qiS0zLGZjpt07IRfdceQ8oLL0vmfsVGGEq800thyfiH4pAUVx3JimS40agkgQVc
KHtKv8MV4Xz8kkUm4VYE+7On+5NpNGPeG3UGonbAJsb3M7t7BlnZ5TActEntV4ilk9GluorAb2CT68NIOsK+1qsdvAEjI3fzzXZaNSgyAP8xdr+aFWht+rlr0Z80PGMIHZ3iqzavI
ojrZCzK+jrGTgDnetZuWgNnLqOAVkwy/0rne5mBr16XuPZDMBgXoxiG/+Px5t6h5SjhoTrLKLX4bcoqcN9Gl68D7mNsIw32cfmR04tuxDQZCy9plSn3T+gPtlKQGQl3ucl7wYpV94
9NIzCdUdgdvl/Ev3jA7Zc25Q++AXeZr4w8vm+L6ATXQSgWoFe3bng0lxOhaq0bsy/apKMUQ2D5OM5Gmfo3+rlNndhD9F8sHbr98fQLCuN8I7Jl4JcEdrPD0GAwxQIycp5givADMTC
5oFalHDeCUP7TblnhoOShoMR8285DbAcwlVW8s/kk2O2lbClH5jgtkpcZqzqjmqSKfOLUr0OKkCqlgLA4XTfaul6sM9nGI+WBA3SohICH0mlPLDxl3i7mnDdo/LkAD2vitEJW+0f4
Rj/GgamuDgSDxLGgfpt2JgtP6/vondovXIzAbyTwxfZ+wQsuZOvS6h+0RGBJJLObUZXWxmKyKRdFfLko9HEN+cSEAiN+TbxXXIIOCbVV/GiOdlj8Pcw8/k/TRiXcDWahrBDsmDMuT
QWAdoTlwZUpiPEhwXxsGoOrpvILVzglfJM/HhqsLEc904dKPJAbk/EAyhIAKa6tPyb9VFnuAhVJSaqMfjFYkb/PnG8kObENBG2HeEYi7biaRrn3l8oQzie6t4jrLXNbUhT9iRCrjY
iwODTCz0azg/zpdxN4LhsmKEez3tPHCT6cmpHjcj5llcME1tSv+HbOv0wg09q3tpo8BWHzxZphiUhO3aRSjPDrYIjIbTM5LFF5ehXvPI7ywaX8QB04QD2GlbHKgUuHasvTaEp69TX
KPt9iPMU4JfagzAf5TMKtoHIkGMzHZm7n8PptYEY8woMRbd+3EF6O1/4HN1fDL/ttCwnsiSTm3oQTw2wbRnf2IVULaRX/rwbefX20p/mi+w3ZFKKJ0ZXCyVV4kFth3Fnpz3+GKlKW
qg5MQMBfMlADirjpMCThfHStzf/8KpPBFdOBH5gVJT2DPldKJLIBlWexQCiYdThHgvVAyCw25lcrpHzC8OCQ2drrAX/8BAaIGGUuMSrf8tRy9UAeL63tJ8cv2Yi4jix2hlJPtgmco
S3quZlBD7744dsckpoykcKkpH0w6UrP0hkJa5dcfWQDVjGF4Z4e8s4Grm25g9r2EqRJfEdlKLwgMtPIyZT5qVfwv+auhAK5lMDsYnUj/bqIqXe7Tnaa0lQ0raiPD/XDq5WCpA0Jdf
5P22pIbhFXBvFog6E9Hg9eqv0McBWM8OxISSUmVwxLvzyMwMmAEez4qZpdPsSJXBHV8AOLYCG5/O5v8Sk3lBQzYQ/yLFpJke+F4yY2RRnyzh6htxxGPwzc6hiRnXXSdrMUKLi5oPu
if/GguilguPPu4l3l23+BRYweBbu6nUB5ixrjo/Gw8u0mMIAcYso8UYWfO8+yGToEkjGeaGT2vPqnit+feh4P6lsjkUFPQPfSQzIlM/L2tECR9nrqyBFcavTX5GYJ7HjJPxklJqy7
c08RQZ6/fQnuAq5IbYL+aPeQ/BZl8SEZ0zFBEC6+/dJ5LQz//Uw/hxfiHyT3l6KZOStAfyM/hZ4Dy9rBAAeRd+fsaejFAoPEEb/bs5lY2opsN4HtOF+3sfSQxH6RGUUfpxnAoi44n
rjcilRfB9oJmILW4caOzqUQpGhFMCJXUnEIwOBTUXJLqRrryYbTOQ0Ymspsgwe+xIRsz6tZIKi45gIfZWJlHZvSeDwkn5BbOPe9ekBsLE9KPjA/Dv6P/46Dsl4JC4NG0+F5QYjICK
AMuvGMTI/1FpD8CgFmWqtl+MzKRmoNJJ8OTL/c2SlCbU3QZpmlPHowl37F/kb6Flwz4h57myEIWxxlSTTuLnOlXIwjUEJrWzoPevWxnhjNsg+VpBOgjeWSyH83nTHkrRPAp/Wd+ib
u2/4E9ZKBvWm65MnJlimHF08lsS88sK5hQhlmWmvb3+UeZr+ufe9AIalBxXL2iFyPCZ2R+2ri0LYmqKzWXaJmE0CyyIaJ9tRaSU2Ez68BGHl7K3XTBmrdRIxsnYiMfhxHda/qsJZk
O/Vi7fX7X7FTqTMnR8blOL0ZlUQ6kUo0nC9RVI4zjt/MrQCJXkCKWh3B8UAKFDTStOSDlzTSMQBR9dDu+t4mwJ9Ph7pjCwaTRGe/xVILElozkce4/A/5OpO3TUHY5IDV2RVf0ID76
GouQhZlZlmQrMGXI0priK8p2gLR0U3jDxu+0Yky5hllLcHWTXHjZG6VVBug+3CGVqXPUP30iPUrlVF9/h2TKqULXN0t2nU/BizMwfyxOnyt7LT0cxovo+UmlEzVOzTsgmXX3wBtlS
56WvKnhkceLytji2ovUWdWZ2MiWYJedu4Ul+M5NWjFbs3ZQW5zWIztrfTbsob78RIhigFNMq/Gara7PRriBFs9CDO9iK8OHt2gqSz2g6bzAeenvP7F9PI9HoiQyNn+wQ0/hnkJniz
```

Figure 8.6: Intercepted encrypted image transmission

After forwarding the intercepted request, the user is presented with a success message, although the encrypted connection was broken and the data could be read in clear text in the proxy logs. Due to the additionally implemented encryption, the visible data is still useless without the appropriate keys. Despite beeing intercepted the data is signed with the clients *CPR-key* and encrypted and maced with the *MHM-key*.

In the final testcase the TLS encryption of the HTTP connection is disabled on purpose in order to simulate a broken TLS encryption layer. The setup is the same as shown in figure 8.4 on page 69 but without an encrypted connection. This time, the transmission can be intercepted by the proxy without the installation of additional software and the transmitted data is also visible to the Man-In-The-Middle as seen in figure 8.7 on page 71.

[10] https://github.com/nabla-c0d3/ssl-kill-switch2



```
POST / HTTP/1.1
Host: srsecurity.home
Accept-Encoding: gzip, deflate
Content-Type: application/octet-stream
Content-Length: 1614512
Accept-Language: en-us
Accept: */*
Connection: keep-alive
User-Agent: SRSecurity/1 CFNetwork/672.1.14 Darwin/14.0.0
```

```
AwAXLXvwt4XZKKNdcF9H/GeqPnLhJoerj98MdQ/wfN0snDDcymyYaz4KyjGFQ65lLemEaRFkdcyOxAnw/lLXjhiGVaxc6+lBfGUgs6J1w/QRPm3ChB9R26+Spx/8K5TQYj6h793V0
fcl6QQUmUAjOlpkKMDVAxQ8x+3VucXpJjVLwDybovx5L/UtWtgpG5QgGKBmbXAW5LfM3lcG02Y1q/g/vVQ4iGuSwedLwtc+9DloIRkXDs5BtLriNRu3LC4/nl8ijekZdtqvUhVfL/
WWzKnJqIUp4SxvAuBv8t6RfDmbaCyoRW3UhHpeiWgrQv6Ix47Dc4ig3KVIw+2ehkjVGO/KJWLYufMc62gbc0GBAYvpTbznNtpMlcAdx1FEcrzG00cP+a6nBfoMzCM6Sotr80nxnFp
yIGK9RSjvij6YNCLMZlYIhz0Ck0C29TCr3reycNtghSli4ci2qYUwxMYN3TSt5a7foGD3gZJSAh6tHYZvEhfcA3/ylbOl9oegCNi2LRvGGrv5Twirq8MqSpwvquBWSc2fqaMu+KsK
LEpwEBI6KYizZ66MiR85X+BNT0mlt9YjNeVsxFFzS4lanvu6NgfYNi0kEgpIt5rNlrz/l9Wt45SlPLl/zdG7SpXIMFVDEb5igf202vXStdqIfrl8zRhFOH6o5Qm92zDlzeATWG/Yz
WqKVmAASsOLY2/JdVfqNZ6riiZxtAocJnyGeoWGownLUGgJcelFeV0LhFXGOOh7p0Q+n9dSQ+Fxz9ZSNCltiEh6YN80jXOeGJf0YBn5SrK6ZRYfNiGz0N+skICR3QQ6DmqSD+8gdU
4oLzRSSdeXO/QPsEujQzy278e6nDtMGh08cjdlVKnyi+46VAC3h2pmlY3Ob2qJgerSlQHhFKKcYxR7p9jcPzyLK66EGEEl2cngnolHEQCGgu4o6GWsp0zJjRhj9JH8LJT/TY45odR
gR7c+HeFUDZga3SW+AezWE7x33SLwElinzlzgw3CjZHurKTiztRxFGLE6JKkPPjmVAHlBaZxuz3oOk4VP4Biqid/lhRz+hHTluf0ANMlcqj0wJc2iuCdcJQh+vouijJjHt+ZfUFvX
ghDvXisgJs2YVO0WxdY4T5Z7TSRSDO8V6NsHOcc4AIPXzxAu9KhoLt9kTDkd2X6wJeVCV8nj/uN0ZPKf4zhsfk55e+aF0xSyuU0ndetmYq7Zd9BTgPV48vc2oMTELNQteTgrtd7ZB
WmS9QHqRfWA6lILB8Davda62WLYfH/vhVQ+LlfBPsYTQaGi89hMJOyi5WZfFCBFpGcFl4rC2+PaUpXJN4lp10QwzcTlaZmWtyqiypu8QOU8+g7rpHjLITSVn7xVJeXBEB9UcsTMqZ
F+P3YJEo2m7t7HeJwGzsFeSrtbpGf/lDNGeUkdYE0RTGZ+oYfHsIGUU54KLnh0cHzcJ8ggnZ4WHazLLyCn62Sw4d3EVErsolS9pqyxDvTkHGNfqogWGj4Ul+Uf0tkdq3QPhR6TrJq
w+C0kDz4BDibvg+ANDSXPlBjSH9O3b/ZdB4rlFjfydYuo0rAjm9RhNscdWmJv3NUyzFCZa8Vs2qiS0zLGZjpt07IRfdceQ8oLL0vmfsVGGEq800thyfiH4pAUVx3JimS40agkgQVc
KHtKv8MV4Xz8kkUm4VYE+7On+5NpNGPeG3UGonbAJsb3M7t7BlnZ5TActEntV4ilk9GluorAb2CT68NIOsK+lqsdvAEjI3fzzXZaNSgyAP8xdr+aFWht+rlr0Z80PGMIHZ3iqzavI
ojrZCzK+jrGTgDnetZuWgNnLqOAVkwy/0rne5mBrl6XuPZDMBgXoxiG/+Px5t6h5SjhoTrLKLX4bcoqcN9Gl68D7mNsIw32cfmR04tuxDQZCy9plSn3T+gPtlKQGQl3ucl7wYpV94
9NIzCdUdgdvl/Ev3jA7Zc25Q++AXeZr4w8vm+L6ATXQSgWoFe3bng0lxOhaq0bsy/apKMUQ2D5OM5Gmfo3+rlNndhD9F8sHbr98fQLCuN8I7Jl4JcEdrPD0GAwxQIycp5givADMTC
5oFalHDeCUP7TblnhoOShoMR8285DbAcwlVW8s/kk2O2lbClH5jgtkpcZqzqjmqSKfOLUr0OKkCqlgLA4XTfaul6sM9nGI+WBA3SohICH0mlPLDxl3i7mnDdo/LkAD2vitEJW+0f4
Rj/GgamuDgSDxLGgfpt2JgtP6/vondovXIzAbyTwxfZ+wQsuZOvS6h+0RGBJJLObUZXWxmKyKRdFfLko9HEN+cSEAiN+TbxXXIIOCbVV/GiOdlj8Pcw8/k/TRiXcDWahrBDsmDMuT
QWAdoTlwZUpiPEhwXxsGoOrpvILVzglfJM/HhqsLEc904dKPJAbk/EAyhIAKa6tPyb9VFnuAhVJSaqMfjFYkb/PnG8kObENBG2HeEYi7biaRrn3l8oQzie6t4jrLXNbUhT9iRCrjY
iwODTCz0azg/zpdxN4LhsmKEez3tPHCT6cmpHjcj5llcMEltSv+HbOv0wg09q3tpo8BWHzxZphiUhO3aRSjPDrYIjIbTM5LFF5ehXvPI7ywaX8QB04QD2GlbHKgUuHasvTaEp69TX
KPt9iPMU4JfagzAf5TMKtoHIkGMzHZm7n8PptYEY8woMRbd+3EF6Ol/4HNlfDL/ttCwnsiSTm3oQTw2wbRnf2IVULaRX/rwbefX20p/mi+w3ZFKKJ0ZXCyVV4kFth3Fnpz3+GKlKW
qg5MQMBfMlADirjpMCThfHStzf/8KpPBFdOBH5gVJT2DPldKJLIBlWexQCiYdThHgvVAyCw25lcrpHzC8OCQ2drrAX/8BAaIGGUuMSrf8tRy9UAeL63tJ8cv2Yi4jix2hlJPtgmco
S3quZlBD7744dsckpoykcKkpH0w6UrP0hkJa5dcfWQDVjGF4Z4e8s4Grm25g9r2EqRJfEdlKLwgMtPIyZT5qVfwv+auhAK5lMDsYnUj/bqIqXe7Tnaa0lQ0raiPD/XDq5WCpA0Jdf
5P22pIbhFXBvFog6E9Hg9eqv0McBWM8OxISSUmVwxLvzyMwMmAEez4qZpdPsSJXBHV8AOLYCG5/O5v8Sk3lBQzYQ/yLFpJke+F4yY2RRnyzh6htxxGPwzc6hiRnXXSdrMUKLi5oPu
if/GguilguPPu4l3l23+BRYweBbu6nUB5ixrjo/Gw8u0mMIAcYso8UYWfO8+yGToEkjGeaGT2vPqnit+feh4P6lsjkUFPQPfSQzIlM/L2tECR9nrqyBFcavTX5GYJ7HjJPxklJqy7
c08RQZ6/fQnuAq5IbYL+aPeQ/BZl8SEZ0zFBEC6+/dJ5LQz//Uw/hxfiHyT3l6KZOStAfyM/hZ4Dy9rBAAeRd+fsaejFAoPEEb/bs5lY2opsN4HtOF+3sfSQxH6RGUUfpxnAoi44n
rjcilRfB9oJmILW4caOzqUQpGhFMCJXUnEIwOBTUXJLqRrryYbTOQ0Ymspsgwe+xIRsz6tZIKi45gIfZWJlHZvSeDwkn5BbOPe9ekBsLE9KPjA/Dv6P/46Dsl4JC4NG0+F5QYjICK
AMuvGMTI/lFpD8CgFmWqtl+MzKRmoNJJ8OTL/c2SlCbU3QZpmlPHowl37F/kb6Flwz4h57myEIWxxlSTTuLnOlXIwjUEJrWzoPevWxnhjNsg+VpBOgjeWSyH83nTHkrRPAp/Wd+ib
u2/4E9ZKBvWm65MnJlimHF08lsS88sK5hQhlmWmvb3+UeZr+ufe9AIalBxXL2iFyPCZ2R+2ri0LYmqKzWXaJmE0CyyIaJ9tRaSU2Ez68BGHl7K3XTBmrdRIxsnYiMfhxHda/qsJZk
O/Vi7fX7X7FTqTMnR8blOL0ZlUQ6kUo0nC9RVI4zjt/MrQCJXkCKWh3B8UAKFDTStOSDlzTSMQBR9dDu+t4mwJ9Ph7pjCwaTRGe/xVILElozkce4/A/5OpO3TUHY5IDV2RVf0ID76
GouQhZlZlmQrMGXI0priK8p2gLR0U3jDxu+0Yky5hllLcHWTXHjZG6VVBug+3CGVqXPUP30iPUrlVF9/h2TKqULXN0t2nU/BizMwfyxOnyt7LT0cxovo+UmlEzVOzTsgmXX3wBtlS
56WvKnhkceLytji2ovUWdWZ2MiWYJedu4Ul+M5NWjFbs3ZQW5zWIztrfTbsob78RIhigFNMq/Gara7PRriBFs9CDO9iK8OHt2gqSz2g6bzAeenvP7F9PI9HoiQyNn+wQ0/hnkJniz
```

Figure 8.7: Intercepted image transmission

Although the possible Man-In-The-Middle can extract the transmitted data, the sent data is still signed with the clients *CPR-key* and encrypted and maced with the *MHM-key*, and therefore the original data is still inaccessible without the corresponding keys.

The following listings shows the matching MD5 hash sums of the data stored on the client (see listing 8.4 on page 71) and the data retrieved from the intercepted connection (see listing 8.5 on page 71).

```
MichiPhone:# cd /private/var/mobile/Applications/ACF2-4A0B-BA29-CD4B7D/Documents
MichiPhone:# cat michi_1444727177.341047.data \
            | base64 \
            | tr -d " =\t\n\r" \
             > michi_1444727177.341047.data.base64

MichiPhone:# md5sum michi_1444727177.341047.data.base64

67eb7765167bf030ddcaa006356db266 michi_1444727177.341047.data.base64
```

Listing 8.4: MD5 hash sum of client data

```
Neon:$ md5 intercepted.data.base64

MD5 (intercepted.data.base64) = 67eb7765167bf030ddcaa006356db266
```

Listing 8.5: MD5 hash sum of intercepted data

8.1.5 Results

The analysis showed that the recommended cipher suite is not supported by the iOS version used on the test device, although Apple uses it, as described in [10]. Therefore the cipher string had to be changed to:

```
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
```

Compared to the recommended cipher string, only the MAC method is changed from AEAD to SHA-384. According to the document "iOS Security for iOS 9.0 or later" of September 2015 (see [12]), iOS 9 supports AES with GCM mode. Due to the limitations of the iOS 7.1.1 test device, the cipher suite support of iOS 9 could not be tested.

The used cipher suite also provides forward secrecy, which means that a decryption of the secured traffic is not possible, even if an attacker gains access to the server's private key (*SPR-key*).

The implemented SSL-pinning method was easily annulled by the software "SSL Kill Switch 2"[11] on the jailbroken iOS 7.1.1 device. This software uses the elevated privileges to patch the low level Secure Transport API. The software could not be tested on a recent iOS version, due to the restricted test setup.

Disabling the SSL-pinning allowed a MITM-attack without the user even noticing. Only the implemented additional layer of security provided a last resort of cryptographic secrecy to the transmitted data. Without this implemented layer, the transmitted data could have been read in plain text.

8.2 Storage Analysis

8.2.1 General

During the image saving process, as described in chapter 6.2 on page 46, the signed, encrypted and maced images are stored on the device storage. The following sections describe the threats and concerns considering the storage of data on the device storage. Recommendations for the storing of data are gathered and the file structure of the prototype application is analyzed.

8.2.2 Threats

According to the chapters 4.3.1 and 4.3.2 on page 24 and 25 the stored data on an iOS device is protected by the operating system's File-System Encryption and Data Protection mechanisms. The functionality of these components can be mitigated or even annulled by jailbreaking or poor configuration choices. Due to the nature of Apple's Hierarchical File System (HFS) journaling operations, experienced forensic examiners may also stand a chance to recover deleted files [76].

8.2.3 Recommendations

At first, developers should deliberate whether it is necessary to store certain data on the device storage. Especially, privacy critical data should not be stored on a mobile device since it is relatively more inclined to being stolen than from other platforms.

[11] https://github.com/nabla-c0d3/ssl-kill-switch2

If it is unavoidable to store data on the device, the operating systems security mechanisms should be thoroughly evaluated and configured.

The File-System Encryption's weakest point is jailbreaking, since no passcode is involved in the File-System Encryption mechanism. This enables the attacker to gain root access to the OS, without knowing any of the encryption keys [69]. Zdziarski (see [76] chapter 13) describes a number of different jailbreak detection mechanisms, which can be implemented to prevent the application to store sensitive information on a jailbroken device.

Although the Data Protection depends on the passcode and strengthens the level of security compared to the File-System Encryption, a passcode which matches predefined criterias and length should be configured, to mitigate the possibility of successful brute force attacks. Furthermore the level of security of Data Protection also strongly depends on the configuration. As elaborated in chapter 4.3.2 on page 25 there are different protection classes, which can be configured by the developer. The developer should have knowledge of the different kind of classes and always chose the strictest possible.

Due to the possibility of broken or bypassed native OS security mechanisms, the developer should implement an additional layer of security. This can be achieved by encrypting the files before storing them to the device storage. The encryption keys should also be handled with care and should not be stored on the file system. An approach to a hierarchic key handling architecture is described in the "Architecture" chapter on page 40.

Various secure file wiping mechanisms are also described in Zdziarski's book (see [76] chapter 11) to mitigate the risk of deleted files being recovered. He also lists the US DoD's requirements for destroying classified data, which are implemented in the application prototype (see 7.3 on page 55).

8.2.4 Analysis

The following file structure (see listing 8.6 on page 73) shows the creation of a SSH connection to the jailbroken test device (iPhone[12] 4S 32GB (Model MD242D/A) with iOS 7.1.1[13]) and switching to the applications directory (/var/mobile/Applications/9C377D21-ED90-4BA1-85A7-D46B9C15D9ED):

```
MichiPhone:# ls *

Documents:

michi_1444727177.341047.data



Library:

Caches/   Preferences/



SRSecurity.app:

```

[12] http://www.apple.com/iphone
[13] http://www.apple.com/ios

```
Base.lproj/   Info.plist   OverlayView.nib   PkgInfo   SRSecurity*
  ↪ _CodeSignature/   embedded.mobileprovision   srsecurity.home.der



tmp:
```

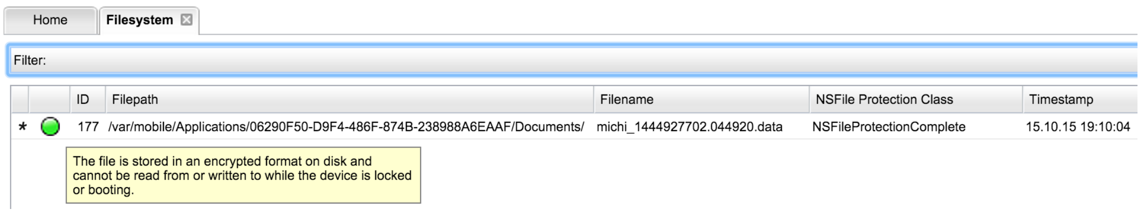
Listing 8.6: Application file structure

These four folders in the application root directory can be described as follows (see [8]):

- **SRSecurity.app:** The application's bundle. This directory is read-only. Any change to this directory alters the app's signature and prevents it from launching. In the prototype application's case the SSL-pinning key (srsecurity.home.der) is stored in this directory.
- **Documents:** This folder holds user-generated data and can be backed up on iTunes, depending on the developers choice. In the prototype application's case, this directory holds the signed, maced and encrypted images of the application users.
- **Library:** The Library folder stores application specific files. The sub-folder Caches is used to store support files which can be recreated easily. The Preferences sub-folder holds application specific preferences and should only be altered by using the CFPreferences API.
- **tmp:** The top-level tmp folder can be used to store temporary files. It may be purged when the app is not running.

The application Snoop-it[14] (version 1.0.10) assists security assessment of iOS applications. It comes with features to monitor, analyze and manipulate applications. In this section the file system access feature is used to determine details for the files stored within the prototype application. Other features are demonstrated in the section "Runtime Analysis" on page 76.

Snoop-it can only be used on jailbroken devices and starts a web interface to provide the features via browser. Filtering the files created by the application user, the image shows that the data-file is protected by the Data Protection Class "NSFileProtectionComplete", which represents the strictest level of security, concerning the protection classes (see figure 8.8 on page 74).



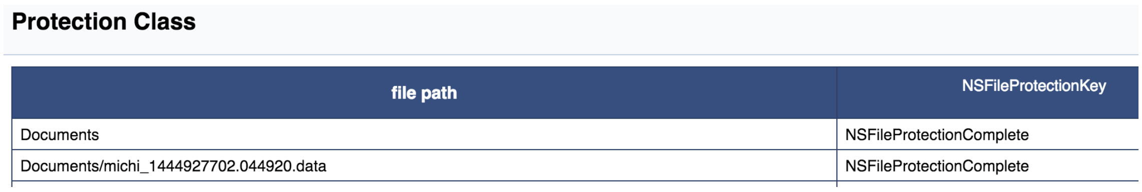
		ID	Filepath	Filename	NSFile Protection Class	Timestamp
*	●	177	/var/mobile/Applications/06290F50-D9F4-486F-874B-238988A6EAAF/Documents/	michi_1444927702.044920.data	NSFileProtectionComplete	15.10.15 19:10:04

The file is stored in an encrypted format on disk and cannot be read from or written to while the device is locked or booting.

Figure 8.8: Snoop-it - Data Protection Class of the stored image data

The application iNalyzer[15] (version 5.5b), which can also only be installed on jailbroken iOS devices, is another iOS penetration testing framework. It extracts the application and gathers information like classes, stored files, view controllers and strings. The figure 8.9 on page 75 shows the result of the iNalyzer protection class analysis.

[14] https://code.google.com/p/snoop-it
[15] https://appsec-labs.com/inalyzer



Protection Class

file path	NSFileProtectionKey
Documents	NSFileProtectionComplete
Documents/michi_1444927702.044920.data	NSFileProtectionComplete

Figure 8.9: iNalyzer - Data Protection Class of the stored image data

The result for both tools is that the image file is stored under the strictest possible Data Protection Class.

Rooting the device enables the user to browse the device's file system and locate the stored image data, but still blocks the access on a locked device, due to the Data Protection Class (see listing 8.7 on page 75).

```
MichiPhone:# cat michi_1444927702.044920.data

cat: michi_1444927702.044920.data: Operation not permitted
```

Listing 8.7: Data Protection on passcode locked, jailbroken device

After unlocking the device, the data can be accessed as shown in listing 8.8 on page 75.

```
MichiPhone:# less michi_1444927702.044920.data

"michi_1444927702.044920.data" may be a binary file.  See it anyway?

^C^@0<A8>D;^]<86><E5>g<<99>J[<B1>^^{<96>"^ZE<84>^\<FB><AB><EE><89>
"b49SZ<BD>Sz<E5><A3>Q^Z<B7>^M,<F9>^C<90>^S<C1><88>mBC><95>fc<C1>
E^?^O<F3><CC>2<CF>t^^4<D4><F1><BF><E1><BD><E4>]I^U^YXse<D1><AA>
~<84>L<ED><AF><C3><B8><DB><D0>.<F6><DE><FA>^F<D1><A6><9F>\E<A5>
...
>Sz<E5><A3>Q^Z<B7>^M,<F9>^C<90>^S<C1><88>mBC><95>fc<C1><B4>L<B9>
E^?^O<F3><CC>2<CF>t^^4<D4><F1><BF><E1><BD><E4>]I^U^YXse<D1><AA>
~<84>L<ED><AF><C3><B8
```

Listing 8.8: Data Protection on unlocked, jailbroken device

Although, in this case, the File-System Encryption and the Data Protection are abrogated, the additional implemented layer of security, the encryption of the image file, is still intact and protects the image data from unauthorized access.

8.2.5 Results

The File-System Encryption is annulled by jailbreaking the test device. The used tools show that the Data Protection for the signed, maced and encrypted images is configured to use the strictest possible protection class.

Although the file system was accessed with elevated privileges, the Data Protection still denied access to the stored images on the passcode locked device. An error message was thrown and the files could not be read.

After unlocking the device by entering the correct passcode, the Data Protection mechanism granted access to the stored signed, maced and encrypted images. Even after bypassing the operating system security mechanism, the stored image files were still cryptographically unreadable without access to the corresponding keys.

8.3 Runtime Analysis

8.3.1 General

In this chapter the variety of features of different mobile penetration testing and security assessment tools is utilized to analyze the prototype application. Runtime threats and recommendations to mitigate these threats are presented. The analysis section shows the information which can be gathered by tools exploiting this kind of threats. The employed tools can only be installed on a device with root privileges and which allows the frameworks to create a system level insight into the application. The test device is a jailbroken iPhone[16] 4S 32GB (Model MD242D/A) with iOS 7.1.1[17].

8.3.2 Threats

Privilege elevation, like jailbreaking, is one of the most crucial threats to an applications runtime security. As demonstrated in the "Analysis" section (see 8.3.4 on page 77), the used security assessment applications Snoop-it[18] (version 1.0.10) and iNalyzer[19] (version 5.5b) are able to gather information from memory and help to reverse engineer the applications architecture. Further features are for example the identification of used sensitive APIs, the listing of keychain entries and the gathering of information of deployed database and plist files. Further threats and recommendations to prevent them are discussed in the next section.

8.3.3 Recommendations

As described in chapter 6.3 on page 50, at first, it has to be discussed whether data should be stored to device storage or held in memory. Depending on the place where the data is stored, the threat vectors diff. If held in memory, the data is only accessible during runtime. The OS provides features to clean-up the memory, but Zdziarski describes methods to securely overwrite and wipe information held in memory (see [76]). The developer does not have to rely on the OS features, which may be altered by a malicious software, but overwrites the data in memory directly. In case of the prototype application the password for the *KDF-key* and *HKDF-key* should be wiped from memory right after the PBKDF has finished in order to reduce the threat level of unintended leakage. View controllers should also only be initiated at the moment they are needed to prevent an attacker to load it directly from memory and for example annul the authentication and login mechanism. Another important step to prevent data leakage during runtime is to evaluate the default settings and values of the operating system and to follow best practice guidelines [69]. To mitigate the possibility of side channel attacks, APIs and external framework should be thoroughly reviewed and used only on a limited basis if not at all avoided [69]. Mechanisms to prevent unwanted debugging or tampering with the application can also be implemented (see [76]). In a company environment, a MDM can be used to prevent alteration of the mobile client and to

[16] http://www.apple.com/iphone
[17] http://www.apple.com/ios
[18] https://code.google.com/p/snoop-it
[19] https://appsec-labs.com/inalyzer

enforce security policies [25]. These kind of tools can also prevent a user from elevating its privileges in an unauthorized way, for example by jailbreaking the device. On a jailbroken device the privileges can be used to access data directly in memory and to alter or bypass the operating system's functions, as utilized in the applications Snoop-it and iNalyzer, described in the "Threats" section (see 8.3.2 on page 76).

8.3.4 Analysis

Snoop-it

Snoop-it's features are divided into 3 categories: "Monitoring", "Analysis" and "Runtime Manipulation". In the following chapter, the features of these three categories are used to analyze the prototype application.

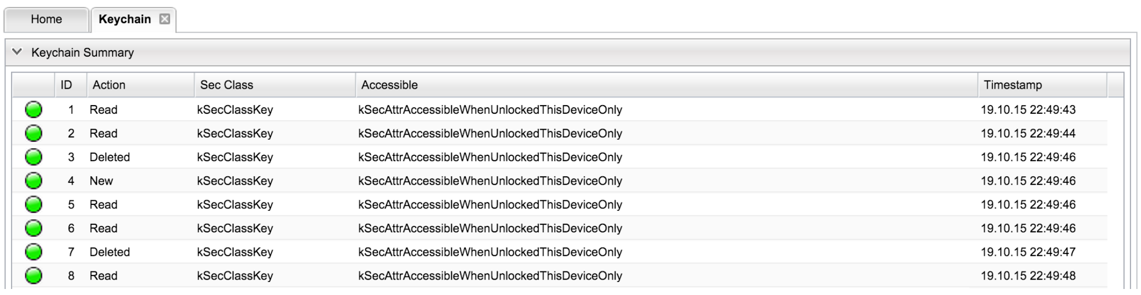
Monitoring:

Filesystem:

The file system is analyzed in chapter 8.2 on page 72.

Keychain:

The "Keychain" function of Snoop-it shows a summary of the actions performed on the keychain. Before the user logs in, the list is empty. After a successful login the entries shown in figure 8.10 on page 77 are created in the keychain summary table.

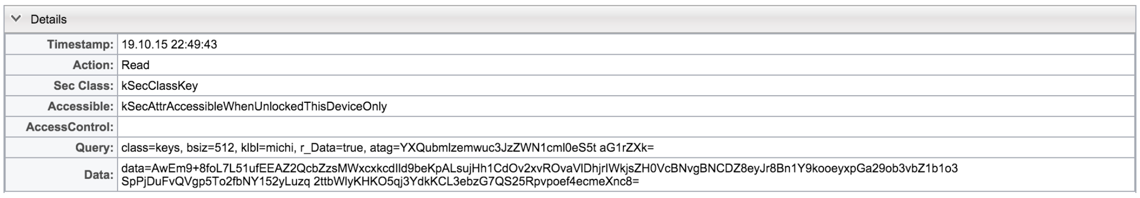


	ID	Action	Sec Class	Accessible	Timestamp
🟢	1	Read	kSecClassKey	kSecAttrAccessibleWhenUnlockedThisDeviceOnly	19.10.15 22:49:43
🟢	2	Read	kSecClassKey	kSecAttrAccessibleWhenUnlockedThisDeviceOnly	19.10.15 22:49:44
🟢	3	Deleted	kSecClassKey	kSecAttrAccessibleWhenUnlockedThisDeviceOnly	19.10.15 22:49:46
🟢	4	New	kSecClassKey	kSecAttrAccessibleWhenUnlockedThisDeviceOnly	19.10.15 22:49:46
🟢	5	Read	kSecClassKey	kSecAttrAccessibleWhenUnlockedThisDeviceOnly	19.10.15 22:49:46
🟢	6	Read	kSecClassKey	kSecAttrAccessibleWhenUnlockedThisDeviceOnly	19.10.15 22:49:46
🟢	7	Deleted	kSecClassKey	kSecAttrAccessibleWhenUnlockedThisDeviceOnly	19.10.15 22:49:47
🟢	8	Read	kSecClassKey	kSecAttrAccessibleWhenUnlockedThisDeviceOnly	19.10.15 22:49:48

Figure 8.10: Keychain - list

As indicated by the green signal in figure 8.10 on page 77, the keychain entries are protected. The access group for all entries is "kSecAttrAccessibleWhenUnlockedThisDeviceOnly", which means that these keychain entries are only available to the application when the device is unlocked and the entries can not be moved off-device, e.g. backed up.

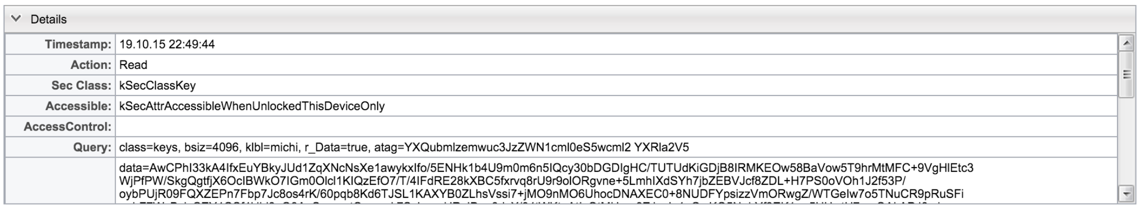
Due to Snoop-it's privileges, it is able to access details, like the data stored into the keychain. Altough, as a result of the additionally implemented encryption layer, the *MHM-key* and the *CPR-key* are encrypted before they are stored in the keychain and are visible in an encrypted form. Both keys can only be decrypted by providing the valid user and password combination. Figure 8.11 on page 78 shows the *MHM-key*, figure 8.12 on page 78 shows the *CPR-key* and figure 8.13 on page 78 shows the *CPUB-key*, stored in the keychain.



Details

Timestamp:	19.10.15 22:49:43
Action:	Read
Sec Class:	kSecClassKey
Accessible:	kSecAttrAccessibleWhenUnlockedThisDeviceOnly
AccessControl:	
Query:	class=keys, bsiz=512, klbl=michi, r_Data=true, atag=YXQubmlzemwuc3JzZWN1cml0eS5t aG1rZXk=
Data:	data=AwEm9+8foL7L51ufEEAZ2QcbZzsMWxcxkcdIld9beKpALsujHh1CdOv2xvROvaVIDhjrIWkjsZH0VcBNvgBNCDZ8eyJr8Bn1Y9kooeyxpGa29ob3vbZ1b1o3 SpPjDuFvQVgp5To2fbNY152yLuzq 2ttbWlyKHKO5qj3YdkKCL3ebzG7QS25Rpvpoef4ecmeXnc8=

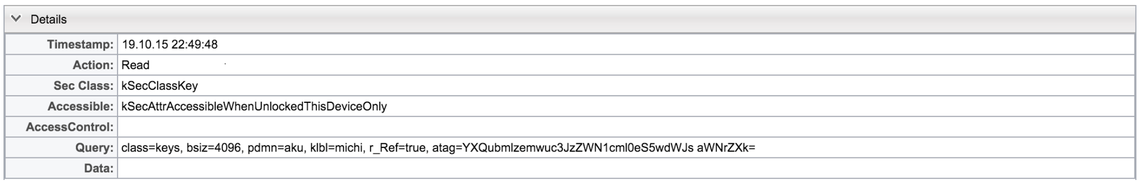
Figure 8.11: Keychain - MHM-key



Details

Timestamp:	19.10.15 22:49:44
Action:	Read
Sec Class:	kSecClassKey
Accessible:	kSecAttrAccessibleWhenUnlockedThisDeviceOnly
AccessControl:	
Query:	class=keys, bsiz=4096, klbl=michi, r_Data=true, atag=YXQubmlzemwuc3JzZWN1cml0eS5wcml2 YXRla2V5
	data=AwCPhI33kA4IfxEuYBkyJUd1ZqXNcNsXe1awykxIfo/5ENHk1b4U9m0m6n5IQcy30bDGDIgHC/TUTUdKiGDjB8IRMKEOw58BaVow5T9hrMtMFC+9VgHIEtc3 WjPfPW/SkgQgtfjX6OcIBWkO7IGm0OIcl1KIQzEfO7/T/4IFdRE28kXBC5fxrvq8rU9r9oIORgvne+5LmhIXdSYh7jbZEBVJcf8ZDL+H7PS0oVOh1J2f53P/ oybPUjR09FQXZEPn7Fbp7Jc8os4rK/60pqb8Kd6TJSL1KAXYB0ZLhsVssi7+jMO9nMO6UhocDNAXEC0+8NUDFYpsizzVmORwgZ/WTGeIw7o5TNuCR9pRuSFi

Figure 8.12: Keychain - CPR-key



Details

Timestamp:	19.10.15 22:49:48
Action:	Read
Sec Class:	kSecClassKey
Accessible:	kSecAttrAccessibleWhenUnlockedThisDeviceOnly
AccessControl:	
Query:	class=keys, bsiz=4096, pdmn=aku, klbl=michi, r_Ref=true, atag=YXQubmlzemwuc3JzZWN1cml0eS5wdWJs aWNrZXk=
Data:	

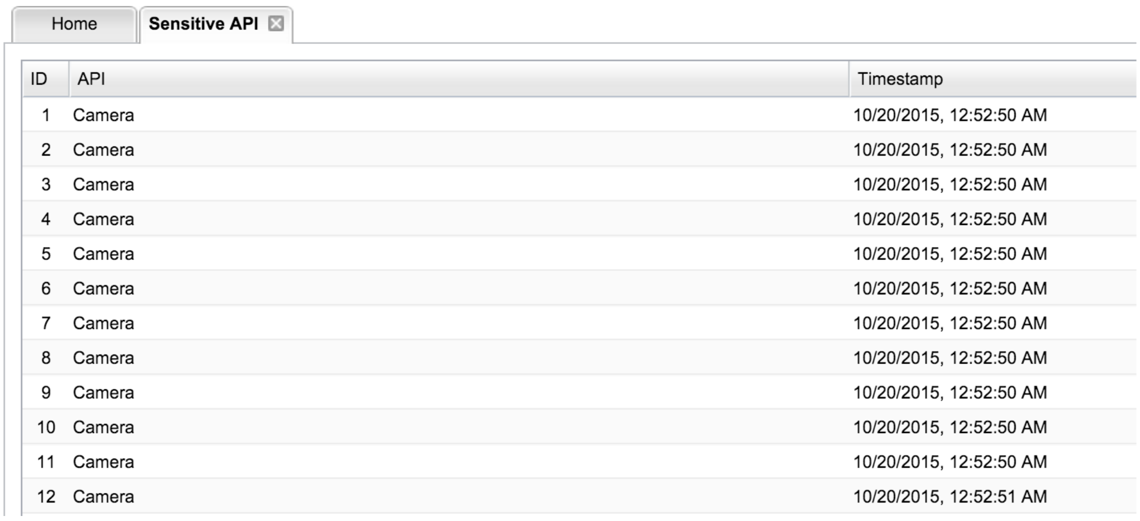
Figure 8.13: Keychain - CPUB-key

Network:

The network connections are analyzed in chapter 8.1 on page 61.

Sensitive API:

"Sensitive API" displays APIs used, which hold or can capture sensitive information, like the picture library, the address book or the camera. As is shown in figure 8.14 on page 78, the only sensitive API's used by the application is the camera. Since the camera functionality is necessarily a core component of the application, the use of this API can not be avoided.



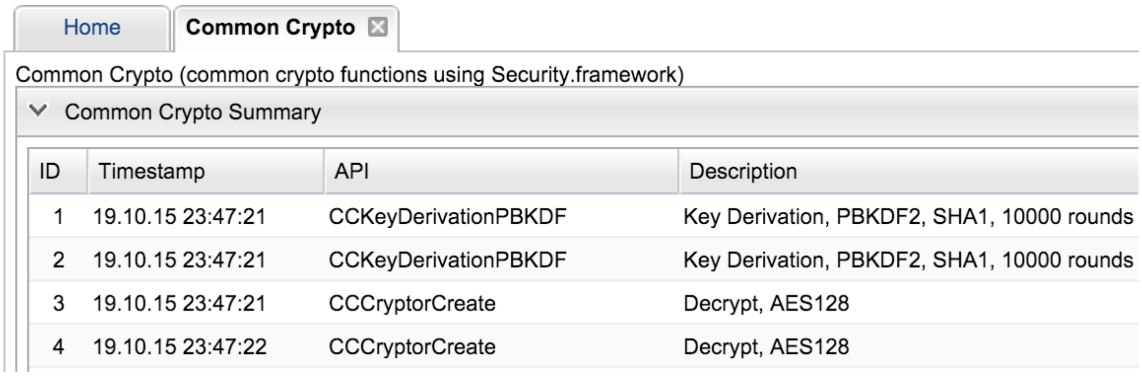
ID	API	Timestamp
1	Camera	10/20/2015, 12:52:50 AM
2	Camera	10/20/2015, 12:52:50 AM
3	Camera	10/20/2015, 12:52:50 AM
4	Camera	10/20/2015, 12:52:50 AM
5	Camera	10/20/2015, 12:52:50 AM
6	Camera	10/20/2015, 12:52:50 AM
7	Camera	10/20/2015, 12:52:50 AM
8	Camera	10/20/2015, 12:52:50 AM
9	Camera	10/20/2015, 12:52:50 AM
10	Camera	10/20/2015, 12:52:50 AM
11	Camera	10/20/2015, 12:52:50 AM
12	Camera	10/20/2015, 12:52:51 AM

Figure 8.14: Sensitive API - list

Common Crypto:

The Common Crypto API (see [6]) is part of the iOS SDK. The API's Password Based Key Derivation Function (PBKDF), encryption and decryption functionalities are used in the prototype application. Snoop-it logs these API calls and gathers detailed information from memory.

The following figure 8.15 on page 79 shows the Common Crypto functions used throughout the login process: two PBKDF calls, to create the *KDF-key* and the *HKDF-key*, and two decryption calls, to decrypt *MHM-key* and the *CPR-key*.



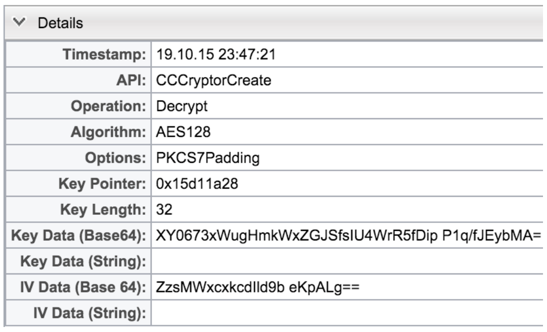
Common Crypto (common crypto functions using Security.framework)

Common Crypto Summary

ID	Timestamp	API	Description
1	19.10.15 23:47:21	CCKeyDerivationPBKDF	Key Derivation, PBKDF2, SHA1, 10000 rounds
2	19.10.15 23:47:21	CCKeyDerivationPBKDF	Key Derivation, PBKDF2, SHA1, 10000 rounds
3	19.10.15 23:47:21	CCCryptorCreate	Decrypt, AES128
4	19.10.15 23:47:22	CCCryptorCreate	Decrypt, AES128

Figure 8.15: Common Crypto - list login process

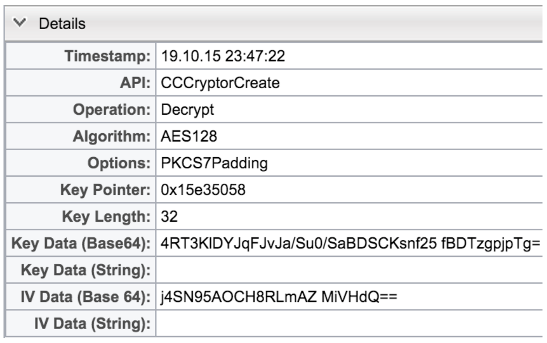
The PBKDF entries do not hold any additional information, but the decryption processes reveal detailed information like the key length, the padding and even the plain text key and IV. The two figures, figure 8.16a on page 79 and figure 8.16b on page 79, represent the decryption details of the *MHM-key* and the *CPR-key*.



Details

Timestamp:	19.10.15 23:47:21
API:	CCCryptorCreate
Operation:	Decrypt
Algorithm:	AES128
Options:	PKCS7Padding
Key Pointer:	0x15d11a28
Key Length:	32
Key Data (Base64):	XY0673xWugHmkWxZGJSfsIU4WrR5fDip P1q/fJEybMA=
Key Data (String):	
IV Data (Base 64):	ZzsMWxcxkcdIld9b eKpALg==
IV Data (String):	

(a) Common Crypto - MHM-key decryption



Details

Timestamp:	19.10.15 23:47:22
API:	CCCryptorCreate
Operation:	Decrypt
Algorithm:	AES128
Options:	PKCS7Padding
Key Pointer:	0x15e35058
Key Length:	32
Key Data (Base64):	4RT3KlDYJqFJvJa/Su0/SaBDSCKsnf25 fBDTzgpjpTg=
Key Data (String):	
IV Data (Base 64):	j4SN95AOCH8RLmAZ MiVHdQ==
IV Data (String):	

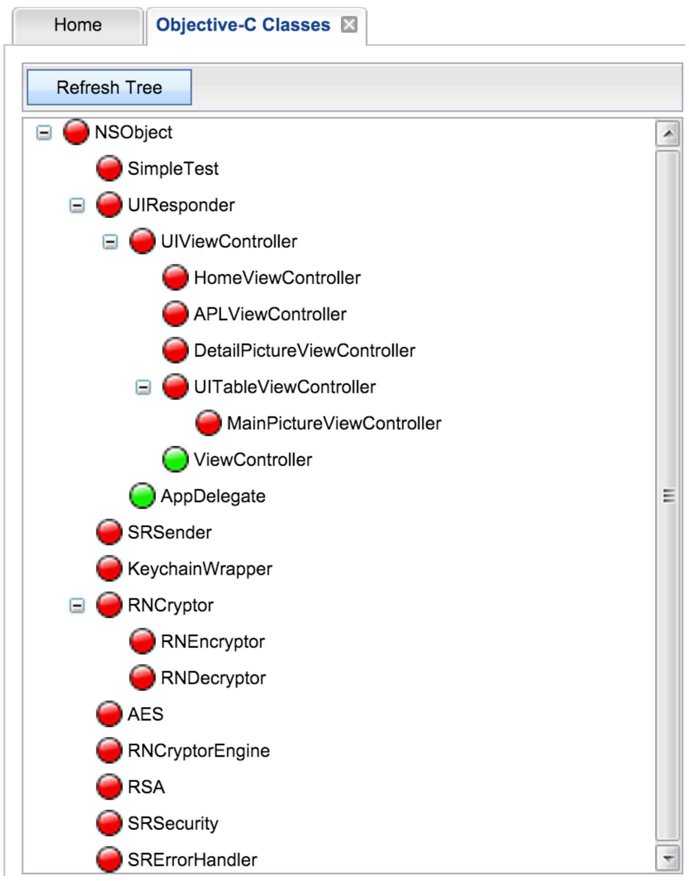
(b) Common Crypto - CPR-key decryption

Figure 8.16: Common Crypto - invoked methods

Analysis:

Objective-C Classes:

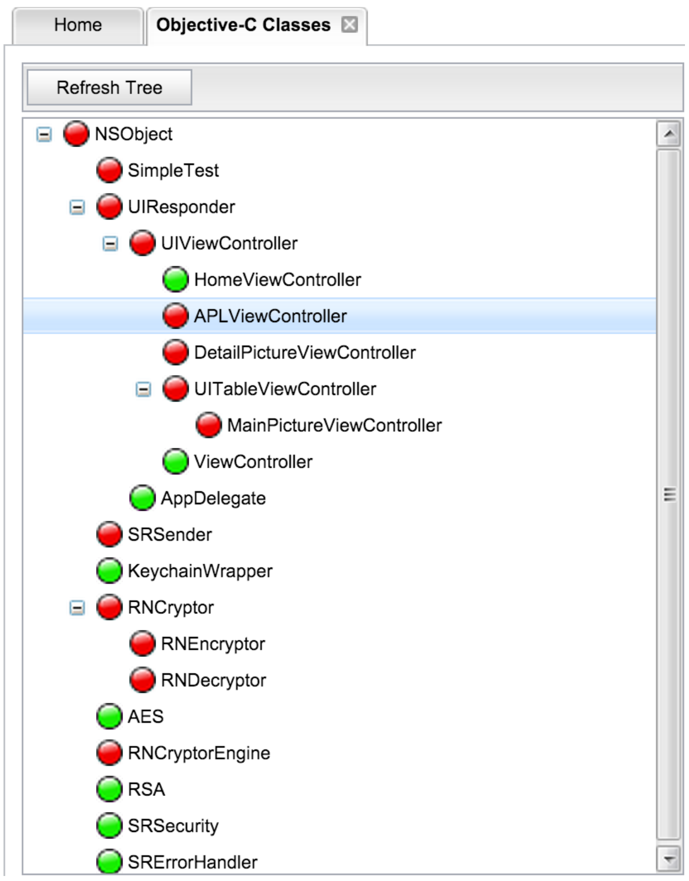
This feature lists the objective-c classes used in the application. The green symbol indicates that currently an instance of the class is available. Before the user logs in, only instances of AppDelegate and the main ViewController are created (see figure 8.17 on page 80).



```
⊟ 🔴 NSObject
     🔴 SimpleTest
   ⊟ 🔴 UIResponder
     ⊟ 🔴 UIViewController
          🔴 HomeViewController
          🔴 APLViewController
          🔴 DetailPictureViewController
        ⊟ 🔴 UITableViewController
              🔴 MainPictureViewController
          🟢 ViewController
       🟢 AppDelegate
     🔴 SRSender
     🔴 KeychainWrapper
   ⊟ 🔴 RNCryptor
       🔴 RNEncryptor
       🔴 RNDecryptor
     🔴 AES
     🔴 RNCryptorEngine
     🔴 RSA
     🔴 SRSecurity
     🔴 SRErrorHandler
```

Figure 8.17: Objective-C Classes - list before login

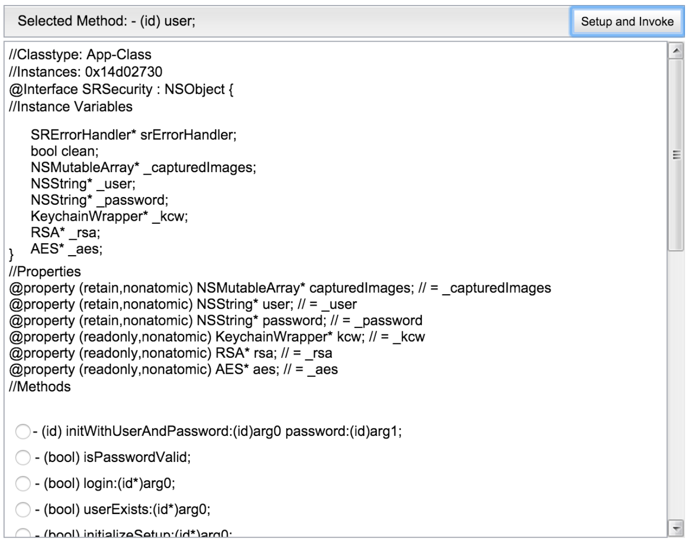
After the user logs in, the HomeViewController is created and instances of the classes SRSecurity, AES, RSA, KeychainWrapper and SRErrorHandler are available (see figure 8.18 on page 81).



```
⊟ 🔴 NSObject
     🔴 SimpleTest
   ⊟ 🔴 UIResponder
      ⊟ 🔴 UIViewController
           🟢 HomeViewController
           🔴 APLViewController
           🔴 DetailPictureViewController
         ⊟ 🔴 UITableViewController
              🔴 MainPictureViewController
           🟢 ViewController
        🟢 AppDelegate
     🔴 SRSender
     🟢 KeychainWrapper
   ⊟ 🔴 RNCryptor
        🔴 RNEncryptor
        🔴 RNDecryptor
     🟢 AES
     🔴 RNCryptorEngine
     🟢 RSA
     🟢 SRSecurity
     🟢 SRErrorHandler
```

Figure 8.18: Objective-C Classes - list after login

Details for each class can be viewed and public methods can be invoked. The details for the instance of SRSecurity are shown in figure 8.19 on page 81

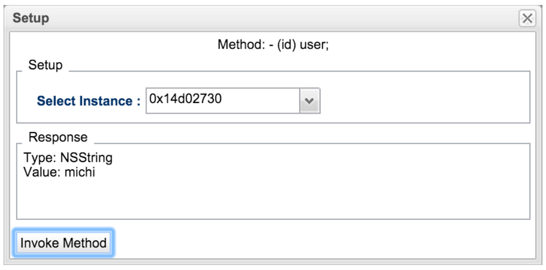


Selected Method: - (id) user;

```
//Classtype: App-Class
//Instances: 0x14d02730
@Interface SRSecurity : NSObject {
//Instance Variables

    SRErrorHandler* srErrorHandler;
    bool clean;
    NSMutableArray* _capturedImages;
    NSString* _user;
    NSString* _password;
    KeychainWrapper* _kcw;
    RSA* _rsa;
    AES* _aes;
}
//Properties
@property (retain,nonatomic) NSMutableArray* capturedImages; // = _capturedImages
@property (retain,nonatomic) NSString* user; // = _user
@property (retain,nonatomic) NSString* password; // = _password
@property (readonly,nonatomic) KeychainWrapper* kcw; // = _kcw
@property (readonly,nonatomic) RSA* rsa; // = _rsa
@property (readonly,nonatomic) AES* aes; // = _aes
//Methods

○ - (id) initWithUserAndPassword:(id)arg0 password:(id)arg1;
○ - (bool) isPasswordValid;
○ - (bool) login:(id*)arg0;
○ - (bool) userExists:(id*)arg0;
○ - (bool) initializeSetup:(id*)arg0;
```

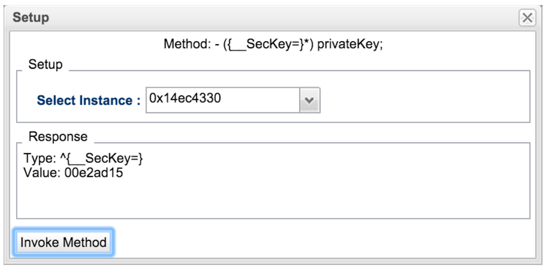
Figure 8.19: Objective-C Classes - SRSecurity details

Information like the user name (see figure 8.20a on page 82) or the private key (see figure 8.20b on page 82) are available after the login and can be retrieved by invoking the appropriate methods. The password is cleared from memory right after the password based key derivation and is therefore not available as shown in figure 8.20c on page 82.



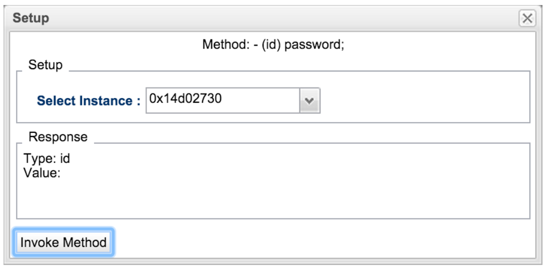
Method: - (id) user;
Setup
Select Instance : 0x14d02730
Response
Type: NSString
Value: michi

(a) Objective-C Classes - SRSecurity user



Method: - ({__SecKey=}*) privateKey;
Setup
Select Instance : 0x14ec4330
Response
Type: ^{__SecKey=}
Value: 00e2ad15

(b) Objective-C Classes - RSA privatekey



Method: - (id) password;
Setup
Select Instance : 0x14d02730
Response
Type: id
Value:

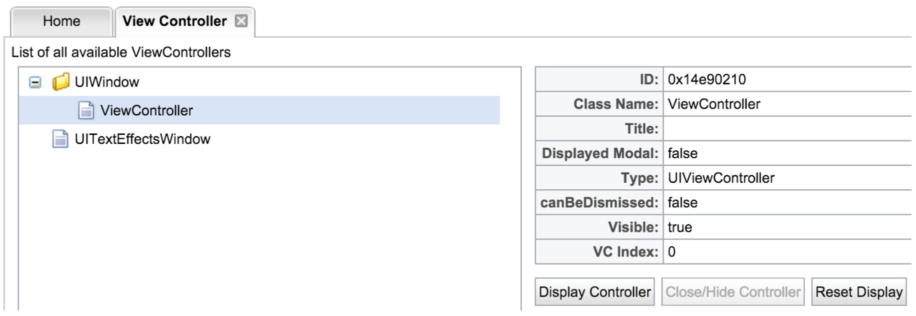
(c) Objective-C Classes - SRSecurity password

Figure 8.20: Objective-C Classes - invoked methods

After a successful logout the “Objective-C Classes” list looks the same like before the login (see figure 8.17 on page 80). Throughout the logout process the keys, the user name, the password and the captured images, are manually overwritten in memory, to make sure that this sensitive information is unrecoverable, before they are set to NULL and the objects are removed by the operating system.

View Controller:

This part of the Snoop-it application lists the available view controllers. Before the login, only the main “ViewController” is available and no other view controller can be instantiated in order to skip the login process and bypass the authentication mechanism (see figure 8.21 on page 82).



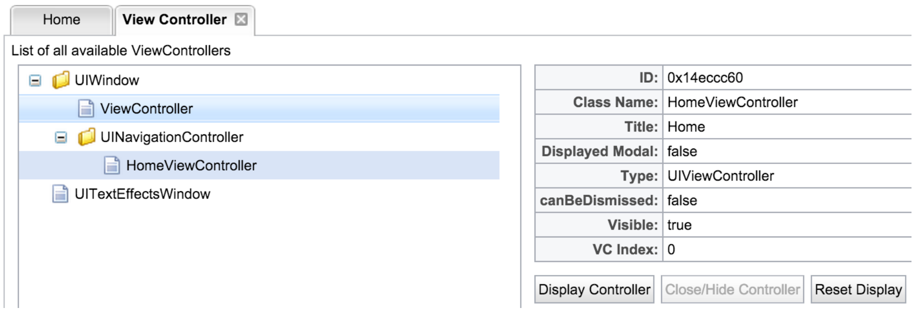
List of all available ViewControllers

UIWindow
ViewController
UITextEffectsWindow

ID:	0x14e90210
Class Name:	ViewController
Title:	
Displayed Modal:	false
Type:	UIViewController
canBeDismissed:	false
Visible:	true
VC Index:	0

Figure 8.21: View Controller - list before login

After the login and depending on the current view chosen in the prototype application, different view controllers can be displayed or the display can be reset. The figure 8.22 on page 83 shows the available view controllers when the home screen is displayed and the figure 8.23 on page 83 shows the available view controllers when a single picture view is active.

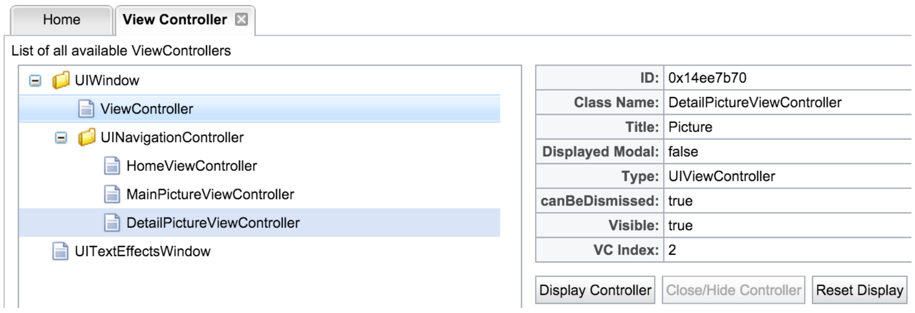


List of all available ViewControllers

- UIWindow
  - ViewController
  - UINavigationController
    - HomeViewController
  - UITextEffectsWindow

ID:	0x14eccc60
Class Name:	HomeViewController
Title:	Home
Displayed Modal:	false
Type:	UIViewController
canBeDismissed:	false
Visible:	true
VC Index:	0

Figure 8.22: View Controller - HomeViewController



List of all available ViewControllers

- UIWindow
  - ViewController
  - UINavigationController
    - HomeViewController
    - MainPictureViewController
    - DetailPictureViewController
  - UITextEffectsWindow

ID:	0x14ee7b70
Class Name:	DetailPictureViewController
Title:	Picture
Displayed Modal:	false
Type:	UIViewController
canBeDismissed:	true
Visible:	true
VC Index:	2

Figure 8.23: View Controller - DetailPictureViewController

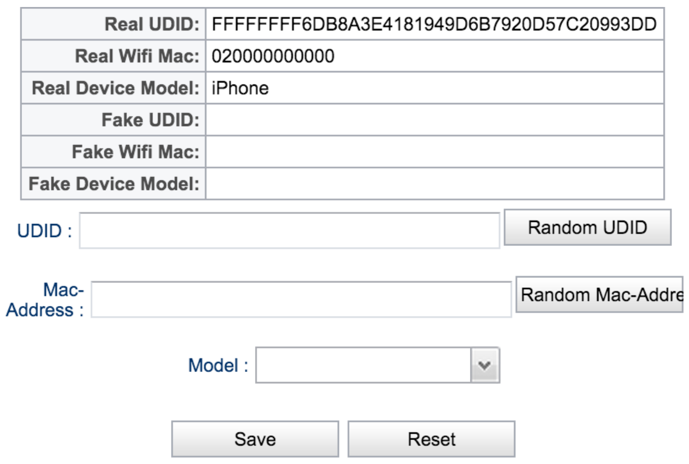
URL Schemes:

Within the prototype application no URL Schemes are used.

Runtime Manipulation:

Hardware Identifier:

This feature displays the device's Unique Device Identifier (UDID) and it's Media Access Control (MAC) address. It also enables the user to set a fake UDID, a fake MAC address and a fake device model (see figure 8.24 on page 84). These features are not applicable in the used test scenarios.

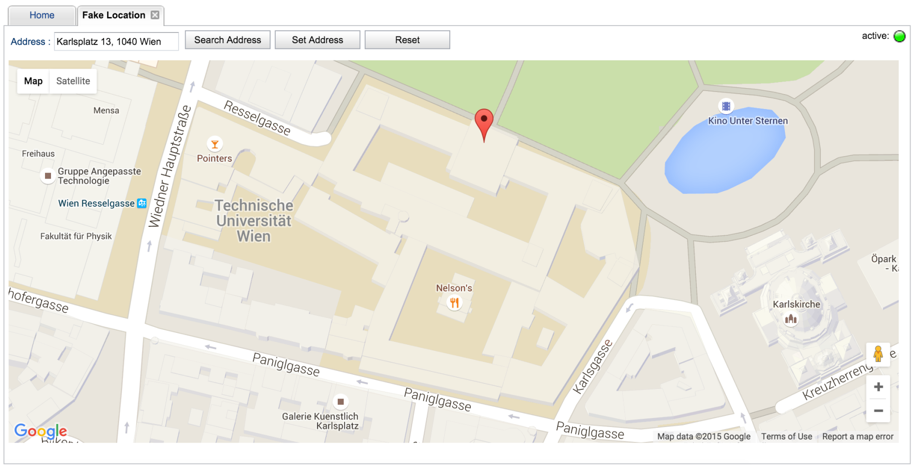


Real UDID:	FFFFFFFF6DB8A3E4181949D6B7920D57C20993DD
Real Wifi Mac:	020000000000
Real Device Model:	iPhone
Fake UDID:	
Fake Wifi Mac:	
Fake Device Model:	

Figure 8.24: Hardware Identifier - overview

Fake Location:

With the Snoop-it application it is also possible to set a fake location (see figure 8.25 on page 84). Since the prototype application does not use any location services, this feature is negligible.



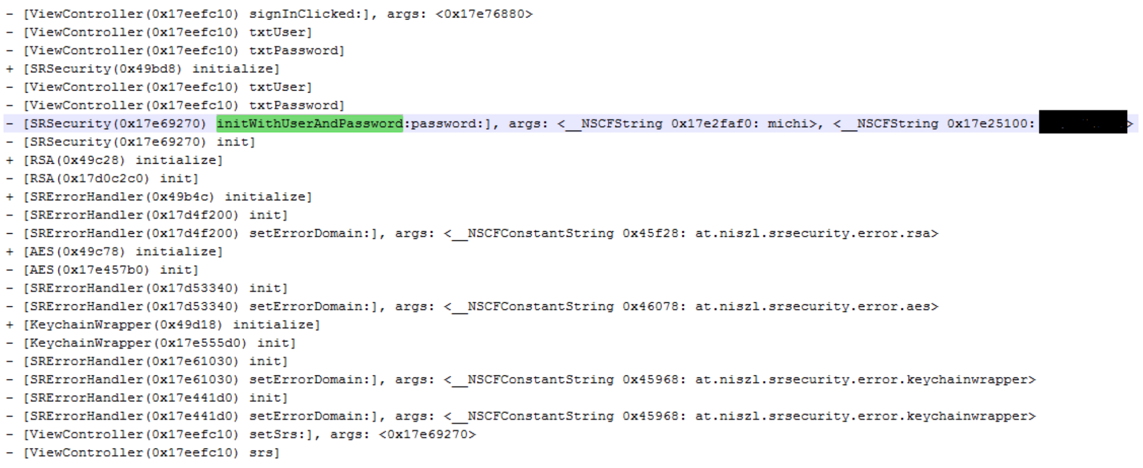
Address : Karlsplatz 13, 1040 Wien

Figure 8.25: Fake Location - overview

Method Tracing:

The last feature, "Method Tracing", allows the tracing of the called methods due to the root privileges of the Snoop-it application. The following section shows parts of the traced methods invoked during a successful login process.

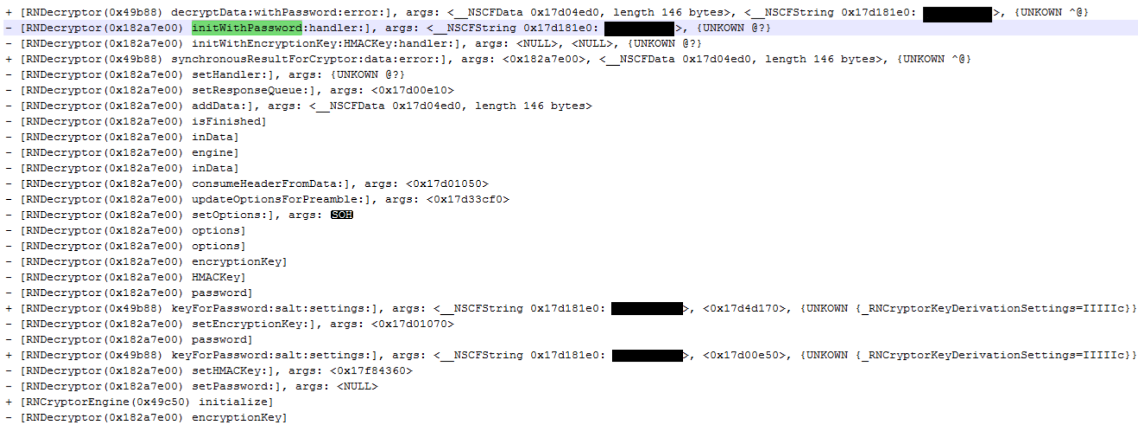
The following figure (see figure 8.26 on page 85) shows the initialization of the SRSecurity object including the username and the plain text password. After that the helper classes RSA, AES, KeychainWrapper and SRErrorHandler are initialized.



```
- [ViewController(0x17eefc10) signInClicked:], args: <0x17e76880>
- [ViewController(0x17eefc10) txtUser]
- [ViewController(0x17eefc10) txtPassword]
+ [SRSecurity(0x49bd8) initialize]
- [ViewController(0x17eefc10) txtUser]
- [ViewController(0x17eefc10) txtPassword]
- [SRSecurity(0x17e69270) initWithUserAndPassword:password:], args: <__NSCFString 0x17e2faf0: michi>, <__NSCFString 0x17e25100: [redacted]>
- [SRSecurity(0x17e69270) init]
+ [RSA(0x49c28) initialize]
- [RSA(0x17d0c2c0) init]
+ [SRErrorHandler(0x49b4c) initialize]
- [SRErrorHandler(0x17d4f200) init]
- [SRErrorHandler(0x17d4f200) setErrorDomain:], args: <__NSCFConstantString 0x45f28: at.niszl.srsecurity.error.rsa>
+ [AES(0x49c78) initialize]
- [AES(0x17e457b0) init]
- [SRErrorHandler(0x17d53340) init]
- [SRErrorHandler(0x17d53340) setErrorDomain:], args: <__NSCFConstantString 0x46078: at.niszl.srsecurity.error.aes>
+ [KeychainWrapper(0x49d18) initialize]
- [KeychainWrapper(0x17e555d0) init]
- [SRErrorHandler(0x17e61030) init]
- [SRErrorHandler(0x17e61030) setErrorDomain:], args: <__NSCFConstantString 0x45968: at.niszl.srsecurity.error.keychainwrapper>
- [SRErrorHandler(0x17e441d0) init]
- [SRErrorHandler(0x17e441d0) setErrorDomain:], args: <__NSCFConstantString 0x45968: at.niszl.srsecurity.error.keychainwrapper>
- [ViewController(0x17eefc10) setSrs:], args: <0x17e69270>
- [ViewController(0x17eefc10) srs]
```

Figure 8.26: Method Tracing - sign in

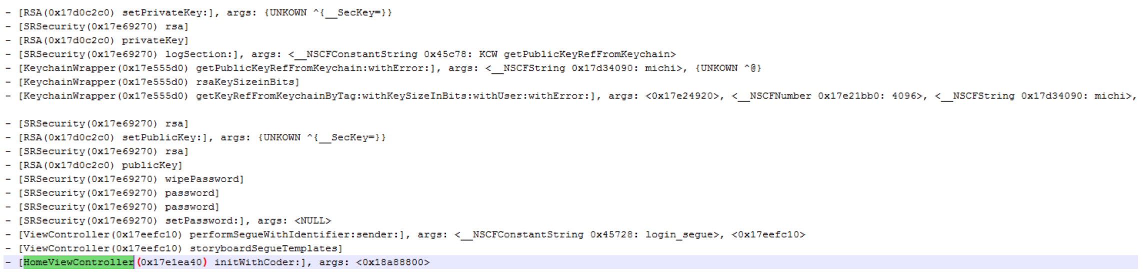
Also the password based key derivation and the decryption process are visible in the log file (see figure 8.27 on page 85).



```
+ [RNDecryptor(0x49b88) decryptData:withPassword:error:], args: <__NSCFData 0x17d04ed0, length 146 bytes>, <__NSCFString 0x17d181e0: [redacted]>, {UNKOWN ^@}
- [RNDecryptor(0x182a7e00) initWithPassword:handler:], args: <__NSCFString 0x17d181e0: [redacted]>, {UNKOWN @?}
- [RNDecryptor(0x182a7e00) initWithEncryptionKey:HMACKey:handler:], args: <NULL>, <NULL>, {UNKOWN @?}
+ [RNDecryptor(0x49b88) synchronousResultForCryptor:data:error:], args: <0x182a7e00>, <__NSCFData 0x17d04ed0, length 146 bytes>, {UNKOWN ^@}
- [RNDecryptor(0x182a7e00) setHandler:], args: {UNKOWN @?}
- [RNDecryptor(0x182a7e00) setResponseQueue:], args: <0x17d00e10>
- [RNDecryptor(0x182a7e00) addData:], args: <__NSCFData 0x17d04ed0, length 146 bytes>
- [RNDecryptor(0x182a7e00) isFinished]
- [RNDecryptor(0x182a7e00) inData]
- [RNDecryptor(0x182a7e00) engine]
- [RNDecryptor(0x182a7e00) inData]
- [RNDecryptor(0x182a7e00) consumeHeaderFromData:], args: <0x17d01050>
- [RNDecryptor(0x182a7e00) updateOptionsForPreamble:], args: <0x17d33cf0>
- [RNDecryptor(0x182a7e00) setOptions:], args: SOH
- [RNDecryptor(0x182a7e00) options]
- [RNDecryptor(0x182a7e00) options]
- [RNDecryptor(0x182a7e00) encryptionKey]
- [RNDecryptor(0x182a7e00) HMACKey]
- [RNDecryptor(0x182a7e00) password]
+ [RNDecryptor(0x49b88) keyForPassword:salt:settings:], args: <__NSCFString 0x17d181e0: [redacted]>, <0x17d4d170>, {UNKOWN {_RNCryptorKeyDerivationSettings=IIIIc}}
- [RNDecryptor(0x182a7e00) setEncryptionKey:], args: <0x17d01070>
- [RNDecryptor(0x182a7e00) password]
+ [RNDecryptor(0x49b88) keyForPassword:salt:settings:], args: <__NSCFString 0x17d181e0: [redacted]>, <0x17d00e50>, {UNKOWN {_RNCryptorKeyDerivationSettings=IIIIc}}
- [RNDecryptor(0x182a7e00) setHMACKey:], args: <0x17f84360>
- [RNDecryptor(0x182a7e00) setPassword:], args: <NULL>
+ [RNCryptorEngine(0x49c50) initialize]
- [RNDecryptor(0x182a7e00) encryptionKey]
```

Figure 8.27: Method Tracing - decryption process

After decrypting and setting the *MHM-key* and the *CPR-key*, the *CPUB-key* is fetched from the keychain, the password is wiped from memory and a segue to the HomeViewController is initiated (see figure 8.28 on page 85).



```
- [RSA(0x17d0c2c0) setPrivateKey:], args: {UNKOWN ^{__SecKey=}}
- [SRSecurity(0x17e69270) rsa]
- [RSA(0x17d0c2c0) privateKey]
- [SRSecurity(0x17e69270) logSection:], args: <__NSCFConstantString 0x45c78: KCW getPublicKeyRefFromKeychain>
- [KeychainWrapper(0x17e555d0) getPublicKeyRefFromKeychain:withError:], args: <__NSCFString 0x17d34090: michi>, {UNKOWN ^@}
- [KeychainWrapper(0x17e555d0) rsaKeySizeinBits]
- [KeychainWrapper(0x17e555d0) getKeyRefFromKeychainByTag:withKeySizeInBits:withUser:withError:], args: <0x17e24920>, <__NSCFNumber 0x17e21bb0: 4096>, <__NSCFString 0x17d34090: michi>,

- [SRSecurity(0x17e69270) rsa]
- [RSA(0x17d0c2c0) setPublicKey:], args: {UNKOWN ^{__SecKey=}}
- [SRSecurity(0x17e69270) rsa]
- [RSA(0x17d0c2c0) publicKey]
- [SRSecurity(0x17e69270) wipePassword]
- [SRSecurity(0x17e69270) password]
- [SRSecurity(0x17e69270) password]
- [SRSecurity(0x17e69270) setPassword:], args: <NULL>
- [ViewController(0x17eefc10) performSegueWithIdentifier:sender:], args: <__NSCFConstantString 0x45728: login_segue>, <0x17eefc10>
- [ViewController(0x17eefc10) storyboardSegueTemplates]
- [HomeViewController(0x17e1ea40) initWithCoder:], args: <0x18a88800>
```

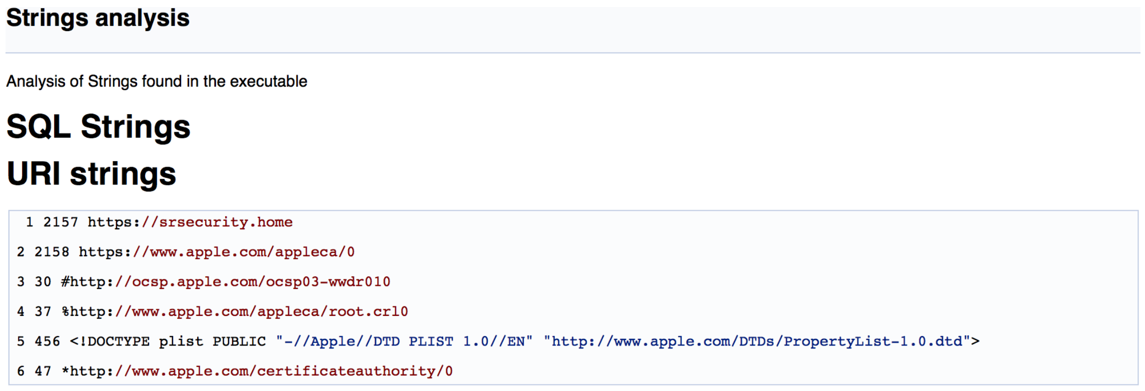
Figure 8.28: Method Tracing - segue to HomeViewController

iNalyzer

iNalyzer is a mobile penetration testing framework, which can be installed on a jailbroken device. It allows to analyze a targeted application's class structure including diagrammed representation of the class hierarchy. Further features are the examination of plist-, database- and user-files and the detailed graphical and textual representation of view controller dependencies. The following section uses the iNalyzer's dashboard to investigate the prototype application.

Strings:

The following figure (see figure 8.29 on page 86) shows the extracted Structured Query Language (SQL)- and Uniform Resource Identifier (URI)-strings.



Strings analysis

Analysis of Strings found in the executable

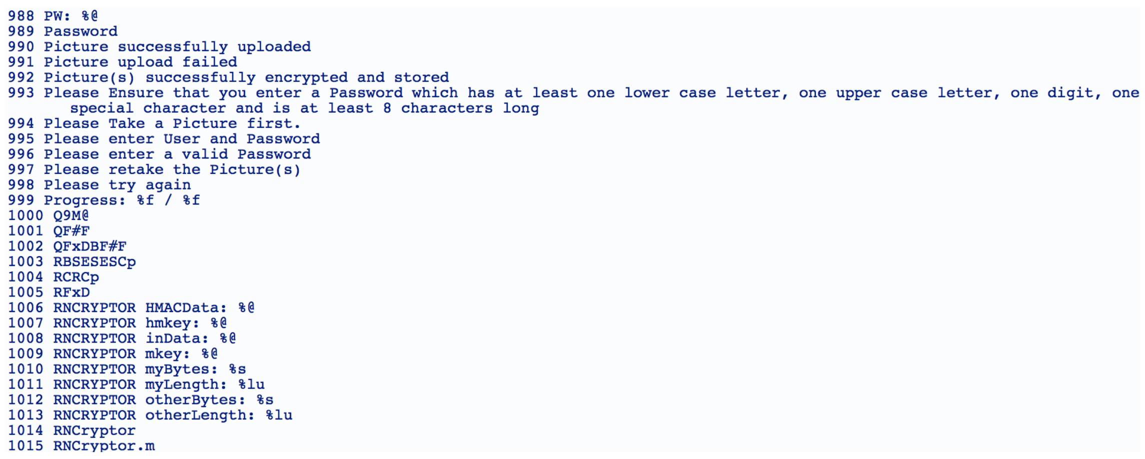
SQL Strings

URI strings

```
2157 https://srsecurity.home
2158 https://www.apple.com/appleca/0
30 #http://ocsp.apple.com/ocsp03-wwdr010
37 %http://www.apple.com/appleca/root.crl0
456 <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
47 *http://www.apple.com/certificateauthority/0
```

Figure 8.29: Strings - SQL- and URI-strings

The only useful information that can be extracted, is the URI of the backend, *https://srsecurity.home*. Another string functionality of the iNalyzer application is "Embedded Strings", which extracts all strings used in the scanned application. This feature can be useful to extract hard coded information, like keys or passwords. The following figure, figure 8.30 on page 86, gives a short insight on the extracted strings.

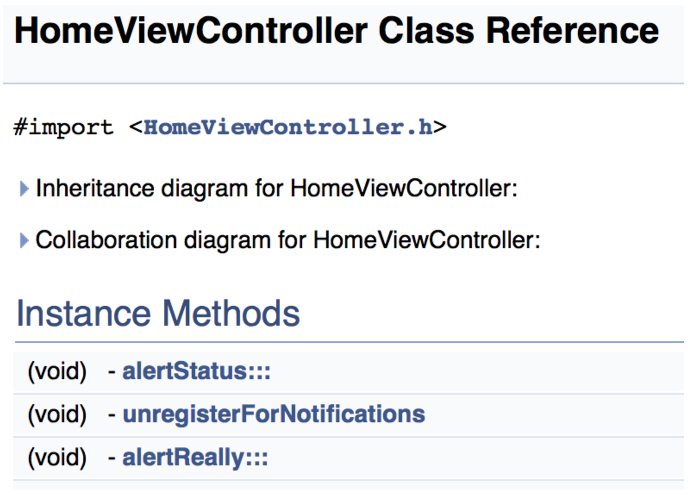


```
988 PW: %@
989 Password
990 Picture successfully uploaded
991 Picture upload failed
992 Picture(s) successfully encrypted and stored
993 Please Ensure that you enter a Password which has at least one lower case letter, one upper case letter, one digit, one
      special character and is at least 8 characters long
994 Please Take a Picture first.
995 Please enter User and Password
996 Please enter a valid Password
997 Please retake the Picture(s)
998 Please try again
999 Progress: %f / %f
1000 Q9M@
1001 QF#F
1002 QFxDBF#F
1003 RBSESESCp
1004 RCRCp
1005 RFxD
1006 RNCRYPTOR HMACData: %@
1007 RNCRYPTOR hmkey: %@
1008 RNCRYPTOR inData: %@
1009 RNCRYPTOR mkey: %@
1010 RNCRYPTOR myBytes: %s
1011 RNCRYPTOR myLength: %lu
1012 RNCRYPTOR otherBytes: %s
1013 RNCRYPTOR otherLength: %lu
1014 RNCryptor
1015 RNCryptor.m
```

Figure 8.30: Strings - embedded strings

ViewControllers:

Similar to the Snoop-it feature, this iNalyzer feature parses information of the view controller from the header files. It shows “Instance Methods” (see figure 8.31 on page 87), “Protected Attributes” (see figure 8.32a on page 87), “Properties” (see figure 8.32b on page 87) and creates a short documentation for each of these characteristics (e.g. “Properties”, see figure 8.33 on page 87).


HomeViewController Class Reference

```
#import <HomeViewController.h>
```

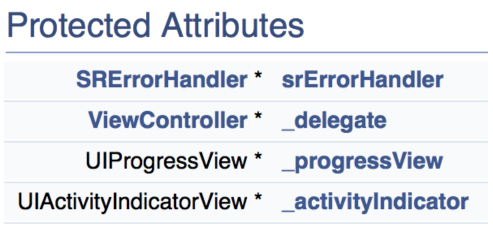
Inheritance diagram for HomeViewController:

Collaboration diagram for HomeViewController:

Instance Methods

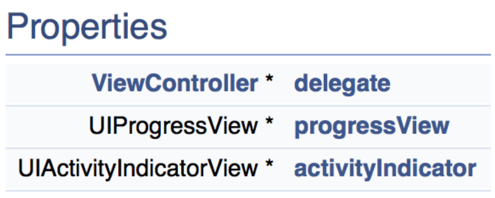
(void)	- alertStatus:::
(void)	- unregisterForNotifications
(void)	- alertReally:::

Figure 8.31: ViewControllers - overview and methods


Protected Attributes

SRErrorHandler *	srErrorHandler
ViewController *	_delegate
UIProgressView *	_progressView
UIActivityIndicatorView *	_activityIndicator

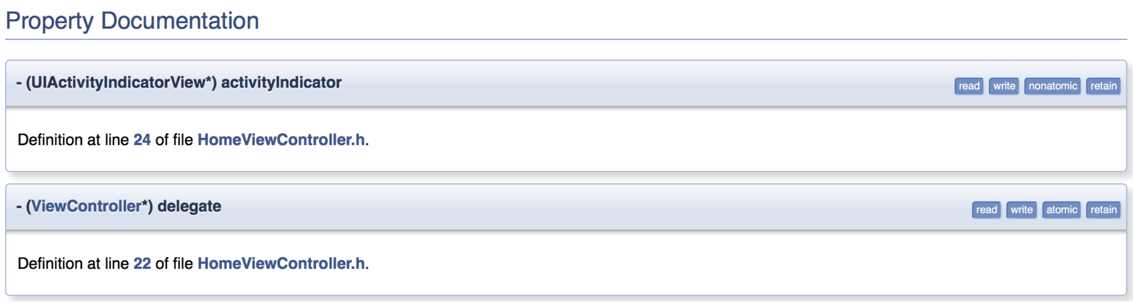
(a) ViewControllers - protected attributes


Properties

ViewController *	delegate
UIProgressView *	progressView
UIActivityIndicatorView *	activityIndicator

(b) ViewControllers - properties

Figure 8.32: ViewControllers - characteristics


Property Documentation

- (UIActivityIndicatorView*) activityIndicator — read write nonatomic retain

Definition at line 24 of file HomeViewController.h.

- (ViewController*) delegate — read write atomic retain

Definition at line 22 of file HomeViewController.h.

Figure 8.33: ViewControllers - documentation properties

Another very useful part of the “ViewControllers” feature is the collaboration diagram, which is available for every class extracted by iNalyzer. The figure 8.34 on page 88 displays the collaboration diagram for “HomeViewController”.

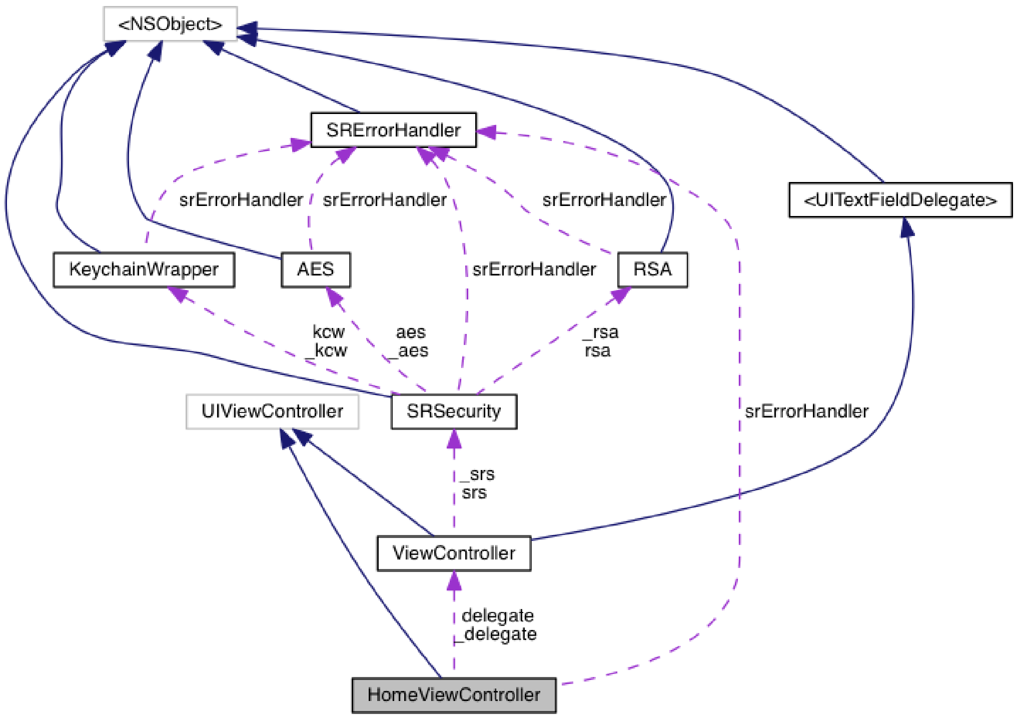
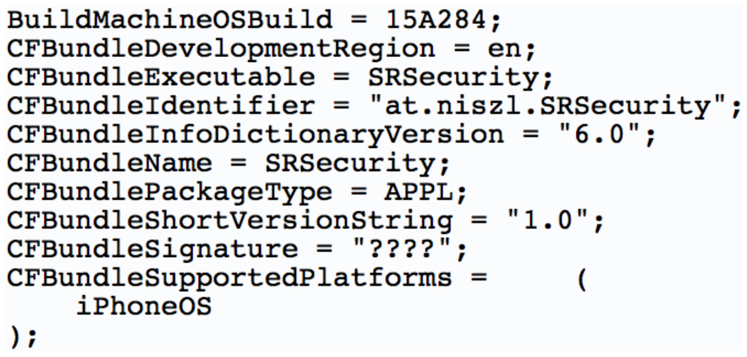


Figure 8.34: ViewControllers - collaboration diagram HomeViewController

Plist Files:

iNalyzer extracts all plist files from the scanned application, including the “Entitlements”. The three extracted “Info.plist” (see [4]) files hold information about the application’s configuration, like the bundle executable, the identifier or the version string. This allows to get an overview of the application bundle’s properties (see figure 8.35 on page 88).



```
BuildMachineOSBuild = 15A284;
CFBundleDevelopmentRegion = en;
CFBundleExecutable = SRSecurity;
CFBundleIdentifier = "at.niszl.SRSecurity";
CFBundleInfoDictionaryVersion = "6.0";
CFBundleName = SRSecurity;
CFBundlePackageType = APPL;
CFBundleShortVersionString = "1.0";
CFBundleSignature = "????";
CFBundleSupportedPlatforms =      (
    iPhoneOS
);
```

Figure 8.35: Plist Files - Info.plist

Another extracted plist file is the “CodeResources” (see [5]) file (see figure 8.36 on page 89), which stores the hashes for the files included in the bundle. The application will not install itself, if the hash values, created at build time, do not match the files hash values, currently installed in the application bundle.

```
"Base.lproj/Main~iphone.storyboardc/Info-8.0+.plist" =         {
    hash = <590ef04a d20258dd 71913736 de2ddaa0 04ced426>;
    optional = 1;
};
"Base.lproj/Main~iphone.storyboardc/Info.plist" =         {
    hash = <46011814 0465f41b 412f1a0b d9253bc6 f6ba9dc3>;
    optional = 1;
};
"Info.plist" = <8dc14743 81b8a67c 13d2a9db e8127a17 6c787cf1>;
"OverlayView.nib" = <93e93156 8e39d3b3 cf404ca9 a688321a b0200e97>;
PkgInfo = <9f9eea0c fe2d65f2 c3d6b092 e375b407 82d08f31>;
"embedded.mobileprovision" = <ed54b08b 9cc8459a 474f8066 540b61ed 578ea100>;
"srsecurity.home.der" = <d8ea5616 11373cd2 6eb75924 1212646f cbcaa1e3>;
```

Figure 8.36: Plist Files - CodeResources.plist

The “Entitlements” (see [3]) file retains security permissions for the iOS application. The figure, figure 8.37 on page 89, shows, for example, the stored keychain access groups and the default Data Protection Class.

```
<plist version="1.0">
<dict>
 <key>application-identifier</key>
 <string>X7289A9QV7.at.niszl.SRSecurity</string>
 <key>com.apple.developer.default-data-protection</key>
 <string>NSFileProtectionComplete</string>
 <key>com.apple.developer.team-identifier</key>
 <string>X7289A9QV7</string>
 <key>get-task-allow</key>
 <true/>
 <key>keychain-access-groups</key>
 <array>
  <string>X7289A9QV7.at.niszl.SRSecurity</string>
 </array>
</dict>
</plist>
```

Figure 8.37: Plist Files - Entitlements.plist

Keychain Data:

The “Keychain Data” feature was not able to extract any entries. Keychain related information is discussed in chapter 8.3.4 on page 77.

Database Files:

Since no databases are used in the prototype application, this feature can not extract any information.

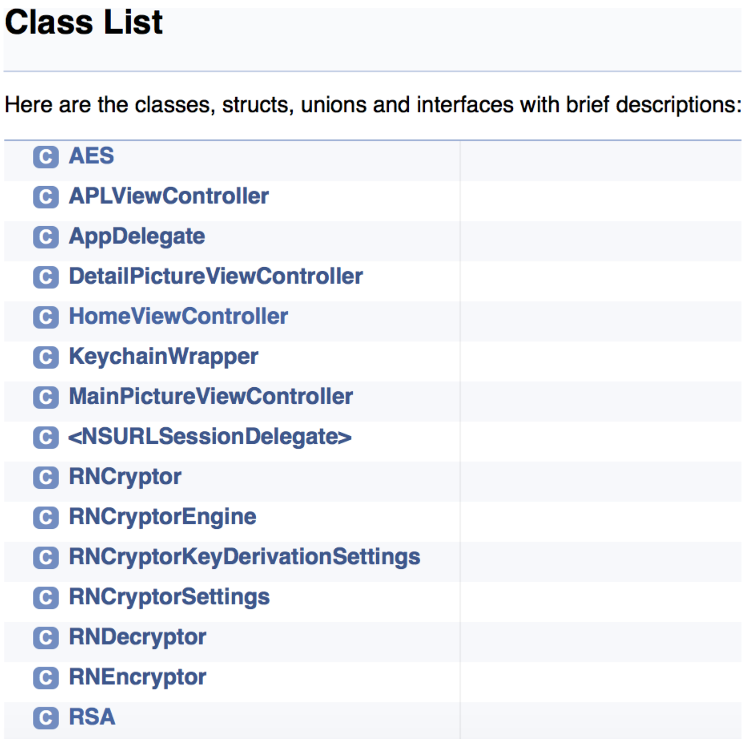
Protection Class:

The files stored by the application and their Data Protection Classes are reviewed in chapter 8.2.4 on page 73.

Classes:

The “Classes” component of iNalyzer is separated into four sections, “Class List”, “Class Index”, “Class Hierarchy” and “Class Members”.

“Class List” provides an overview of the extracted classes (see figure 8.38 on page 90).



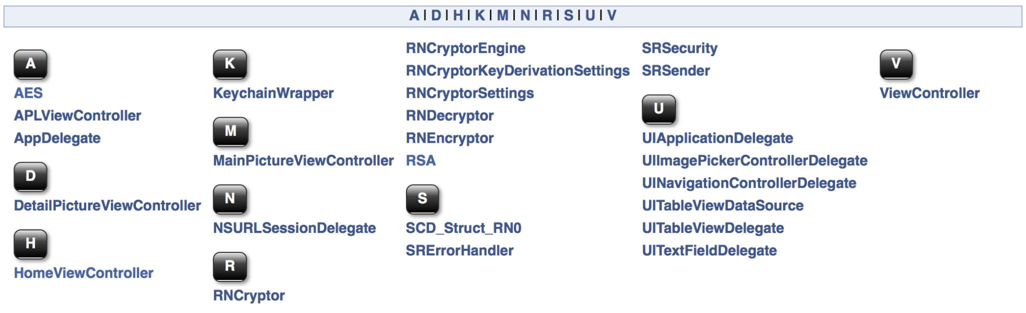
Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

- C AES
- C APLViewController
- C AppDelegate
- C DetailPictureViewController
- C HomeViewController
- C KeychainWrapper
- C MainPictureViewController
- C <NSURLSessionDelegate>
- C RNCryptor
- C RNCryptorEngine
- C RNCryptorKeyDerivationSettings
- C RNCryptorSettings
- C RNDecryptor
- C RNEncryptor
- C RSA

Figure 8.38: Classes - list

“Class Index” shows the classes in alphabetical order (see figure 8.39 on page 90).



A | D | H | K | M | N | R | S | U | V

A
AES
APLViewController
AppDelegate

D
DetailPictureViewController

H
HomeViewController

K
KeychainWrapper

M
MainPictureViewController

N
NSURLSessionDelegate

R
RNCryptor
RNCryptorEngine
RNCryptorKeyDerivationSettings
RNCryptorSettings
RNDecryptor
RNEncryptor
RSA

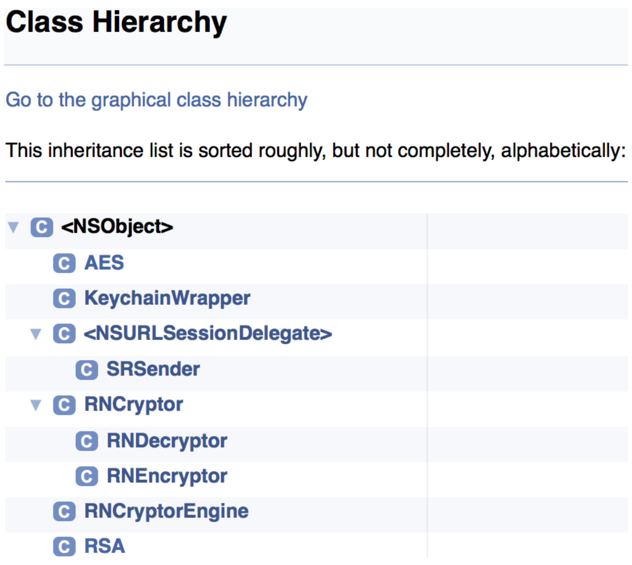
S
SCD_Struct_RN0
SRErrorHandler
SRSecurity
SRSender

U
UIApplicationDelegate
UIImagePickerControllerDelegate
UINavigationControllerDelegate
UITableViewDataSource
UITableViewDelegate
UITextFieldDelegate

V
ViewController

Figure 8.39: Classes - index

“Class Hierarchy” also holds the inheritance structure of the classes (see figure 8.40 on page 91).



Class Hierarchy

Go to the graphical class hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- <NSObject>
  - AES
  - KeychainWrapper
  - <NSURLSessionDelegate>
    - SRSender
  - RNCryptor
    - RNDecryptor
    - RNEncryptor
  - RNCryptorEngine
  - RSA

Figure 8.40: Classes - hierarchy

It is also possible to extract a global hierarchy diagram (see figure 8.41 on page 91).

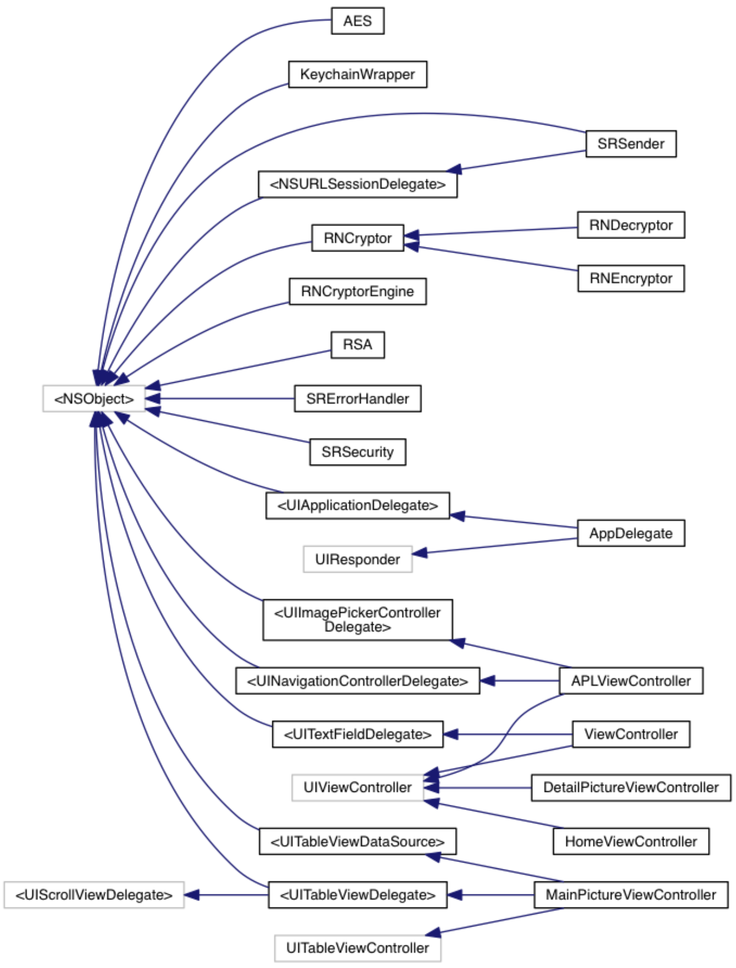
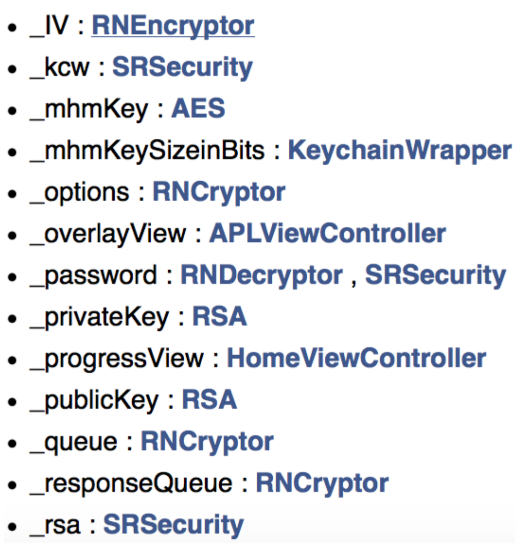


Figure 8.41: Classes - inheritance diagram

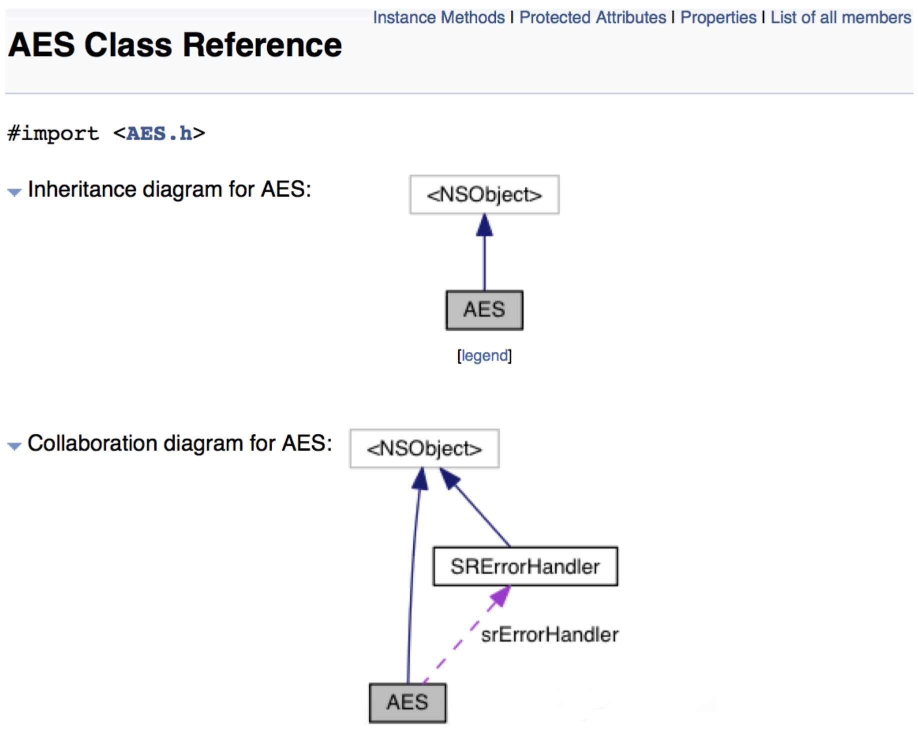
“Class Members” gives a list of all class members and associates them with their class (see figure 8.42 on page 92).



- _IV : RNEncryptor
- _kcw : SRSecurity
- _mhmKey : AES
- _mhmKeySizeinBits : KeychainWrapper
- _options : RNCryptor
- _overlayView : APLViewController
- _password : RNDecryptor , SRSecurity
- _privateKey : RSA
- _progressView : HomeViewController
- _publicKey : RSA
- _queue : RNCryptor
- _responseQueue : RNCryptor
- _rsa : SRSecurity

Figure 8.42: Classes - members

Each class, just like every view controller, provides additional information like “Instance Methods”, “Protected Attributes”, “Properties” and a short documentation from the parsed header files (see figure 8.43 on page 92).



Instance Methods | Protected Attributes | Properties | List of all members

AES Class Reference

```
#import <AES.h>
```

Inheritance diagram for AES:

Collaboration diagram for AES:

Figure 8.43: Classes - class reference

8.3.5 Results

The runtime analysis showed the vast threat a jailbroken device poses to an application's security. Snoop-it allowed to record actions performed on the keychain and also was able to read these keychain entries in clear text. Only the additionally implemented encryption layer for the *MHM-key* and the *CPR-key* obstructs the Snoop-it user from reading the plain key data.

The Common Crypto API, which is part of the iOS SDK, is used to encrypt and decrypt keys and images within the prototype application. The security assessment tool was able to record every Common Crypto API call and furthermore reveal the unencrypted *MHM*- and *CPR-key*.

Snoop-it also created a list of objective C classes including detailed information, parsed from the projects header files, and displayed whether an object instance is currently available in memory. For available object instances it was possible to invoke public methods and retrieve information like the user name or the plain text private key (*CPR-key*).

Both tools, Snoop-it and iNalyzer, allowed to gather extensive information about the applications class structure. A full class hierarchy was extracted including the instance methods, protected attributes and properties. For every class, including the view controllers, iNalyzer was able to generate an inheritance- and a collaboration-diagram. This assemblage of information gives valuable support during a possible reverse engineering process.

iNalyzer also gathered URI- and SQL-string information and was able to extract the backend URI from the application package. The extracted plist-files provided information about the app's configuration and security permissions, like the default keychain access group and the default Data Protection Class.

Snoop-it's tracing feature recorded all method calls, including the parameters, during the prototype applications runtime. This enabled the gathering of crucial information like plain text passwords and keys used during the usage of the app.

9 Conclusion

This chapter discusses the results of the analysis in reference to the hypothesis proposed in section 1.2 on page 1. The ongoing improvements in iOS, based on the changes in the current major release of iOS 9, are disclosed. At last, findings and consequences are discussed and future trends for mobile security and the privacy of data, in general, are highlighted.

9.1 Results

As defined in chapter 1.2 on page 1 the hypothesis states that *"...an iOS application's data can be secured even if the operating system's native encryption mechanisms are broken or bypassed."*. The results of the analysis carried out in chapter 8, beginning on page 61, show that the majority of the operating system's native security mechanisms can be bypassed on a jailbroken device. Only the Data Protection mechanism could not be annulled, even with elevated privileges (see section 8.2.4 on page 75). After unlocking the device, by entering a valid passcode, the Data Protection mechanism was disabled and granted access to the data. But even on an unlocked and rooted device, the data was still encrypted, due to the additionally implemented security layer and therefore cryptographically inaccessible for a potential attacker. Although the raised privileges allowed to annul the implemented certificate pinning mechanism, the transmitted data was still inaccessible due to the lack of the corresponding keys (see figure 8.6 on page 70). The strength of the additional layer's security mainly depends on the user's password, which is the origin of the whole key architecture hierarchy. If a potential attacker cannot gain access to the password, it is cryptographically impossible to derive the keys used for the encryption of the master key set. Nevertheless the runtime analysis showed that even method calls, like the Password Based Key Derivation Function can be traced on a jailbroken device and therefore the password can be read from memory (see figure 8.26 on page 85). Snoop-it furthermore recorded every action performed on the keychain, including the stored data. Due to the design of the additionally implemented layer, the *MHM-key* and *CPR-key* are only stored in an encrypted form (see figure 8.11 on page 78 and figure 8.12 on page 78). Unfortunately, the encryption and decryption of the keys, using the Common Crypto API, are also recorded and display all keys in clear text (see figure 8.16 on page 79). If the attacker is able to retrieve any of the plain key or password information, the additional layer of security is in jeopardy.

The second hypothesis declares that *"...only iOS crypto API functionalities are needed to secure the application data in a state of the art manner."* (1.2 on page 1). The prototype application setup is described in section 7.3 on page 55 and only utilizes the built-in CommonCrypto API (see [6]) throughout the whole encryption and decryption mechanisms. As mentioned in section 8.1.4 on page 67, the tested iOS 8.4 SDK does not provide AES with Galois/Counter Mode (GCM), which is the recommended state of the art AES mode. With the introduction of iOS 9 (see [12]), Apple has enabled AES with Galois/Counter Mode (GCM), which couldn't be tested due to the restrictions of the test setup. A secure file wiping mechanism, like the one described by Zdziarski in his book "Hacking and Securing iOS Applications" (see [76]) and recommended by the US DoD (see page 274 of [76]), also is not available in the Software Development Kit (SDK) and needs to be implemented separately in order to hinder potential attackers to retrieve classified deleted data.

9.2 Current Improvements

Since 2006 and iOS 6, Apple validates the system's cryptographic modules for compliance with the US Federal Information Processing Standard (FIPS) 140-2 Level 1. The current crypto modules in iOS 9 are the same as in iOS 8 and are validated under "Apple iOS FIPS Cryptographic Modules v5.0" (see [14]). This validation guarantees the integrity of the iOS cryptographic services and the operations carried out using these modules [14].

Beginning with iOS 9 Apple extends their aspiration for certification and added Common Criteria Certification (CCC) (ISO-15408) and the Commercial Solution for Classified (CsfC) Program to their list. The Mobile Device Fundamental Protection Profile v2.0 (MDFFP2) and the VPN IPSecPP 1.4 Client Protection Profile (VPNIPSecPP1.4) are the first two modules certified under CCC. Apple is consecutively evaluating available and already certified Protection Profiles (PPs) and concentrates on the development of new key mobile security PPs [14]. To develop and maintain a secure mobile environment, Apple has partnered with governments worldwide and created best-practice guidelines and recommendations (see [14]).

The current major release, iOS 9, also comes with other security related new features.

The default passcode requires six digits, instead of four digits, as in previous versions. This generates 1 million different combinations instead of 10.000 and therefore impedes a brute force attack. iOS 9 also provides full native support for two-factor authentication [18].

Apple also switched from OpenSSL[1] to LibreSSL[2]. LibreSSL is a fork of OpenSSL which gained popularity after the OpenSSL's "Heartbleed"[3] issue was discovered [18].

Another major security improvement regarding the new iOS version is the App Transport Security (ATS). App Transport Security enforces the current best practice for secure connections. This means that servers have to support TLSv1.2, forward secrecy and provide a valid certificate signed using SHA-256, 2048 bit RSA, 256 bit elliptic curve keys, or better. These requirements are applied to iOS 9 applications by default. Requirements can be overwritten and connections fail if they are not met. CFNetwork also disallows SSLv3 by default in the current version [12].

Apple is an early adopter of "Certificate Transparency". "Certificate Transparency" is a framework created by Google which enables an easy approach to audit digital certificates. It ensures that a certain certificate is neither malicious nor forged [18].

The iOS 9 release patches over 100 security holes since iOS 8. This includes a patch for a bug that allowed the interception of TLS connections (CVE-2015-5824[4]). iOS 8 also allowed arbitrary code execution within a crafted font file (CVE-2015-5874[5]). Another recently patched security hole allowed an attacker to determine a RSA private key after the observation of many decryption or signing attempts. By exploiting the vulnerability with the CVE-ID CVE-2015-5876[6], an application was able to execute arbitrary code with system privileges.

[1] https://www.openssl.org/
[2] http://www.libressl.org
[3] http://heartbleed.com
[4] http://www.cvedetails.com/cve/CVE-2015-5824/ - Serkan Oezkan - (visited on 11/09/2015)
[5] http://www.cvedetails.com/cve/CVE-2015-5874/ - Serkan Oezkan - (visited on 11/09/2015)
[6] http://www.cvedetails.com/cve/CVE-2015-5876/ - Serkan Oezkan - (visited on 11/09/2015)

9.3 Discussion

As is determined in the Analysis chapter on page 61, unauthorized privilege escalation is the biggest threat to information stored on an iOS device. Current trends show that in the last three iOS releases the time span between an iOS release and the release of a jailbreak for this version has decreased. While the jailbreak release for iOS 7[7] took 95 days, it took 35 days for an iOS 8[8] jailbreak and even 28 days for iOS 9[9].

The past has shown that an upgrade to a major release iOS version does not only patch older vulnerabilities, but also potentially opens up new ones. The figure 9.1 on page 96 shows the reported vulnerabilities for the iOS platform by year. The number of vulnerabilities reported in 2015 are nearly three times as high as the numbers reported in 2014. It is not possible to state with certainty whether there were more vulnerabilities in the iOS versions reported in 2015 compared to 2014 or if the interest in iOS vulnerabilities drastically increased.

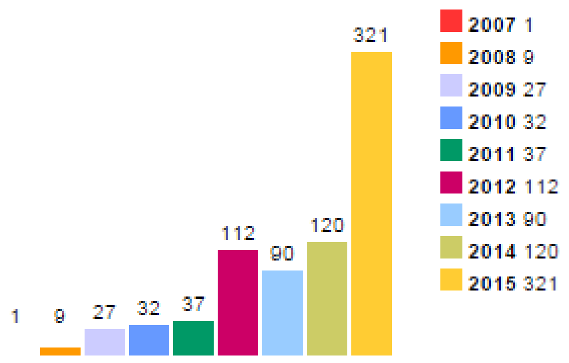


Figure 9.1: Vulnerabilities, by year [63]

The top 3 vulnerability types reported since 2007, are “Denial of Service”- “Execute Code”- and “Memory Corruption”-vulnerabilites [63], as is displayed in figure 9.2 on page 96.

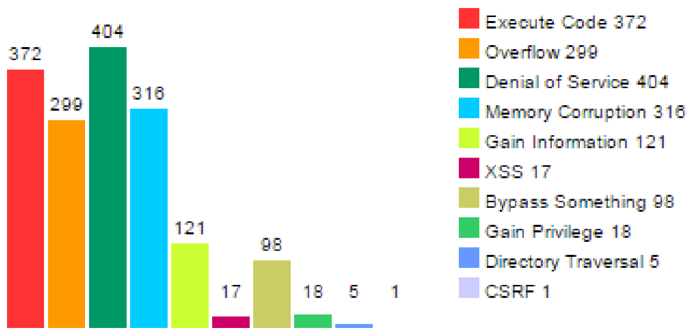


Figure 9.2: Vulnerabilities, by type [63]

[7] http://evasi0n.com/
[8] http://en.pangu.io/
[9] http://en.pangu.io/

There is a very lucrative business behind detecting vulnerabilities and especially zero-day exploits[10]. Companies like Zerodium[11] or its predecessor VUPEN[12] offer high rewards for undisclosed vulnerabilities. Customers of this kind of corporations range from major technology or finance firms as well as organizations on a governmental level [34].

Documents released under the Freedom of Information Act (FOIA) revealed a contract signed in 2012 between the National Security Agency (NSA) and VUPEN (see [2]) for a one year subscription to VUPEN's "binary analysis and exploits service". In November 2011, the German *Federal Office for Information Security*[13] signed a contract with VUPEN to annually acquire 50 "Credits", which could be traded to receive reports about security holes and vulnerabilities. This was part of the "Programm Threat Protection Program - Comprehensive Level" contract (see [59]) between the *Federal Office for Information Security* and VUPEN which was disclosed under the German *Freedom of Information Act*[14].

Zerodium hosted a zero-day bug bounty program in September 2015, putting a one million dollar bounty on a browser-based and untethered jailbreak for the latest Apple iOS 9. This kind of exploit would allow an attacker to remotely install an application with full privileges and intercept calls and messages or access sensitive data stored on the mobile device. The company announced that a team was able to remotely jailbreak an iOS9.1/9.2b device via a browser-based exploit [34].

A bounty program with a reward of one million dollars, which is much higher than what other companies normally pay out, emphasizes the value of this kind of vulnerabilities.

There is also an increased interest in bypassing encryption, especially on a governmental level, as a testimonial of the Federal Bureau of Investigation (FBI) director, James B. Comey, shows (see [27]). Comey criticises the new scale mainstream products, designed to give the users the sole control over their data and how criminals and terrorists might use this technology to their advantage. He also states that law enforcement is unable to recover this kind of data and that even providers, which are served with a warrant, can not provide the data due to the design of the technology. He describes how the current changes in technology hinder law enforcement to investigate critical leads on terrorists, which is among the highest priorities for the Department of Justice and the US government. Director Comey emphasizes that the FBI is not asking to expand the government's surveillance authority, but to ensure that there are lawful ways to obtain electronic evidence while promoting strong encryption to protect privacy [27].

In the wake of the terrorist attacks in Paris on September 13, 2015, government officials renewed their arguments against encryption. The Central Intelligence Agency (CIA) Director John Brennan stated that the current technological capabilities make it technically and legally difficult for intelligence and security services to gain the insight they need. He also criticises that policies and legal actions make the ability to find terrorist more challenging and he hopes that this is a wake-up call especially for certain areas of Europe [21].

According to Wired.com, encryption will be a key issue as part of the national security conversation in the 2016 presidential elections. The democratic candidates do not give much detail on where they stand regarding this topic. A standpoint against encryption could estrange the powerful Silicon Valley base whereas a pro encryption position could mean loosing votes of people mainly concerned of national security. Former Secretary of State Hillary Clinton represents a very uncertain standpoint regarding encryption. On the one hand, she said that the Cybersecurity Infor-

[10] An unknown security flaw, which hasn't been fixed yet.
[11] https://www.zerodium.com/
[12] http://www.vupen.com/
[13] *Bundesamt für Sicherheit in der Informationstechnik (BSI)*
[14] *Informationsfreiheitsgesetz*

mation Sharing Act (CISA) does not motivate technology companies enough to share information. On the other hand, she supported the USA Freedom Act that modified several provisions of the Patriot Act, like the NSA data collection program. Senator Bernard Sanders has a more forthright opinion on surveillance and encryption and demonstrated it by voting against the Patriot Act. The republican candidates Donald Trump, Jeb Bush, Marco Rubio, Carly Fiorina, and Chris Christie advocate for the need of governmental surveillance and are strictly against encryption [53].

The Information Technology Industry Council (ITI) serves as platform and unified voice of the major technology companies, including Google, Apple, IBM, Microsoft, Facebook, Twitter, Intel and many others. ITI's President and CEO Dean Garfield released a response regarding several statements of governmental representatives criticising encryption:

"Encryption is a security tool we rely on everyday to stop criminals from draining our bank accounts, to shield our cars and airplanes from being taken over by malicious hacks, and to otherwise preserve our security and safety. We deeply appreciate law enforcement's and the national security community's work to protect us, but weakening encryption or creating backdoors to encrypted devices and data for use by the good guys would actually create vulnerabilities to be exploited by the bad guys, which would almost certainly cause serious physical and financial harm across our society and our economy. Weakening security with the aim of advancing security simply does not make sense." [35]

Since the two largest mobile OS manufacturers, Google and Apple, are US based companies, the outcome of the encryption debate strongly affects the mobile OS sector and furthermore the mHealth industry. A backdoor for governmental organizations would not only obliterate the primal idea of encryption, but would also make the encryption mechanism more attack prone and therefore endanger sensitive data stored on a mobile device. This would increase the need for organizations to implement their own encryption algorithms. Especially groupings threatening the national security would find a way to implement their own encryption systems or would use non-US appliances, making one of the main arguments of encryption antagonists obsolete. A ruling opposing encryption would only threaten the end-user's privacy and would not contribute to national security.

Austrian laws regarding medical data (see 3.5 on page 18) demand protection of unauthorized access, unlawful destruction and improper use of medical data. The Austrian Health Telematics Law[15] furthermore requires the implementation of state of the art standards to provide confidentiality to the sensitive data in transit. It has to be discussed whether the mobile operating system's native encryption mechanisms provide an adequate level of security and if US governmental organizations can force the OS manufacturers by law to share the data stored on a mobile device. Laws are by definition a very rigid construct and are not designed to be adopted in a very frequent manner. Vulnerabilities in cryptographic algorithms or methods, on the other hand, are not a rarity. As is described on page 97, vulnerability detection is a very lucrative business and the computational power to crack cryptographic algorithms also is on the rise. A timely adoption of required standards is therefore a crucial factor to the security of medical data. The latest regulation on health telematics is from the year 2013 and includes permitted cryptographic algorithms specified in the signature regulation dated back to 2008.

The IHE Cross Document Sharing Profile, described in section 3.5 on page 18, also does not provide specific technical details on how to secure medical data. It furthermore describes itself as an approach to provide support for control mechanisms specified by law or other regulations. The XDS Profile defines security and privacy policies on an enterprise level interpreted and adapted from regulations or laws, which have to be supported by an XDS environment. Security objectives

[15] *Gesundheitstelematikgesetz*

like an encrypted channel are listed, but recommendations for specific cipher suits are not part of the Profile and therefore leave room for interpretation regarding the implementation.

In a professional medical environment, a MDM is indispensable and is a vital asset to data security on mobile devices. It provides features to apply security policies and detect changes to the mobile device. This allows a better supervision of the mobile phone's state and helps to mitigate the risk of undetected privilege escalation or runtime manipulation. Projects like xCon[16] aim to bypass an app's jailbreak detection mechanism to conceal the fact that it is running on a jailbroken device. xCon patches functions on a low level to allow a system wide utilization of its counter detection functionalities. The xCon project was originally an open source software, but decided to move to a closed source approach in order to impede developers from working around its implemented bypass mechanisms.

Concluding from current trends in mobile device vulnerabilities and the governments interest in legally bypassing encryption standards, a mobile system's native security mechanisms are not sufficient in providing security for critical data. Laws and guidelines like the IHE Profiles provide vital indications for a general security framework for medical data. Due to the rigid and steady nature of laws and guidelines, they specify more general approaches and can not be adapted to up-to-date recommendations in such a frequent manner. Considering the stated facts, it is not recommended for a system architect to exclusively rely on these basic specifications, but to evaluate state of the art algorithms and mechanisms suitable for each specific purpose. An additionally implemented security layer, using state of the art cryptographic algorithms, and components securing an application's runtime are crucial for the safety of sensitive data and help to mitigate or even prevent potential threats. It has to be kept in mind that security mechanisms have to be evaluated and updated on a regular basis, because after all, quoting Bruce Schneier[17]: *"Security is a process, not a product."*.

[16] https://www.theiphonewiki.com/wiki/XCon
[17] https://www.schneier.com/essays/archives/2000/04/the_process_of_secur.html

Bibliography

References

[1] David Adrian et al. "Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice". In: *ACM Conference on Computer and Communications Security (CCS'15)*. 2015.

[16] Clara B. Aranda-Jan, Neo Mohutsiwa-Dibe, and Svetla Loukanova. "Systematic review on what works, what does not work and why of implementation of mobile health (mHealth) projects in Africa." In: *BMC public health* 14.1 (Jan. 2014), p. 188.

[17] Aswami Ariffin et al. "iOS Forensics: How Can We Recover Deleted Image Files with Timestamp in a Forensically Sound Manner?" In: *ARES*. 2013, pp. 375–382.

[19] Nadarajah Asokan and Cynthia Kuo. "Usable Mobile Security". In: *Distributed Computing and Internet Technology - 8th International Conference, {ICDCIT} 2012, Bhubaneswar, India, February 2-4, 2012. Proceedings*. 2012, pp. 1–6.

[23] Bundesministerium für Wirtschaft und Energie. *IT-Sicherheitsniveau in kleinen und mittleren Unternehmen*. Tech. rep. 2012.

[24] Steve Burgess. "Efficiency and freedom through mobile devices". In: *Tennessee Medicine* 105(5), 35 (2012).

[26] Mahinthan Chandramohan and Hee Beng Kuan Tan. "Detection of Mobile Malware in the Wild". In: *Computer* 45.9 (2012), pp. 65–71.

[27] James B. Comey. "Going Dark: Encryption, Technology, and the Balances Between Public Safety and Privacy". In: *Federal Bureau of Investigation* (2015).

[28] Michael P. Craven, Alexandra R. Lang, and Jennifer L. Martin. "Developing mHealth Apps with Researchers: Multi-Stakeholder Design Considerations". English. In: *Design, User Experience, and Usability. User Experience Design for Everyday Life Applications and Services SE - 2*. Ed. by Aaron Marcus. Vol. 8519. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 15–24.

[29] Rüdiger Dierstein. "Duale Sicherheit - IT-Sicherheit und ihre Besonderheiten". In: *G. Müller, A. Pfitzmann (Eds.): Mehrseitige Sicherheit in der Kommunikationstechnik*. Addison-Wesley, 1997.

[31] Cosmetics and medical devices European Commision, Directorate B, Unit B2. *Medical devices: Guidance document - Classification of medical devices*. DG health and consumer, June 2010.

[32] European Parliament and the Council. *The protection of individuals with regard to the processing of personal data and on the free movement of such data*. Tech. rep. 1995.

[33] Simone Fischer-Hübner. *IT-security and Privacy: Design and Use of Privacy-enhancing Security Mechanisms*. Berlin, Heidelberg: Springer-Verlag, 2001.

[36] Thomas Grechenig et al. *Softwaretechnik: Mit Fallbeispielen aus realen Entwicklungsprojekten*. IT Informatik. Pearson Studium, 2010.

[38] Clemens Hlauschek et al. "Prying Open Pandora's Box: KCI Attacks against TLS". In: *9th USENIX Workshop on Offensive Technologies (WOOT 15)*. Washington, D.C.: USENIX Association, Aug. 2015.

[39] Andrew Hoog and Katie Strzempka. *iPhone and iOS Forensics: Investigation, Analysis and Mobile Security for Apple iPhone, iPad and iOS Devices*. 1st. Syngress Publishing, 2011.

[42] International Telecommunication Union. *The World in 2014: ICT Facts and Figures*. Tech. rep. 2014.

[43] Internet Engineering Task Force (IETF). *RFC 2898 - PKCS #5: Password-Based Cryptography Specification Version 2.0*. Tech. rep. 2000.

[44] Internet Engineering Task Force (IETF). *RFC 3394 - Advanced Encryption Standard (AES) Key Wrap Algorithm*. Tech. rep. 2002.

[45] Internet Engineering Task Force (IETF). *RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2*. Tech. rep. 2008.

[46] Internet Engineering Task Force (IETF). *RFC 7457 - Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)*. Tech. rep. 2015.

[47] Internet Engineering Task Force (IETF). *RFC 7525 - Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. Tech. rep. 2015.

[48] Misha Kay, Jonathan Santos, and Marina Takane. *mHealth: New horizons for health through mobile technologies*. World Health Organization, 2011.

[49] Lutz Kolbe and Thierry Jean Ruch. “Mobile Security : Herausforderungen neuer Geräte und neuer Nutzeransprüche”. In: *HMD : Praxis der Wirtschaftsinformatik*. HMD : Praxis der Wirtschaftsinformatik. - Springer Vieweg, ZDB-ID 10157311. - Vol. 51.2014, 295, p. 9-23 51.295 (2014), pp. 9–23.

[51] Mariantonietta Noemi La Polla, Fabio Martinelli, and Daniele Sgandurra. “A Survey on Security for Mobile Devices”. In: *Communications Surveys Tutorials, IEEE* 15.1 (2013), pp. 446–471.

[54] Neal Leavitt. “Today’s Mobile Security Requires a New Approach”. In: *Computer* 46.11 (Nov. 2013), pp. 16–19.

[55] Qing Li and Gregory Clark. “Mobile Security: A Look Ahead”. In: *Security Privacy, IEEE* 11.1 (2013), pp. 78–81.

[57] Donna Malvey and Donna J. Slovensky. *MHealth: Transforming Healthcare*. Springer, 2014.

[58] Garrett Mehl and Alain Labrique. “Prioritizing integrated mHealth strategies for universal health coverage”. In: *Science* 345.6202 (Sept. 2014), pp. 1284–1287.

[65] Organisation for Economic Co-operation and Development. *OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data*. Tech. rep. 2013.

[66] Nicholas Penning et al. “Mobile malware security challeges and cloud-based detection”. In: *Collaboration Technologies and Systems (CTS), 2014 International Conference on*. 2014, pp. 181–188.

[67] Ponemon Institute LLC. *2014 Cost of Data Breach Study: Global Analysis*. Tech. rep. 2014, p. 28.

[68] Tim Proffitt. *Forensic Analysis on iOS Devices*. Tech. rep. SANS Institute, 2012.

[69] Peter Teufl et al. “iOS Encryption Systems - Deploying iOS Devices in Security-Critical Environments”. 2013.

[71] Peter Torr. “Demystifying the threat modeling process”. In: *Security Privacy, IEEE* 3.5 (Sept. 2005), pp. 66–70.

[72] United Nations. *Guidelines for the Regulation of Computerized Personal Data Files, as adopted by General Assembly resolution 45/95 of 14 December 1990*. Tech. rep. 1990.

[73] US-CERT/NIST. *Format string vulnerability in VPN in Apple iOS before 5.1 allows remote attackers to execute arbitrary code via a crafted racoon configuration file.* Tech. rep. US-CERT/NIST, 2012.

[74] Samuel Warren and Louis Brandeis. "The Right to Privacy". In: *Harvard Law Review* No. 5 (1890).

[75] Alan F. Westin. "Privacy and freedom". In: *Washington and Lee Law Review* 25.1 (1968), p. 166.

[76] Jonathan Zdziarski. *Hacking and Securing iOS Applications - Stealing Data, Hijacking Software, and How to Prevent It*. O'Reilly, 2012.

Online References

[2] Heather Akers-Healy. *Vupen Contracts with NSA*. 2013. URL: https://www.muckrock.com/foi/united-states-of-america-10/vupen-contracts-with-nsa-6593/ (visited on 11/11/2015).

[3] Apple Inc. *About Entitlements*. 2014. URL: https://developer.apple.com/library/mac/documentation/Miscellaneous/Reference/EntitlementKeyReference/Chapters/AboutEntitlements.html (visited on 10/22/2015).

[4] Apple Inc. *About Information Property List Files*. 2015. URL: https://developer.apple.com/library/ios/documentation/General/Reference/InfoPlistKeyReference/Articles/AboutInformationPropertyListFiles.html (visited on 10/22/2015).

[5] Apple Inc. *Code Signing Overview*. 2012. URL: https://developer.apple.com/library/ios/documentation/Security/Conceptual/CodeSigningGuide/AboutCS/AboutCS.html (visited on 10/22/2015).

[6] Apple Inc. *Common Crypto*. 2007. URL: https://developer.apple.com/library/ios/documentation/System/Conceptual/ManPages_iPhoneOS/man3/CC_crypto.3cc.html\# (visited on 10/02/2015).

[7] Apple Inc. *Custom Keyboard*. 2014. URL: https://developer.apple.com/library/ios/documentation/General/Conceptual/ExtensibilityPG/Keyboard.html (visited on 09/10/2014).

[8] Apple Inc. *File System Basics*. 2015. URL: https://developer.apple.com/library/ios/documentation/FileManagement/Conceptual/FileSystemProgrammingGuide/FileSystemOverview/FileSystemOverview.html (visited on 05/20/2015).

[9] Apple Inc. *iOS Developer Library*. 2012. URL: https://developer.apple.com/library/ios/documentation/cocoa/reference/foundation/Classes/NSHTTPCookieStorage_Class/Reference/Reference.html (visited on 09/10/2014).

[10] Apple Inc. *iOS Security. April 2015*. 2015. URL: http://www.apple.com/business/docs/iOS_Security_Guide.pdf (visited on 07/14/2015).

[11] Apple Inc. *iOS Security. October 2014*. 2014. URL: http://www.apple.com/business/docs/iOS_Security_Guide.pdf (visited on 12/01/2014).

[12] Apple Inc. *iOS Security. September 2015*. 2015. URL: http://www.apple.com/business/docs/iOS_Security_Guide.pdf.

[13] Apple Inc. *PhotoPicker*. 2013. URL: https://developer.apple.com/library/ios/samplecode/PhotoPicker/Introduction/Intro.html (visited on 07/14/2015).

[14] Apple Inc. *Product security certifications, validations, and guidance for iOS*. URL: https://support.apple.com/en-us/HT202739 (visited on 05/20/2015).

[15] Apple Inc. *UIApplicationDelegate*. 2014. URL: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/index.html (visited on 08/22/2015).

[18] Lucian Armasu. *iOS 9 Packed With Many New Security Features, Patches*. 2015. URL: http://www.tomshardware.com/news/ios-9-new-security-features,30106.html (visited on 11/09/2015).

[20] Karthikeyan Bhargavan et al. *Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS*. 2014. URL: https://secure-resumption.com/tlsauth.pdf (visited on 12/01/2014).

[21] John O. Brennan. *Global Security Forum 2015: Opening Session*. 2015. URL: http://csis.org/files/attachments/151116_GSF_OpeningSession.pdf.

[22] Wolfgang Breyha et al. *bettercrypto.org - Applied Crypto Hardening*. 2014. URL: https://bettercrypto.org/static/applied-crypto-hardening.pdf (visited on 12/01/2014).

[25] CDW LLC. *Mobile Device Management: Not What It Used To Be*. 2012. URL: http://webobjects.cdw.com/webobjects/media/pdf/mobile-device-df.pdf (visited on 12/01/2014).

[30] Tom Eston et al. *OWASP Mobile Security Project - Mobile Application Threat Model*. 2014. URL: https://www.owasp.org/index.php/OWASP_Mobile_Security_Project\#tab=OWASP_Mobile_Threat_Model_Project (visited on 08/31/2014).

[34] Lorenzo Franceschi-Bicchierai. *Somebody Just Claimed a \$1 Million Bounty for Hacking the iPhone*. 2015. URL: http://motherboard.vice.com/read/somebody-just-won-1-million-bounty-for-hacking-the-iphone (visited on 11/11/2015).

[35] Dean Garfield. *Tech Responds to Calls to Weaken Encryption*. 2015. URL: https://www.itic.org/news-events/news-releases/tech-responds-to-calls-to-weaken-encryption (visited on 11/24/2015).

[37] Jason Haddix, Daniel Miessler, and Jack Mannino. *OWASP Mobile Security Project - Top Ten Mobile Risks*. 2014. URL: https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks (visited on 09/01/2014).

[40] Inc. IHE International. *IHE IT Infrastructure (ITI) Technical Framework - Volume 1 (ITI TF-1) Integration Profiles*. 2015. URL: http://www.ihe.net/uploadedFiles/Documents/ITI/IHE_ITI_TF_Vol1.pdf.

[41] Inc. IHE International. *IHE IT Infrastructure (ITI) Technical Framework - Volume 2x (ITI TF-2x) Volume 2 Appendices*. 2015. URL: http://www.ihe.net/uploadedFiles/Documents/ITI/IHE_ITI_TF_Vol2x.pdf.

[50] Andreas Kurtz. *What Apple Missed to Fix in iOS 7.1.1*. 2014. URL: http://www.andreas-kurtz.de/2014/04/what-apple-missed-to-fix-in-ios-711.html (visited on 08/31/2014).

[52] Adam Langley. *Apple's SSL/TLS bug*. 2014. URL: https://www.imperialviolet.org/2014/02/22/applebug.html (visited on 08/31/2014).

[53] Issie Lapowsky. *After Paris, Encryption Will Be a Key Issue in the 2016 Race*. 2015. URL: http://www.wired.com/2015/11/after-paris-encryption-will-be-a-key-issue-in-the-2016-race/ (visited on 11/24/2015).

[56] Natasha Lomas. *Everything You Need To Know About iOS 8 Keyboard Permissions (But Were Afraid To Ask)*. 2014. URL: http://techcrunch.com/2014/10/04/everything-you-need-to-know-about-ios-8-keyboard-permissions-but-were-afraid-to-ask/ (visited on 09/01/2014).

[59] Andre Meister. *BSI Vertrag mit VUPEN*. 2014. URL: https://fragdenstaat.de/anfrage/vertrag-mit-vupen/ (visited on 11/11/2015).

[60] Zheng Min, Xue Hui, and Wei Tao. *Background Monitoring on Non-Jailbroken iOS 7 Devices — and a Mitigation*. 2014. URL: http://www.fireeye.com/blog/technical/2014/02/background-monitoring-on-non-jailbroken-ios-7-devices-and-a-mitigation.html (visited on 09/10/2014).

[61] Erik Miyake. *Safari 5.1 Cookie.binarycookie File Format [DRAFT]*. 2012. URL: http://www.tengu-labs.com/documents/Miyake-SafariCookie.binarycookieFormat0_2[Draft].pdf (visited on 10/01/2014).

[62] Robert Napier. *RNCryptor - Cross-platform AES data format and implementations*. 2014. URL: http://rncryptor.github.io/ (visited on 08/31/2014).

[63] Serkan Oezkan. *CVE Details - iOS - Vulnerability Statistics*. 2015. URL: http://www.cvedetails.com/product/15556/Apple-Iphone-Os.html (visited on 11/09/2015).

[64] Erlend Oftedal and Andrew van der Stock. *REST Security Cheat Sheet*. 2014. URL: https://www.owasp.org/index.php/REST_Security_Cheat_Sheet\#Input_validation_101 (visited on 08/31/2014).

[70] The American Health Information Management Association. *Just Think App! Mobile Health Apps 101: A Primer for Consumers*. 2013. URL: http://myphr.com/Stories/NewsStory.aspx?Id=596 (visited on 01/15/2015).

[77] Jonathan Zdziarski. *iOS Forensic Investigative Methods*. 2012. URL: http://www.zdziarski.com/blog/wp-content/uploads/2013/05/iOS-Forensic-Investigative-Methods.pdf (visited on 09/10/2014).

A Appendix

A.1 Use Cases

Initial Setup:

1. **"Identification summary"**
    - ***Title:*** Initial Setup
    - ***Scope:*** Start Screen
    - ***Level:*** System Goal
    - ***Actors:*** User
    - ***Summary:*** The user has entered a valid user name and password. Symmetric and asymmetric keys are created.
2. **"Scenarios"**
    - ***Pre-Conditions:*** A valid user name and password are supplied.
    - ***Main-Scenario:*** The user enters a user name and password. If they fulfill the predefined conditions an asymmetric key pair is generated (*CPR-key* and *CPUB-ke*y). Then a symmetric AES key (*M-key*) and a symmetric HMAC key (*HM-key*) are created. The *CPR-key* is encrypted and maced with the *M-key*, *HM-ke*y and a random *IV* and stored in the keychain. The *M-key* and *HM-key* are concatenated to create the *MHM-key* which is also encrypted by the keys derived from the user's passwor, and stored in the keychain. Finally the *CPUB-key* is also stored in the keychain.
    - ***Error-Scenario:*** If the user name and password do not match the criteria, the initial setup is not executed and an error message is shown to the user.
    - ***Alternative-Scenario:*** No alternative scenario.
    - ***Post-Conditions:*** The user is logged in and the user's *M-key*, *HM-key*, *CPR-key* and *CPUB-key* are available to the user.
3. **"Nonfunctional Constraints"**
    - Depending on the hardware the key generation takes a variable amount of time, therefore no timeout is set for the key generation. It should not take longer than 2 minutes to finish the setup.

Login:

1. **"Identification summary"**
    - ***Title:*** Login
    - ***Scope:*** Start Screen
    - ***Level:*** System Goal

- ***Actors:*** User
- ***Summary:*** The user has entered a valid user name and password. The *M-key*, *HM-key*, *CPR-key* and *CPUB-key* are retrieved form the keychain.

2. **"Scenarios"**

- ***Pre-Conditions:*** A valid user name and password are supplied. The user has to be already created and the user name and password combination has to match.
- ***Main-Scenario:*** The user enters a user name and password. If they match a user name and password combination which has already been created, the user's keys are retrieved from the keychain. At first, the *MHM-key* is retrieved, decrypted with the keys derived from the user's password, and split into *M-key* and *HM-key*. Then the encrypted *CPR-key* is retrieved from the keychain and decrypted with the *M-key*, *HM-key* and *IV*. Finally the CPUB-key is also fetched from the keychain.
- ***Error-Scenario:*** If the user name and password do not match a user stored in the keychain, the login process is interrupted, no keys are fetched an the user is presented an error message.
- ***Alternative-Scenario:*** No alternative scenario.
- ***Post-Conditions:*** The user is logged in and the user's *M-key*, *HM-key*, *CPR-key* and *CPUB-key* are available to the user.

3. **"Nonfunctional Constraints"**

- Depending on the hardware the key retrieval and decryption takes a variable amount of time, therefore no timeout is set for the key generation. It should not take longer than 2 minutes to finish the process.

Logout:

1. **"Identification summary"**

- ***Title:*** Logout
- ***Scope:*** Home Screen
- ***Level:*** System Goal
- ***Actors:*** User
- ***Summary:*** The user is logged out. The *M-key*, *HM-key*, *CPR-key* and *CPUB-key* are removed from the device memory.

2. **"Scenarios"**

- ***Pre-Conditions:*** The user has to be already logged in and the *M-key*, *HM-key*, *CPR-key* and *CPUB-key* have to be available to the user.
- ***Main-Scenario:*** The *M-key*, *HM-key*, *CPR-key* and *CPUB-key* are removed from the device memory. Open images are also removed from the memory.
- ***Error-Scenario:*** If the clean up was not successful the user receives an error message and has to try to logout again.
- ***Alternative-Scenario:*** If not all keys are available to the user, only the ones residing in the memory are removed.

- ***Post-Conditions:*** The user is logged out and the user's *M-key*, *HM-key*, *CPR-key*, *CPUB-key* and images are removed from memory.

3. **"Nonfunctional Constraints"**

    - It should not take longer than 30 seconds to finish the process.

Image Saving:

1. **"Identification summary"**

    - ***Title:*** Image Saving
    - ***Scope:*** Picture Screen
    - ***Level:*** System Goal
    - ***Actors:*** User
    - ***Summary:*** The user has taken a picture and the picture is directly encrypted and signed in the memory before storing it to the device storage.

2. **"Scenarios"**

    - ***Pre-Conditions:*** The user has to be already logged in and the *M-key*, *HM-key*, *CPR-key* and *CPUB-key* have to be available to the user. The user has to take a picture and enough free space needs to be available on the device storage.
    - ***Main-Scenario:*** The user takes a picture and before it is stored to the device storage it is encrypted and maced with the *M-key*, the *HM-key* and a random *IV*, which is stored with the picture. Additionally it is also signed with the *CPR-key* before it is written to the device storage. The Data Protection Class for the image is "Complete" (see chapter 4.3.2 on page 25).
    - ***Error-Scenario:*** If the encryption, signing or storage of the image was not successful the user is represented with an error message and the image is wiped from memory.
    - ***Alternative-Scenario:*** No alternative scenario.
    - ***Post-Conditions:*** The image is encrypted and maced with the *M-key*, the *HM-key* and a random *IV*, which is stored with the picture. Additionally it is also signed with the *CPR-key* before and stored on the device storage. No images are present in the device memory.

3. **"Nonfunctional Constraints"**

    - It should not take longer than 30 seconds to finish the process.

Image Opening:

1. **"Identification summary"**

    - ***Title:*** Image Opening
    - ***Scope:*** Home Screen
    - ***Level:*** System Goal
    - ***Actors:*** User

- ***Summary:*** A user's image is retrieved from the device storage, the signature is validated and the image is decrypted before it is opened.

2. **"Scenarios"**

- ***Pre-Conditions:*** The user has to be already logged in and the *M-key*, *HM-key*, *CPR-key* and *CPUB-key* have to be available to the user. The user has to have a signed and encrypted image stored on the device storage.
- ***Main-Scenario:*** The image is fetched from the device storage, the signature is validated with the *CPUB-key* and it is decrypted with the the *M-key*, *HM-key* and the *IV*. If all steps are successful, the unencrypted image is present in memory and can be opened.
- ***Error-Scenario:*** If the retrieval, signature verification or decryption of the image were not successful the user is represented with an error message.
- ***Alternative-Scenario:*** No alternative scenario.
- ***Post-Conditions:*** The unencrypted, verified image is present in memory and is wiped from memory after closing.

3. **"Nonfunctional Constraints"**

- It should not take longer than 30 seconds to finish the process.

Image Uploading:

1. **"Identification summary"**

- ***Title:*** Image Uploading
- ***Scope:*** Home Screen
- ***Level:*** System Goal
- ***Actors:*** User
- ***Summary:*** A user's encrypted and signed images are retrieved from the device storage and sent to the server.

2. **"Scenarios"**

- ***Pre-Conditions:*** The user has to be already logged in and the servers TLS certificate has to be present for SSL pinning. The user has to have at least one signed and encrypted image stored on the device storage. The device is able to reach and create a secure connection to the server.
- ***Main-Scenario:*** The images are fetched from the device storage. After verifying the remote servers SSL certificate with the locally stored server certificate a secure connection is established and the images are sent.
- ***Error-Scenario:*** If the retrieval of an image was not successful the user is represented with an error message. If the server can not be reached within a predefined amount of time or the SSL certificate can not be validated the user receives an error and the process is aborted.
- ***Alternative-Scenario:*** No alternative scenario.
- ***Post-Conditions:*** The encrypted and signed images are retrieved by the remote server.

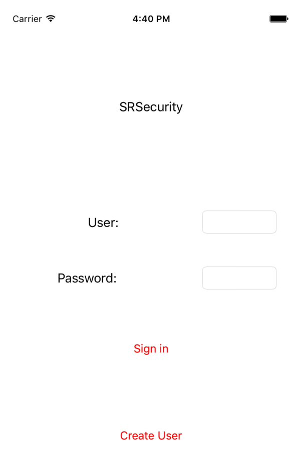
3. **”Nonfunctional Constraints”**

    - The process depends on the connection bandwidth, the image size and the remote and local device hardware. The upload is done in the background and the user should not close the app, before he receives a success message.

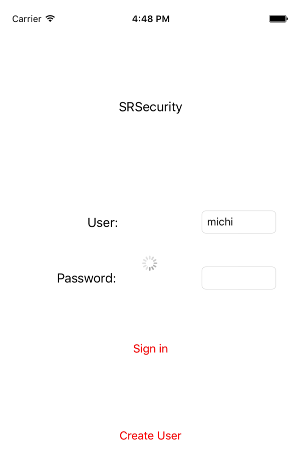
A.2 Views

Welcome View

- Show the input fields for the user name and the password (see figure A.1a on page 111)
- Login User with the user name and the password supplied
- Create User with the user name and the password supplied
- An activity indicator is shown during the ”initial setup” (see figure A.1b on page 111)
- An error message is displayed when the password is not valid (see figure A.2a on page 112)
- An error message is displayed when the user name is already taken (see figure A.2c on page 112)
- An error message is displayed when the user name and the password combination do not match (see figure A.2b on page 112)
- An error message is displayed when the user name does not exist (see figure A.2b on page 112)
- An error message is displayed when ”initial setup” fails (see figure A.2b on page 112)

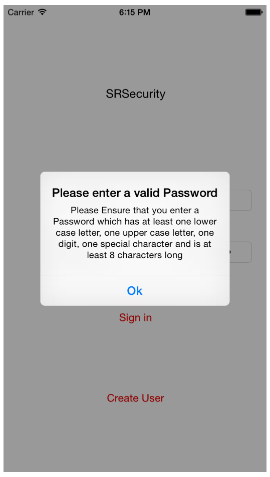


(a) Default



(b) Generating keys

Figure A.1: Welcome View - General



Carrier 6:15 PM

SRSecurity

Please enter a valid Password

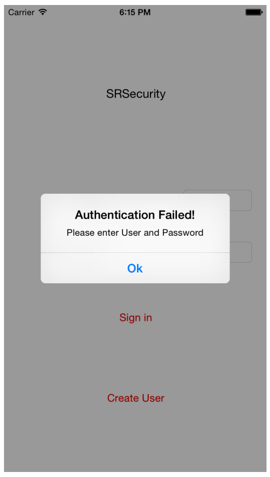
Please Ensure that you enter a Password which has at least one lower case letter, one upper case letter, one digit, one special character and is at least 8 characters long

Ok

Sign in

Create User

(a) Password not valid



Carrier 6:15 PM

SRSecurity

Authentication Failed!

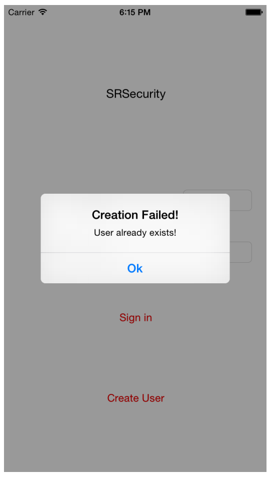
Please enter User and Password

Ok

Sign in

Create User

(b) General error



Carrier 6:15 PM

SRSecurity

Creation Failed!

User already exists!

Ok

Sign in

Create User

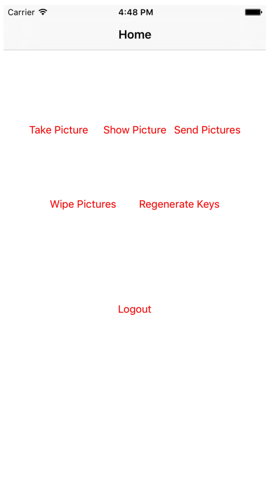
(c) User already exists

Figure A.2: Welcome View - Error

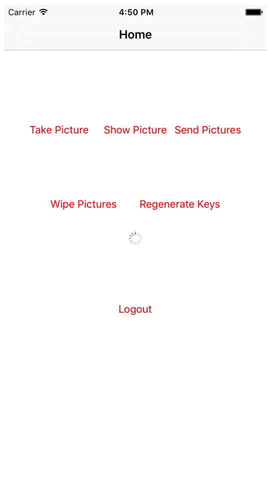
Home View

- Enable the user to switch to the ”Take Picture View” (see figure A.3a on page 113)
- Enable the user to switch to the ”Show Picture View” (see figure A.3a on page 113)
- Enable the user to send her/his stored pictures (see figure A.3a on page 113)
- A success message is displayed when the ”send pictures” process is finished (see figure A.4a on page 113)
- An error message is displayed when the ”send pictures” process fails (see figure A.4b on page 113)
- An error message is displayed when the retrieval of the pictures fails (see figure A.4c on page 113)
- Enable the user to delete all her/his stored pictures (see figure A.3a on page 113)
- The user has to confirm that she/he wants to delete all her/his pictures (see figure A.5a on page 114)
- A success message or an error message is displayed when the ”wipe pictures” process is finished (see figure A.5b on page 114)
- An activity indicator is shown during the ”wipe pictures” process (see figure A.3b on page 113)
- Enable the user to regenerate her/his keys (see figure A.3a on page 113)
- The user has to confirm that she/he wants to regenerate her/his keys and to delete all her/his pictures (see figure A.6a on page 114)
- The user has to provide a password for the newly generated keys (see figure A.6b on page 114)
- A success message or an error message is displayed when the ”regenerate keys” process is finished (see figure A.6c on page 114)

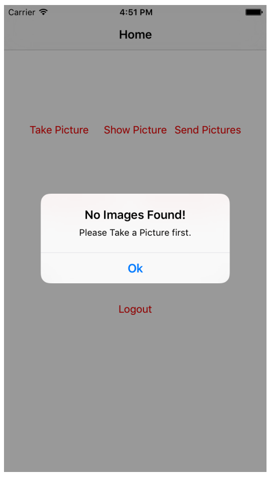
- An activity indicator is shown during the ”regenerate keys” process (see figure A.3b on page 113)
- An info message is displayed when the user does not have any stored pictures and wants to send, show or wipe his/her pictures (see figure A.3c on page 113)
- Enable the user to logout (see figure A.3a on page 113)
- A successful logout removes all keys from memory and changes the view to the ”Welcome View”



(a) Default



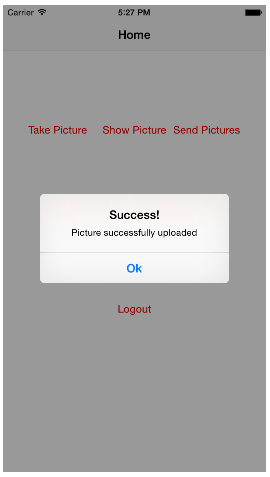
(b) Activity indicator regenerating keys or wiping pictures



No Images Found!
Please Take a Picture first.

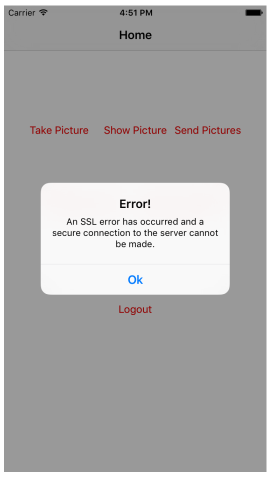
(c) No pictures message

Figure A.3: Home View - General



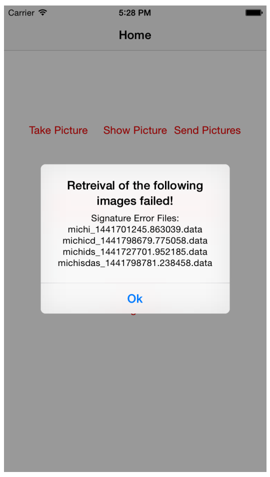
Success!
Picture successfully uploaded

(a) Send pictures success



Error!
An SSL error has occurred and a secure connection to the server cannot be made.

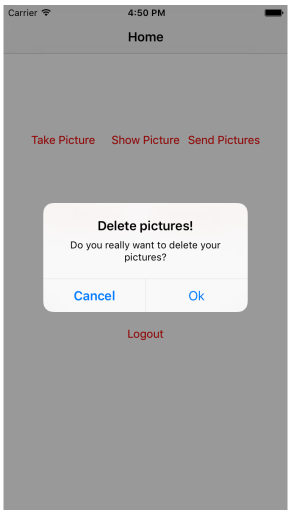
(b) Send pictures error



Retreival of the following images failed!
Signature Error Files:
michi_1441701245.863039.data
michicd_1441798679.775058.data
michids_1441727701.952185.data
michisdas_1441798781.238458.data

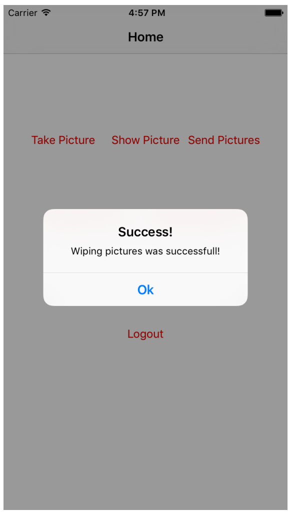
(c) Retrieve pictures error

Figure A.4: Home View - Send pictures



Delete pictures!
Do you really want to delete your pictures?
Cancel Ok

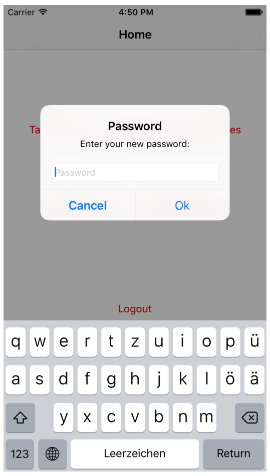
(a) Wipe pictures confirm



Success!
Wiping pictures was successfull!
Ok

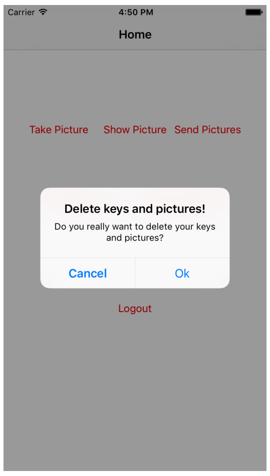
(b) Wipe pictures success

Figure A.5: Home View - Wipe pictures



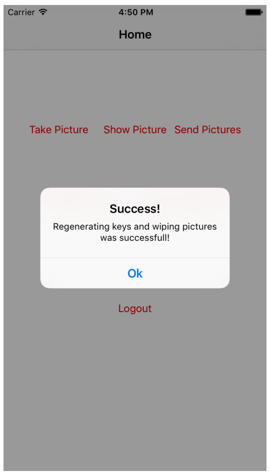
Password
Enter your new password:
Password
Cancel Ok

(a) Regenerate keys confirm



Delete keys and pictures!
Do you really want to delete your keys and pictures?
Cancel Ok

(b) Regenerate keys password



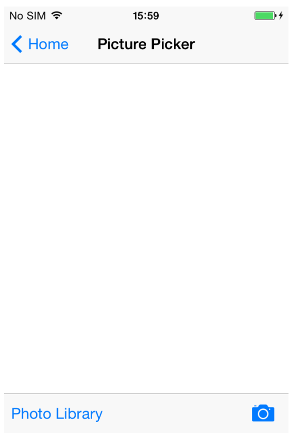
Success!
Regenerating keys and wiping pictures was successfull!
Ok

(c) Regenerate keys success

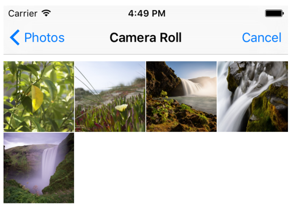
Figure A.6: Home View - Regenerate keys

Take Picture View

- Enable the user to take a picture with the device camera (see figure A.7a on page 115 or figure A.8a on page 115)
- Enable the user to choose a picture from the the device library (for testing purpose only) (see figure A.7b on page 115)
- An info message is displayed when the picture was successfully encrypted, signed and stored to the device storage (see figure A.8b on page 115)

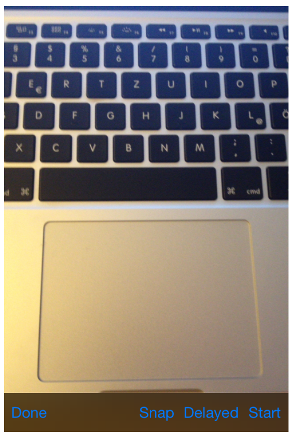


(a) Device library or camera

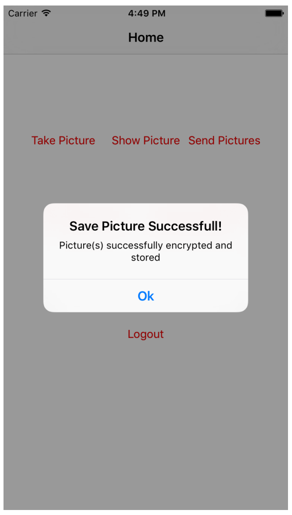


(b) Gallery

Figure A.7: Take Picture View - Overview



(a) Camera



Save Picture Successfull!

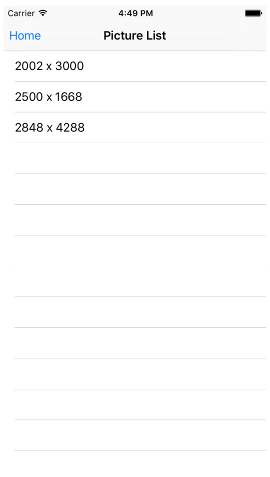
Picture(s) successfully encrypted and stored

(b) Success

Figure A.8: Take Picture View - Camera success

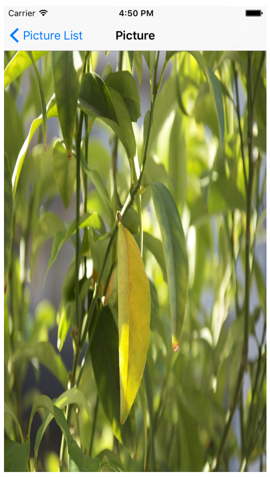
Show Picture View

- Enable the user to list all her/his stored images (see figure A.9a on page 116)
- Enable the user to open a picture from the list (see figure A.9b on page 116)
- An error message is displayed when the picture retrieval fails (see figure A.9c on page 116)

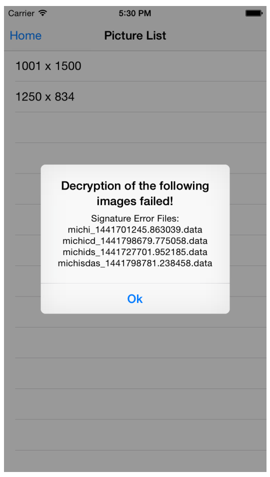


2002 x 3000
2500 x 1668
2848 x 4288

(**a**) List pictures



(**b**) Show picture



1001 x 1500
1250 x 834

Decryption of the following images failed!
Signature Error Files:
michi_1441701245.863039.data
michicd_1441798679.775058.data
michids_1441727701.952185.data
michisdas_1441798781.238458.data

(**c**) Retrieval error

Figure A.9: Show Picture View - General