University of
Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study programme/specialization: *Master in Industrial Economics,* *Specialization Investment and Finance*	Spring semester, 2019 Open
Author: *Elisabeth Bratli*	Elisabeth Bratli (signature of author)
Programme coordinator: *Roy Endré Dahl* Supervisor: *Hein Meling*	
Title of master's thesis: *Document Verification System on iOS with Face ID/Touch ID*	
Credits: *30 ECTS*	
Keywords: *Document verification* *Cryptography* *Biometrics* *iOS* *Swift* *Digitalization* *Document storage*	Number of pages: 70 Other: link to code is included in Appendix Stavanger, *15.06.2019*

Title page for Master's Thesis
Faculty of Science and Technology

UNIVERSITY OF STAVANGER

MASTER'S THESIS

Document Verification System on iOS with Face ID/Touch ID

Author:
Elisabeth Bratli

Supervisors:
Roy Endré Dahl
Hein Meling

A thesis submitted in fulfillment of the requirements for the degree of Master in Industrial Economics

in the

Faculty of Science and Technology
Department of Safety, Economics and Planning

June 15, 2019

Abstract

The level of digitalization in today’s society has contributed to both the private and public sector in several ways. Modern technologies allow for automated processes, electronic communication, cloud storage, and much more. These technologies provide significant benefits in various situations, although they do also come with some issues and vulnerabilities. The vulnerabilities that will be addressed in this thesis are the electronic identification of individuals and document authenticity.

Managing sensitive information has become more secure in recent years with the introduction of new technologies that allow for electronic identification. The requirements for electronic identification vary from what information an individual is requesting, but the methods used today all have some form of vulnerability. A method that would provide further security is the use of biometrics, as biometrics are unique for every person, and is difficult to steal. Both fingerprint and face recognition can uniquely identify a person and are becoming standardized on most modern mobile devices.

The purpose of this project is to design, implement, and evaluate a document verification system with globally distributed participants, that uses biometrics as the form of identification. Such a system would improve security and reliability to a vast selection of industry sectors, including banking, oil and gas, and validating certificates.

The result of this project is presented in the form of an iOS application, DCApp, that uses biometrics to ensure a secure document verification system. A degree certificate system has been implemented for this sample application.

Lastly, an evaluation of the performance and security of the system will be provided. The system is compared to current solutions and is discussed in the context of some use cases.

Acknowledgements

First and foremost, I would like to thank my two supervisors, Roy Endré Dahl and Hein Meling for invaluable guidance throughout this project. I would especially like to thank Roy Endré for providing insight into the societal need for the system designed in this thesis. To Hein, I would like to extend my gratitude for supporting me during weekly meetings, and for his assistance in clearing up roadblocks encountered during the implementation of the system.

Lastly, I would like to thank Daniel Barati and my family for their love and support.

Contents

Chapter 1

Introduction

1.1 Digital Transformation

In recent years there has been much focus on digitalizing, both in the private and public sector. Digital communication is becoming widely used in most contexts, and thus, there is a huge societal need for secure, electronic identification. Steps towards a more digital society provide greater availability of information, more effective processes, and much more. One example is that certain documents can be provided automatically if a person identifies themselves electronically. These documents include grade transcripts, bank statements, and pre-qualification letters needed in order to place bids on property. In Norway, all of these documents can be provided automatically to any individual who provides the required electronic identification, making the processes very effective. In some cases, identification through email and password is sufficient, some require a social security number, and others require the person to identify themselves using BankID, an electronic form of identification and signing method. BankID will be explained further in Section 6.3.1. All of these identification methods have flaws and vulnerabilities, and a method to secure these vulnerabilities would be of great significance to society. The implementation of biometrics in identification would eliminate most of the flaws associated with electronic identification.

1.2 Use Cases

In today's society, there are many challenges regarding document verification and authentic identification. Implementing a secure document verification system based on biometrics in relevant sectors could stimulate a positive development to society. Some of the use cases for the system are mentioned below.

1.2.1 Certificates of Employment and Degree Certificates

Certificates of employment and degree certificates are often used in the same context, which is in the application process for a new job or school. Both of these types of certificates can easily be forged or modified, and individuals can often be tempted to do so in their favor. A method to verify that a certificate is authentic is to make a phone call the individual or institute who allegedly issued the certificate. However, if an individual has strong enough incentives to forge a certificate, they could find a solution to manipulate this verification step.

A straightforward and more secure method to authenticate the certificate would be to use the system that will be presented in this thesis. The need for biometrics for both the certificate owner and signer (issuer) prevents someone who has incentives to modify or forge a certificate in his favor to do so.

1.2.2 Banking

The banking sector has some vulnerabilities, such as identity theft and document forgery, which will be further addressed in Section 6.3. A system where biometrics must be used to verify one's identity and for document signing would provide advanced security. It would provide immense protection against identity theft, and ensure that documents presented, e.g. proof of a financial situation, are not modified.

1.2.3 Medicine

The documentation of medical journals has improved in the past years, but also in this sector there is room for improvements. Both the security and availability of an individual's medical records are important. A system where all medical records could be securely stored and be available to the record owner would make it simple for individuals to share their medical history in situations where needed, such as when applying for insurance or if they are abroad. Information such as vaccines, allergies, blood type, medical conditions, and past operations would be available on an individual's device and could easily be authenticated and verified.

Another example in medicine is the prescription of medications. Situations that could arise is the forgery or modifications of prescriptions, or using someone else's prescription. This system has become more secure with the introduction of electronic prescriptions, but still allows for some possibility of fraud. If a system such as the one that will be explained throughout this thesis is implemented, individuals would provide an authentic biometric identification and prove that the prescription is not forged or modified through a simple verification process.

1.2.4 Oil and Gas

The oil and gas industry has experienced vast amounts of digitalization and is continuing to implement new solutions. According to article [1], three out of four oil and gas companies have been exposed to severe cyber attacks globally, and the threat of such attacks is increasing in correlation with the degree of digitalization. The cost of cyber attacks can be enormous in the form of lost revenue or data, and it can also affect health, safety, and environment (HSE) in operations. Hence, proper security measures should be taken to prevent attacks. Although the implementation of newer systems is expensive, the financial savings will be more significant. Secure systems protect companies against lost control, lost revenue and data, and secures the companies' reputations. In the oil and gas sector, there is a substantial flow of documents, both internally and between companies. How a secure document verification system will benefit the oil and gas industry will be evaluated in Section 6.5.

1.3 The Project's Purpose

The goal of this project is to build a document verification process with globally distributed participants. A participant can be a signer, a verifier, or a document owner. The contents of the document can be published openly or kept private between the main participants. However, even if it is kept private, the signature of the document must be published openly. This is to guarantee the trust relationships. The document owner should be able to verify the document but is otherwise considered untrusted. This is to ensure that the content is correct and unmodified, and prevents the owner from changing the content in his favor if he has incentives to do so. Documents should be stored such that they are easily accessible to the document owner. To secure further trust, biometrics will be used as the form of identification.

1.4 Sample Application

In order to demonstrate the system explained in Section 1.3 an iOS application, named DCApp (Degree Certificate Application), acting as a degree certificate (grade transcript) system has been built. The university acts as a signer, the student acts as a document owner, and any third party whom the student shares a document with acts as a verifier. A verifier could, for example, be a person at an admissions office or an employer. This application is a prototype implementation of the document verification system that will be explained throughout this thesis. Although DCApp demonstrates the fundamentals behind the system, not all features for the system described will be implemented. The prioritized features of the system are biometric identification, cryptography, document storage, and validity of the verification. This is mainly due to the time constraint of the project, and thus the most important, underlying principles have been prioritized. Also, a real world application would require much larger scalability, meaning, for example, that it would have to be available on other platforms than iOS.

1.5 Thesis Outline

Through the use of current technologies, this project creates and evaluates the use of biometrics in a document verification system. The thesis is organized as follows:

- Chapter 2 introduces theoretical concepts and background used to create the system explained in Section 1.3. This chapter goes into detail on one of the major topics in this thesis, which is cryptography.

- Chapter 3 explains some ongoing work which is relevant to the principles in this project, as well as the relevant technologies used to develop DCApp. Specifically, iOS technologies needed for this application and services to handle the backend of the application are described.

- Chapter 4 describes the design used to implement the system as described in Section 1.4. Cryptographic methods are evaluated, and the application logic for student and administrative interfaces are explained.

- Chapter 5 describes this implementation in further detail. This includes class structure, user authentication, biometric identification, cryptographic operations, and communication between participants.

- Chapter 6 presents an evaluation of the design and implementation. Here the security of the system, possible impact on society through several use cases, as well as possible improvements for future work are discussed.

- Chapter 7 provides the final conclusion on how DCApp and the system explained throughout the thesis would impact industry sectors and companies.

Chapter 2

Background

In this chapter, the project which this thesis is based on is briefly introduced, as well as other content which is important to understand for the document verification system which this thesis describes.

2.1 BBChain Project

Blockchain technology was first used in the cryptocurrency known as Bitcoin, but has since been found useful in other cases as well. Blockchain allows digital information to be distributed and verified by using cryptography. It is a time-stamped series of an immutable record of data that is managed by a cluster of computers not owned by any single entity, thus has no central authority. [2]

BBChain is a project funded by the Research Council of Norway which this project is based on. The goal of the BBChain project is to use blockchain technology for secure and permanent storage of data whose authenticity is verifiable [3].

Taking grade transcripts as an example, the principle is to create a storyline of an individual's study course. The blockchain for this individual will consist of smaller blocks representing material such as assignments or lab approvals, and more significant blocks such as final exams and final thesis. A third party, a verifier, would be able to verify the student's education from the blockchain.

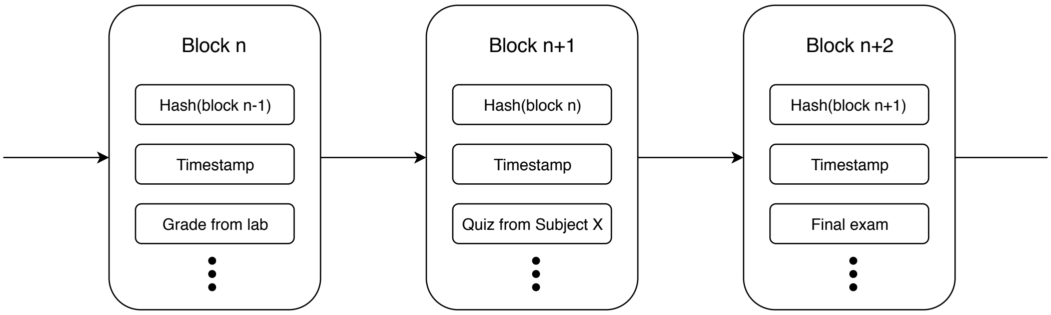


FIGURE 2.1: Example blockchain.

A simple illustration of the blockchain is shown in Figure 2.1. Each block contains the hash of the previous block, a time-stamp, and other relevant information. In this example, information about a student's education is stored, and the blockchain can also be used to store information such as if the student loses their private key. This information is important to all participants communicating with the student, as it means that the private key is compromised.

DCApp is based on some of the features which are to be implemented in BBChain, such as biometric identification and asymmetric encryption. The main goal for both projects is to create a more solid document verification system by using existing technologies available in iOS and cryptography. Blockchain is one of the important technologies in the BBChain project, but is not used in DCApp. Instead, the implementation of DCApp is done with other technologies in order to create a robust document verification system.

2.2 Cryptography

Cryptography is important for secure communication in any digital system, and can be categorized into symmetric and asymmetric cryptography. In this project, cryptography is crucial, and is used for encryption, decryption, signing, and verification of data.

Symmetric encryption requires that the same key is used for both encryption and decryption of messages, and thus the key is shared among two or more parties. One of the issues with this cryptographic method is the distribution of the key. It relies on either physical delivery of the key, or some other channel for the delivery [4]. If the key is delivered over an unprotected channel, an eavesdropper can intercept the key, and thus be able to read messages delivered over the channel as well. This is known as the man-in-the-middle attack [5]. Another problem with symmetric encryption is that since the communicating parties have the same key, there is no way of knowing which of them created a message, and hence it cannot be used to create signed messages. From this, it is clear that symmetric encryption is not suited for the necessary cryptography required in DCApp.

Asymmetric encryption is also known as public key encryption. For this type of encryption, the parties involved have a pair of keys: a private one and a public one [6]. The private key is only known to the owner of the key pair, and the public key can be shared with any other party who wishes to communicate with the owner of the key pair, hence the name of the protocol. If a party, B, wishes to send a message to another party, A, the following steps are required with asymmetric encryption:

- B obtains a copy of A's public key.
- B uses A's public key to encrypt the message.
- B sends the encrypted message to A.
- A uses her private key to decrypt the message.

When one B encrypts a message with A's public key, A is the only party who can decrypt the message, since only the corresponding private key can decrypt it. An illustration of asymmetric encryption is shown in Figure 2.2.

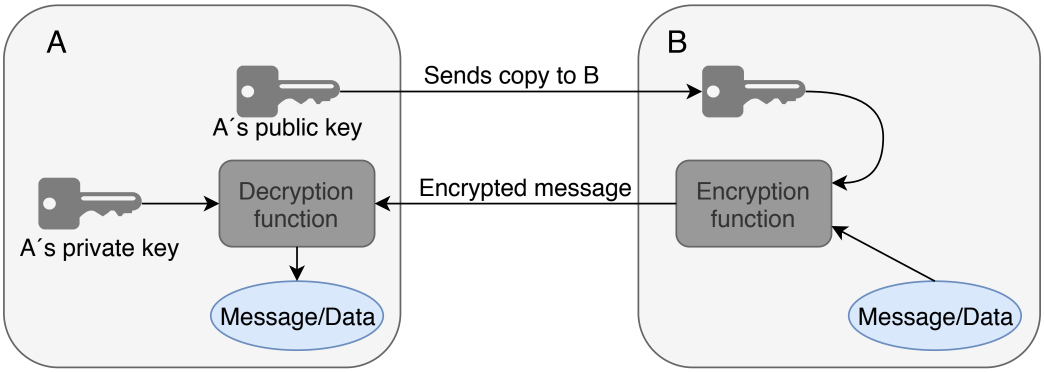


FIGURE 2.2: Illustration of asymmetric encryption, also known as public key encryption.

Although asymmetric encryption is more secure than symmetric encryption, it can also be susceptible to a man-in-the-middle attack. In a man-in-the-middle attack, if A wants to communicate with B, and vice versa, they are instead communicating with a man-in-the-middle, also called an attacker. The attacker has his own pair of keys which is used to encrypt and decrypt messages. Instead of A and B communicating directly, the messages are intercepted by the attacker. The attacker pretends to be B when communicating with A, and pretends to be A when communicating with B. This means that when A and B were exchanging public keys, the attacker intercepted them, and distributed his public key instead. Unless they were to exchange the keys during a physical meeting, neither A or B can know if the public key they receive is from an attacker. In a man-in-the-middle attack, if a message somehow is delivered directly between A and B, it can not be decrypted because they both obtain the attacker's public key, not each others'. The man-in-the-middle attack is illustrated in Figure 2.3.

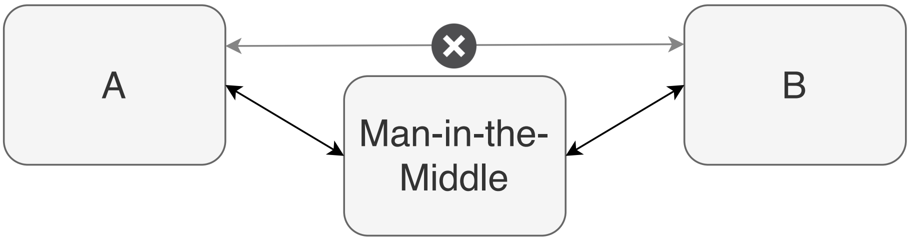


FIGURE 2.3: Illustration of man-in-the-middle attack.

To secure a system against the man-in-the-middle attack, it is necessary with a public key infrastructure (PKI). This is addressed in Section 2.3.

In addition to providing secure communication, asymmetric encryption provides the possibility of creating reliable digital signatures, which is a requirement in DCApp. With these features of asymmetric encryption, it is natural to use this for cryptography in this project.

2.2.1 Digital Signatures

Digital signatures guarantee that the contents of a message have not been altered [7]. A digital signature is created when a signer creates a one-way hash of the message, and then encrypts this hash with their private key. The signature is a digital code which is attached to the digital message in transit. In order for a receiver to verify that the message has not been modified, the digital signature can be used. Performing the same hashing algorithm on the message should create the same result as the decrypted digital signature. Identical hashes validate the message. This is illustrated in Figure 2.4.

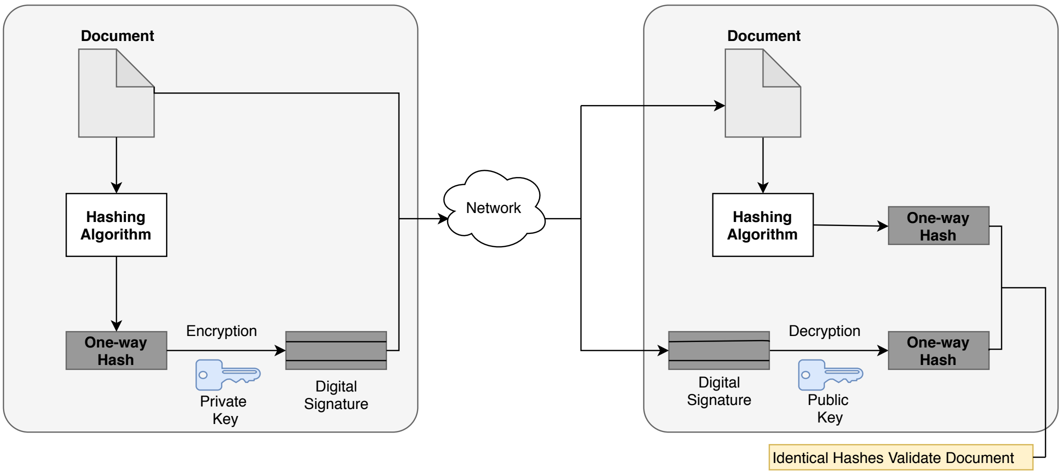


FIGURE 2.4: The process of verifying a digital signature.

Hashing Algorithms

A hashing algorithm is a cryptographic hash function designed to be a one-way function, infeasible to invert. It is a mathematical algorithm that maps data of arbitrary size to a hash of a fixed size. The size of this hash depends on what hashing algorithm is used. Some early hashing algorithms have been compromised, but the SHA-2 family is still considered to be secure. SHA-256 is a part of this family and is used to perform the hashing in this project. [8]

2.2.2 Cryptographic Keys

There are several types of public keys that can be used for cryptographic processes such as signing, verification, encryption, and decryption. Two of the most popular and widely used types are Elliptic Curve and RSA. When using Elliptic Curve Cryptography (ECC), 256-bit keys are used, while RSA uses 3072-bit keys to reach the same cryptographic strength. ECC is appealing for devices with limited storage or processing power due to its relatively small keys. Using these smaller keys without sacrificing security makes the key generation and exchange less cumbersome. Due to these characteristics, ECC is used in DCApp. [9]

2.3 Public Key Infrastructure

This section will address the solution to solving the largest trust issue with public key encryption, which is trusting that the sender of a public key is who they claim to be. An article from Techopedia, [10], covers the technology behind this solution quite thoroughly.

Public key infrastructure (PKI) is the solution to the problem associated with trusting the owner of a public key in asymmetric encryption. A PKI involves the participation of trusted third parties who can verify the identity of the parties wishing to engage in secure communication by issuing digital certificates. The protocol can be divided into five steps:

1. A subject sends his public key to a trusted third party called a registration authority (RA), who verifies the identity of the public key holder.

2. The RA instructs another body, the certificate authority (CA), to issue a digital certificate.

3. The certificate is issued to the subject, containing the subject's public key, identity and a digital signature computed by the CA from the subject's public key and identity [11].

4. The subject's certificate is sent to any party (verifier) who needs his public key. In DCApp this would be done when a verifies needs to verify the digital signature of a student's document, and thus requires the university's (subject's) public key.

5. The certificate, and thus the public key and identity of the subject, is verified when the verifier requests the CA's public key, and checks the digital signature of the certificate.

Using such certificates enables secure communication, given that the verifier can trust that there has not been a man-in-the-middle attack between communication with the CA. The process is illustrated in Figure 2.5. Since DCApp is a prototype application, a PKI has not been implemented, but it is important to understand the PKI in context to the design of a real world application. In Section 4.9, a method used for the sample application instead of a PKI will be described.

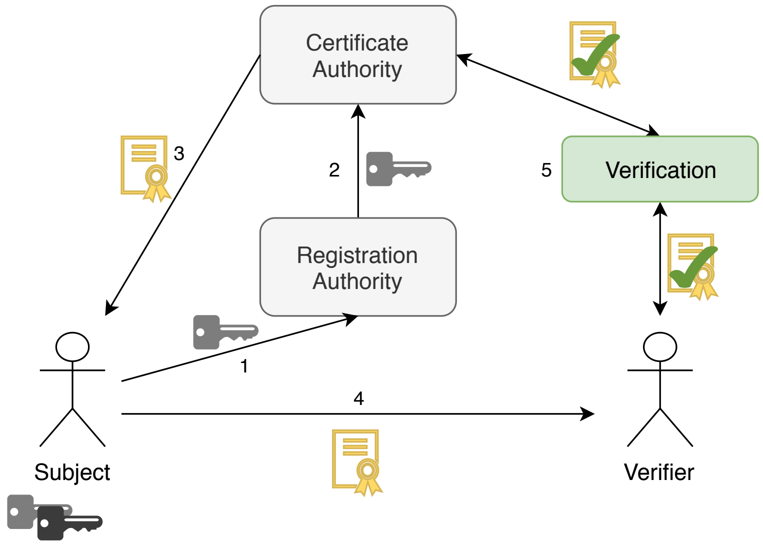


FIGURE 2.5: Using certificates to authenticate the identity of a subject.

2.3.1 Certificate Authority

A certificate authority (CA) is the trusted third party responsible for validating the identity of a person or organization. CAs are organizations consisting of people and technologies whose job it is to validate the identity of those seeking digital certificates. A certificate server generates a digital certificate, signed by the CA's private key, containing the subject's public key, and other relevant information about the subject, once the subject has been verified. The trustworthiness of a subject's identity is connected to the trustworthiness of the CA. It is therefore important that the receiver of a public key is familiar with the CA's procedures and protocols. [10]

2.4 Biometrics

Biometrics is the technical term for body measurements and calculations, and is relevant in this project because it will be the method used for identification. It refers to the metrics related to human characteristics. This includes features in the face, fingerprint, the iris, and any characteristic that can uniquely identify an individual. Biometrics can be used for authentication in computer science, and is normally used for identification and access control [12]. Biometric technology is integrated into several types of devices, including Apple products such as the iPhone. As of the release of iPhone 5S, biometrics is used for user authentication in the form of fingerprint recognition, called Touch ID. With the release of iPhone X, this has been replaced by face recognition, Face ID. Such features provide easy to use and secure methods of identifying the user.

2.4.1 Biometrics Versus Classical Authentication

Classical authentication schemes are based on the knowledge of the user, and takes the form of secret identifiers such as passwords or pin codes. These schemes are still widely used for identification and transactions, as they are user-friendly and simple to integrate.

One problem with this authentication scheme is that it is not necessarily unique, and could thus be stolen or guessed by an attacker. Another problem is that they become compromised if password databases are compromised. Passwords are usually hashed before being stored, and thus even if a password database is compromised, the attacker could not read the actual password. However, if a weak hash function is used, for example, MD5, the hash could be reverse-engineered, revealing the password [13]. Compromised password databases are also susceptible to dictionary attacks. These attacks consist of brute force attacks where the attacker tries different possible passwords, hashes them, and compare them to the hashes on the password database [14]. The difficulty of this attack is quite high, and for more advanced passwords (longer passwords containing lower case, upper case, numbers, and symbols), this type of attack is not feasible.

Although there is a vulnerability with a password becoming compromised, an advantage with classical authentication schemes over biometrics is that they are renewable at any time [15]. This is considered a strength compared to biometrics because if a user's biometrics is stolen, the user is unable to change his biometrics. It is therefore crucial for systems that use biometrics to be secure enough to handle this information. How this is done in iOS will be explained in Section 3.1.1.

2.5 iCloud and File-Based Storage

One of the requirements for this project is storing documents that belong to the document owner. The documents should be stored securely and be easily accessible. To achieve this, file-based storage can be used.

File-based storage is a method for storing data in a hierarchical structure, where the data is saved and organized in files and folders [16]. Another method for storing data is to store documents in a database. However, file-based storage is the preferred method in this project for reasons explained below.

File-based storage in the cloud is an efficient and inexpensive method to store data. Cloud-based solutions have become increasingly popular, and there are a lot of options to choose from. Some examples are solutions offered by Azure, AWS or Firebase. Firebase is used for some of its services, but the file storage offered is not used. Even if storage in the cloud is not considered very expensive, it will still become increasingly expensive as more people use the application, and store more documents. A way to prevent this is to use Apple's cloud solution, iCloud Drive. Documents connected to a user will be stored on their personal iCloud Drive, meaning that the application does not need to offer any other means of document storage. It can be considered a free service, because every Apple user has 5GB of free storage. Users only pay if they need to upgrade to more storage. The main disadvantage of using iCloud Drive is that it is mainly used on Apple devices, and hence the application is not very scalable. However, since this is a sample application solely for iOS, this is not a concern.

Chapter 3

Technology Review

This chapter explains the technologies used to build the document verification system, DCApp, in addition to the topics outlined in Chapter 2.

3.1 iOS Technologies

This section will explain some of the technologies integrated into products that run iOS, Apple's operating system, and what they offer to DCApp.

3.1.1 Secure Enclave Processor

An additional processor in iOS devices called the Secure Enclave is used for data protection, Touch ID, and Face ID [17]. The purpose of this processor is to handle information that is considered too sensitive for the application processor. In particular, it can be used to store cryptographic keys and biometric information in the form of mathematical representations [18]. The Secure Enclave does, however, have some restrictions for cryptographic keys. For example, it can only store and create 256-bit elliptic curve private keys, and preexisting keys cannot be imported [19]. It runs its own microkernel, which cannot be accessed by any programs or the operating system itself. When a request is sent to the Secure Enclave, the operations are performed in the Secure Enclave, and only the result is returned. Such operations can be:

- Signing some data, where a signature is returned as the result.
- Decrypting some data, where the decrypted data is returned as the result.
- Authenticating Face ID or Touch ID attempts. The result of this operation is a fail or pass.

3.1.2 iCloud Keychain

The iCloud Keychain, also referred to as the keychain, is a system developed by Apple which is built into every iPhone, Mac, and iPad. It is mostly used for password management, but is also used to store items such as credit cards and cryptographic keys. Passwords in the keychain are synchronized through iCloud across all Apple devices that a user approves, but cryptographic keys are not [20]. In order to synchronize these keys, they need to be retrieved from an encrypted backup from iTunes. In practice, this means that a user should have up-to-date backups in iTunes, that can be used if the device is lost or outdated. To access keychain items the services within the *Security framework*[1] is used. When items are protected by biometrics the services rely on the *LocalAuthentication framework*[2] to present this requirement to the user [23]. The Secure Enclave then carries out the authentication, returning only fail or pass as a result, which in turn grants or denies access to the keychain item. Figure 3.1 shows the essential parts and technologies required to retrieve items from the keychain.

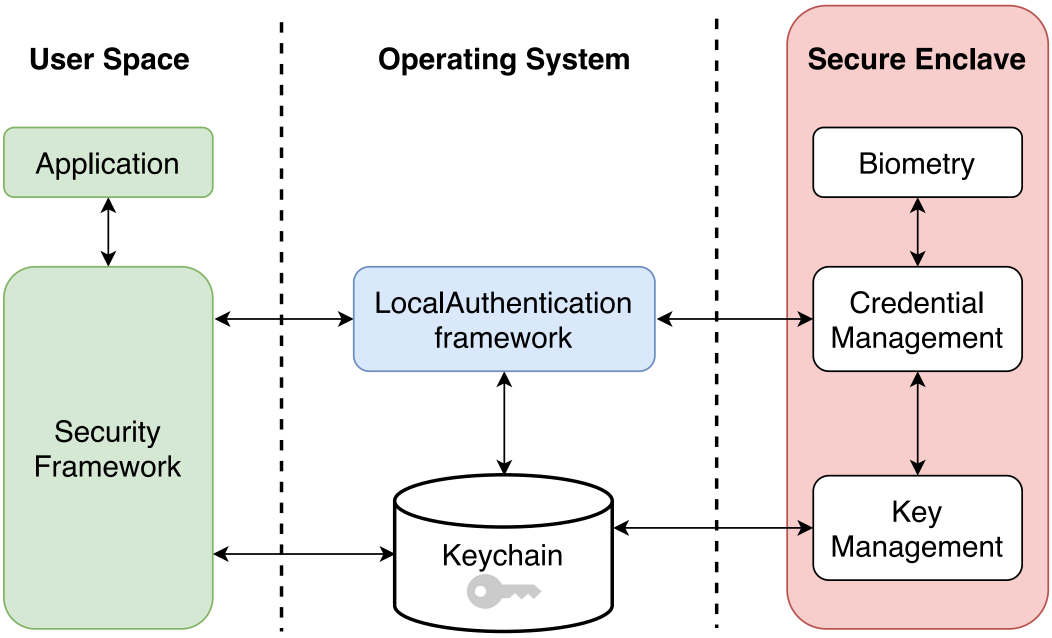


FIGURE 3.1: Illustration of the necessary parts for retrieving a keychain item.

When adding and storing items in the keychain, the items can have tokens attached to them, which sets rules for the items. These rules decide when and how the items can be obtained, such as what applications can use the items, and what type of authentication is required to use them.

[1] A framework available on iOS to encrypt and decrypt data [21].

[2] A framework for authenticating users with biometrics or a passcode [22].

3.1.3 Touch ID

Touch ID was Apple's first use of biometric recognition on their devices. It is a fingerprint recognition feature that allows users to unlock their Apple device, make purchases in digital media stores, and authenticate themselves. Touch ID is built into the home button, which can read the user's fingerprint from any direction. When the user uses the Touch ID feature, the data from the fingerprint is compared to the template data, which is stored in the Secure Enclave. For security reasons, only confirmation or denial is returned from the Secure Enclave, and no biometric data is exposed outside of the Secure Enclave. Although Touch ID is considered a secure method for identification, Apple still predicts that 1 in 50 thousand people can unlock a given iPhone, posing a small security risk. Touch ID was first introduced on the iPhone 5S in 2013, and has been implemented on all iPhone models since the iPhone 5S up until the iPhone 8 and 8 Plus [24]. After this, the Touch ID feature was replaced by Face ID. [18]

3.1.4 Face ID

Face ID is Apple's facial recognition system and was introduced on the iPhone X. The security risk associated with Face ID is smaller than Touch ID, where with Face ID only 1 in 1 million other people should be able to unlock a given iPhone. Face ID is also advanced enough to know if the person's face it is recognizing has the eyes open, is looking at the camera, and should be able to detect the user even with changes in hairstyle, facial hair, or glasses. The technology that allows for this is known as Apple's TrueDepth camera system, which is made up of several components. These components can be seen in Figure 3.2. [18]

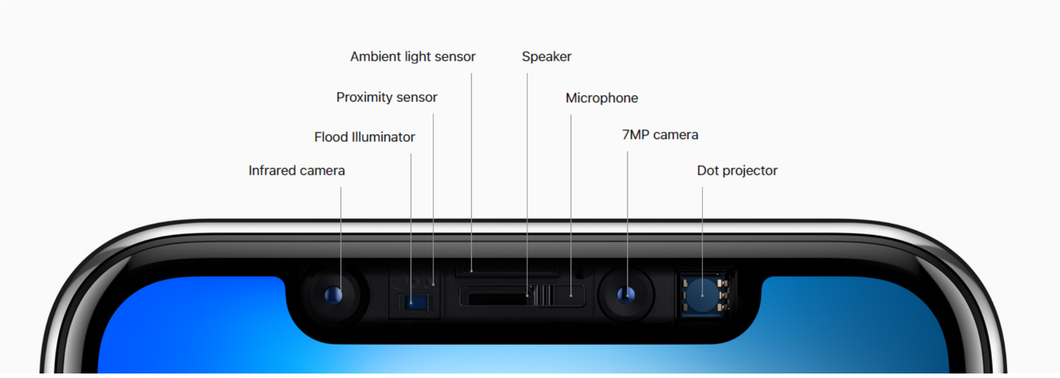


FIGURE 3.2: Components in Apple's TrueDepth camera system [25].

Working together, the sensors and components project 30 thousand infrared dots onto the user's face, which is used to map out all curves [26]. The data collected from the TrueDepth camera system is compared to the template data, which is stored in the Secure Enclave. The verification works the same way as Touch ID, and only confirmation or denial is returned. Another feature of Face ID is that its ability to recognize the user improves with usage. The template data which is used for comparison on identification is updated as the features of the user is altered, even if there are small changes. Hence, the more often a user uses Face ID, the more up to date the software is on the biometrics of the user.

For both Face ID and Touch ID, the biometrics of the user is converted to a mathematical representation. On each identification attempt, the biometrics provided by the user is converted to a mathematical representation and compared to the enrolled mathematical representation [18].

3.1.5 AirDrop

AirDrop is a service in Apple's operating systems, meaning that it is integrated into all iPhones and other Apple products since 2011. It is a file sharing service used to share files over Wi-Fi and Bluetooth. AirDrop uses Bluetooth to create a peer-to-peer Wi-Fi network between the devices [27]. The files are sent encrypted, and each device creates a firewall around the connection, making it a secure, quick, and reliable file sharing option. AirDrop requires participants to be in Bluetooth range of each other.

3.2 Xcode and Swift

Xcode is an integrated development environment (IDE) created by Apple for developing iOS software. It includes the necessary development tools needed to build iOS applications, and only runs on Apple's Macs [28].

Swift is Apple's programming language to build iOS applications and is used in Xcode. It is a general-purpose, multi-paradigm, compiled programming language, and is designed to work with the large body of existing Objective-C code written for iOS applications prior to the introduction of Swift in 2014. Pre-defined functions in Swift can be used for cryptographic processes required in DCApp, such as signing and encryption. Also, Swift is designed to work with Apple's Cocoa Touch framework [29]. This framework provides an abstraction layer of iOS. The most noteworthy features of Cocoa Touch that are relevant for this project are the gesture recognizers[3] [30].

[3] Gesture recognizer: logic for handling touch events on a device, e.g. iPhone.

3.2.1 Podfile

A file referred to as a Podfile is a file included in an Xcode project to specify dependencies. This file allows the use of services such as Firebase's, as will be explained in Section 3.3.

3.2.2 Uniform Type Identifier

In order to create a form of data that is connected to DCApp, a uniform type identifier (UTI) needs to be defined. A UTI provides a consistent identifier for data that the application accepts. In Xcode, unique UTI's can be defined such that data with this identifier is directed and handled by the application when received on a device by AirDrop. If more applications handle this type of data, the user will get a choice on what application to handle the data.

3.3 Firebase

The development of an application requires several services, and one provider of many relevant services is Firebase. Firebase is a platform by Google used for application development. It provides tools to handle services that developers otherwise have to create themselves. From these characteristics, Firebase can be considered a Backend-as-a-Service [31]. The services that have been implemented into this project are authentication and real-time database.

3.3.1 Dependencies

In order to add the services, the appropriate dependencies have to be added to the Xcode project. This is done by adding them to the project's Podfile, and then running **`pod install`** from the command line. In this project the following dependencies are added to the Podfile:

```
pod 'Firebase/Core'
pod 'Firebase/Auth'
pod 'Firebase/Database'
pod 'Firebase/Firestore'
```

LISTING 3.1: Podfile configuration.

3.3.2 Authentication

Authentication in applications is used to associate a user with an identity, and information needs to be handled securely. It is also used to store data about a user. Firebase Authentication provides service for authenticating users in applications. There are several methods to authenticate users, including via Google, Facebook, email and password, and others. Only authentication through email and password will be used to log users into DCApp.

3.3.3 Cloud Firestore

Most applications require some form of data storage and the feature of synchronized data. DCApp needs to store information about users, as well as other data such as requests for grade transcripts. This data should be synchronized, so that updates on requests are available to all relevant users in real time.

The Firebase Cloud Firestore Database is a flexible, scalable NoSQL cloud-hosted database [32]. Firestore is created for mobile, web, and server development, and is constructed to keep data in sync across client applications. This database stores data in documents, which contains fields mapping to values. Documents are stored in containers called collections, which can be used to organize data and build queries[4]. Each document contains fields, commonly known as attributes when dealing with databases. An illustration of how the Firestore Database is structured is shown in Figure 3.3. A database will usually contain more than one collection.

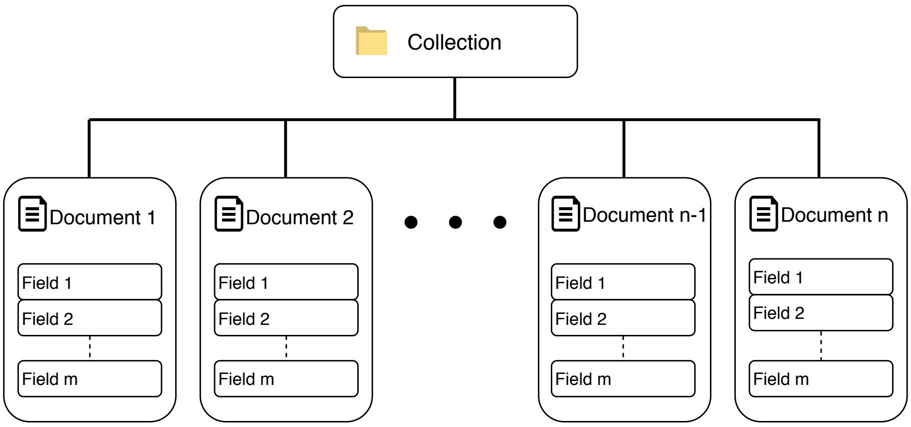


FIGURE 3.3: Structure of Firestore database.

[4] Query: the action of retrieving data from a database [33]

Firestore offers access management and authentication to the data in the database through security rules [34]. These rules are set in the Firebase console and restrict who can read and write to the documents. In this context, reading means retrieving information about documents through queries. Writing means creating or modifying documents.

Chapter 4

Design

This chapter will explain the design used to develop DCApp, and how the technologies explained in the previous chapters have been utilized and implemented. First, a high-level description of the application will be provided. Then a deeper low-level explanation will follow, where the choices for cryptographic method and private key handling are essential. The features of the application will be outlined, as well as which features have been partially implemented, and which have not been implemented, and the reason.

4.1 Application Overview

As stated in Section 1.4, the goal for DCApp is to create a prototype application of a document verification system that utilizes biometric identification. The university is the signer (administrator), the student is the document owner, and any third party is the verifier. This section will present an overview of how the student and university interact in DCApp. These actions are summarized in Figure 4.1.

First, a student can sign up to DCApp from any location, but must meet an administrator face-to-face to complete the signup process. The reason for this is that the university must verify the student's identity with an ID-card, and observe a successful Touch ID or Face ID on the device. During this encounter, the student will be verified by the university.

Secondly, the student must be able to request documents from the university. For DCApp, one option will be provided to the student, which is to request the most recent grade transcript. The university will provide the student with the document, and the document must be verifiable.

Lastly, the student can now send the document to whom they wish, and the verifier can verify the document.

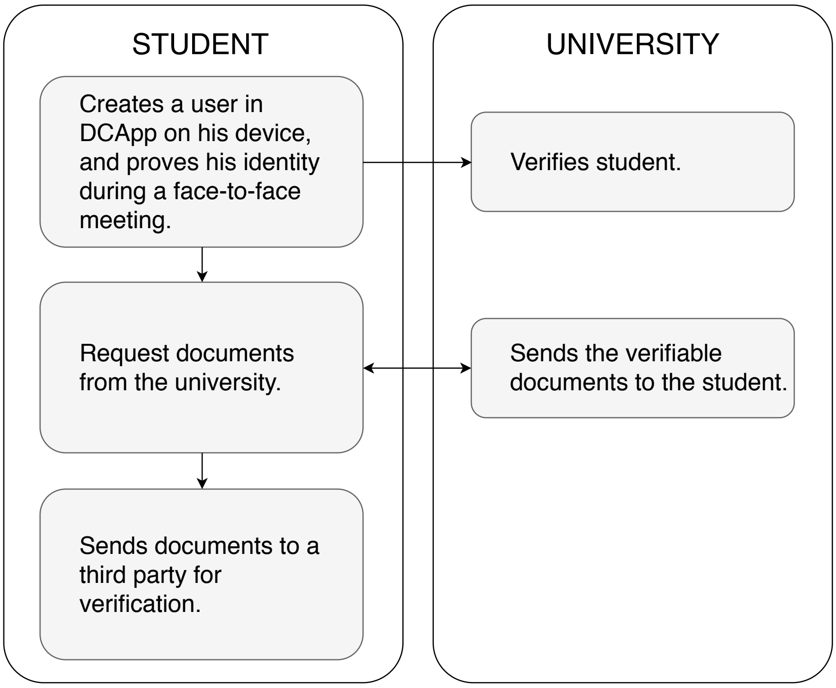

STUDENT
UNIVERSITY
Creates a user in DCApp on his device, and proves his identity during a face-to-face meeting.
Verifies student.
Request documents from the university.
Sends the verifiable documents to the student.
Sends documents to a third party for verification.

FIGURE 4.1: The process in DCApp for student sign up to document verification.

In order to prepare DCApp for use by students, there need to be administrators who can verify students and send documents. To ensure this, there is one main administrator, who has all of the responsibilities as a regular administrator, as well as creating new administrators. When new administrators are created, they need to be verified by the main administrator in the same manner as students are verified by administrators.

The rest of this chapter will go into more detail on how the processes explained in this section will be designed in order to create a secure document verification system.

4.2 Cryptographic Methods

As mentioned in Section 2.2, public key encryption will be used as the cryptographic method in this project. There are several possible approaches when constructing a secure cryptographic scheme where public key encryption is applied. The underlying cryptographic scheme implemented in DCApp is considered one of the major objectives of this thesis. It is crucial to choose a secure cryptographic scheme, which is compatible with iOS technology and supports other requirements in this thesis. Some of the other requirements are that the cryptographic keys can be moved to new devices, and that they are only accessible through the use of biometric identification. How these requirements are handled is described in Section 4.3.

To construct a secure cryptographic scheme for DCApp, two schemes were considered and evaluated: zero-knowledge proof and a scheme based on several signing processes. For both of these schemes, it is necessary for the signer to have obtained a certificate from a CA that can be used to prove its identity and provide its public key to any verifier who needs it. This step is not implemented in DCApp, but the schemes are explained with this as a prerequisite, as they are constructed for a real world application.

4.2.1 Zero-Knowledge Proof

The zero-knowledge proof algorithm is a method where one party can prove to another party that they have some information, without actually conveying the information. This algorithm could be used to prove the identity of the student to a third party who wishes to verify a document, by proving that the student is the owner of the document. It requires that signers have a pair of keys used for signing, but not the document owner. To provide further security against unwanted parties reading the document, the document owner would need a key pair for encrypting the data upon storage. The algorithm is explained step by step below, based on [35]:

- The student generates a one time value $y = g^x \bmod p$, and transfers y to the university upon sign up. x is the student's secret value, p is a large prime and g is a generator.

- Upon a document request from a student, the university signs the document, creating a hash from the document and y. This is sent to the student along with the document.

- The student computes a value $C = g^r \bmod p$, where r is a random value. He then sends the document, the university's signature, y, and C to a third party for verification.

- The third party checks that the document has not been modified by creating a hash from the document and y, and checking that it is a match to the university's signature. This verification process is illustrated in Figure 2.4.

- The third party will then verify that the student is whom he claims to be by making the student calculate $(x + r) \bmod (p - 1)$, or returning the value r. In the first case the student verifies $(C \cdot y) \bmod p \equiv g^{(x+r) \bmod (p-1)} \bmod p$. In the second case he verifies $C \equiv g^r \bmod p$. This proves that the student is the owner of the original value, x. The verifier must also verify the university's signature by obtaining the certificate issued by a CA, and the CA's public key. This process is explained in detail in Section 2.3. If all processes conclude in a valid verification, the verifier can trust that the student is who he says he is, and that the document he has provided is authentic.

This method provides sufficient security but does have some flaws. First, the student needs to send y every time he wants to request a new document, which could be considered unnecessary. Secondly, the verifier needs to participate in further communication with the student to verify that the student knows the value of x. For even further security, the last verification step would be repeated several times, as this increases the credibility of the student. It is possible for an attacker to guess the value of r or $(x + r)$. Even if there is a very low probability for an attacker to guess these values, the process should still be performed several times to diminish this probability.

4.2.2 Scheme Based on Signatures

The following scheme involves three signing processes, given that there is one signer, but does not require the verifier to engage in further communication with the student after he has received the data. This scheme requires that all signers and document owners have their own key pair used for signing, encryption, and decryption. The following steps describe this scheme:

- When the student first signs up, he must verify his identity at the university, and in the process share a copy of his public key, $K_{pu,s}$, with the university. From now on, when he requests documents, he does not have to send any more data, because the university stores a copy of his public key. The university signs the copy of the student's public key with their private key, which will be used later in a verifying process. This signature will be $\sigma_1 = Sign(K_{pr,uni}, Hash(K_{pu,s}))$, and only needs to be produced once.

- When a student requests a document, the university signs the hash of this document with their private key, $\sigma_d = Sign(K_{pr,uni}, Hash(Document))$. The document, σ_1, and σ_d are sent to the student encrypted with the student's public key.

- The data is stored in the student's iCloud Drive, encrypted with the student's public key. This ensures that the document can only be decrypted from DCApp, with the student's private key.

- When the student wishes to share the document with a third party, he signs σ_d with his private key to generate a new signature, $\sigma_s = Sign(K_{pr,s}, Hash(\sigma_d))$.

The student sends the document, $K_{pu,s}$, σ_1, σ_d, and σ_s to a third party.

- The third party gets the university's public key from a certificate issued by a CA directly from the university. He then verifies the document, student, and university by verifying the digital signatures. The verification process of a signature is shown in Figure 2.4. If all of the signatures are verified, the document, student, and university are verified.

This scheme is the preferred cryptographic method for this project because it uses established cryptographic techniques, and limits communication between the student and verifier to a minimum.

4.3 Handling the Private Key

For this application, the private key is crucial to create a reliable result. The private key belonging to both administrators and students are used to sign and decrypt documents, and thus, it must only be known to its owner. In iOS, there are two main ways to handle private keys, either in the keychain or Secure Enclave. In both of these methods, biometrics can be integrated as a security step to use the private key. Information on these two methods has been obtained from articles [19] and [23], published by Apple.

The Secure Enclave provides extremely high security, as the application never handles the key directly, making it unlikely for it to get compromised. Instead, the key is created, stored, and used to perform operations inside the Secure Enclave when correct Face ID or Touch ID is presented, and only the output of these operations is handed to the application. Although this creates an extra layer of security, it also adds a significant restriction to the system. It limits the user to use one specific device, as the private key is locked to this particular device. If the user was to lose or upgrade their device, they would have to generate a new key pair, and also meet in person with the relevant institute to identify themselves and share a copy of their new public key.

When storing the private key in the keychain, there is a small vulnerability because the key must be briefly copied as a plain-text version into the memory system. This presents a small possibility for the private key to be compromised, meaning that an attacker could act on behalf of the owner of the private key. However, the keychain is still considered to be very secure and provides a much more user friendly experience than the Secure Enclave option. When biometric authentication is used, the Secure Enclave is still a factor in the process of obtaining the keychain item. The Secure Enclave carries out the authentication, and gates access to the keychain item upon a pass result. Using the keychain also provides the possibility to use the application on multiple devices. Hence, this is the key handling method that is implemented into DCApp.

4.3.1 Access Control on Keys

In order to provide further security to DCApp, the elliptic curve key pair generated from the framework can have restricted accessibility. The access control is used to set restrictions to the keys along two dimensions: accessibility and authentication. This allows keychain items to be protected with biometric authentication [23]. It also allows the keys to migrate to other devices owned by the same user, by setting the accessibility as `kSecAttrAccessibleWhenUnlocked` [36]. In addition to this security feature, keys stored in the keychain from DCApp are only accessible through DCApp. This is due to the private access group protected by the unique app ID[1] belonging to this application.

4.4 User Handling and Authentication

DCApp requires all users to have a user associated with the application. This is to connect the user with information in the database. All actions regarding user authentication are handled by Firebase Authentication. The services in Firebase Authentication provide support for creating new users and logging in existing users. For simplicity reasons, only the authentication through email and password is implemented in DCApp, as mentioned earlier. Firebase user objects have a set of properties, including unique user ID (UID) and email [37]. To include more properties, also referred to as attributes, to the users in DCApp, objects in the database are created and connected to the Firebase users. The objects are connected to the users by setting the document ID equal to the UID of the user.

4.5 Firestore Database Design

The database is used to store student users, administrative users, verified students, and requests. By querying the database, users can obtain relevant data. Every document (object) is assigned a unique document ID, which is used to create references to all objects in the database. The structure of the database is shown in Table 4.1.

[1] An ID that uniquely identifies every application.

Collection	**Documents**	**Attributes**	
students	Document ID/ User ID	isVerified: studentId: name:	Bool Int String
admins	Document ID/ User ID	isMain: isVerified: updatedPwd: role: signedPk:	Bool Bool Bool String String
verifiedStudents	Document ID	studentId: publicKey: signedPK:	Int String String
requests	Document ID	studentId: status:	Int String

TABLE 4.1: Structure of database in DCApp.

The database contains a set of rules, restricting access to particular objects by unauthorized users, and will be explained further in Section 5.2.1.

4.6 Student Process

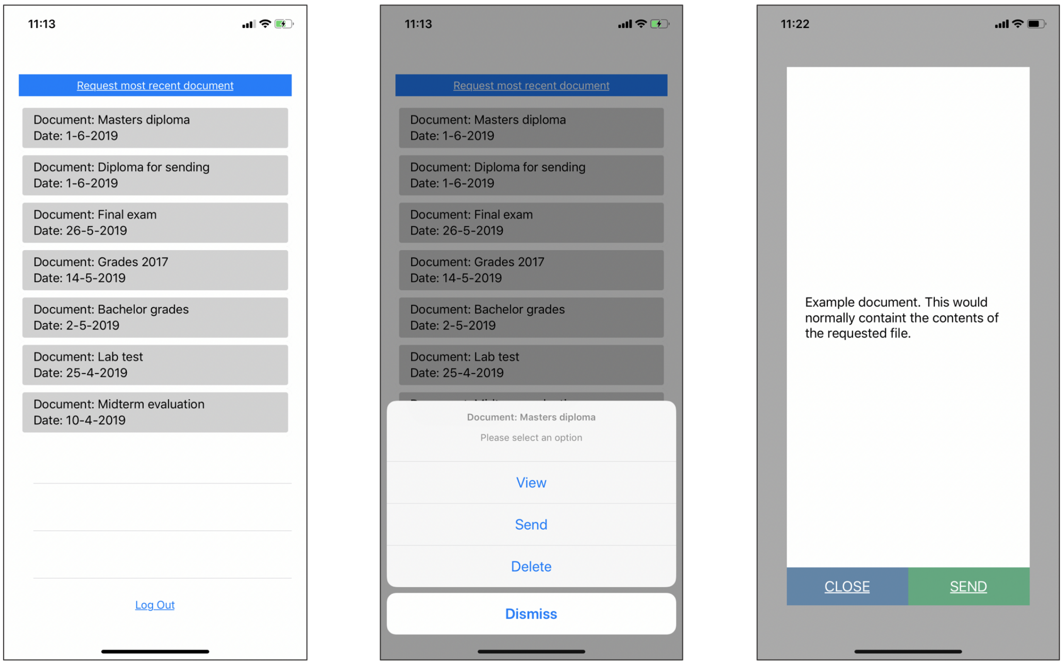
From the specifications of the system design explained so far in this chapter, we can give a more detailed description of the student process in DCApp.

As explained in Section 4.1, the student needs to meet an administrator in person to prove his identity. During this encounter, the student will be verified by the university, and the university will obtain a copy of the student's public key. A copy of the public key is shared by using the secure connection in AirDrop, and is given a unique UTI to the application, **.spk**. This ensures that the data will be directed and handled by DCApp on the administrator's device, and not any other program.

Secondly, the student should be able to request documents from the university. In order to implement the logic for this, a collection named **requests** in the database is created. When a student requests a document, he creates a **request** object in the database. Objects in this collection will contain the student number of the student who sent the request and the status of the request. The status will be set as pending on creation and can be changed later by an administrator. Since Firestore is a real-time database, the requests will be updated in real time across all of the administrator interfaces.

The student receives the requested document, along with two signatures, σ_1 and σ_d. This data is encrypted with the student's public key. The encrypted data is saved to the student's iCloud Drive, and can only be decrypted in DCApp with his private key, after a valid biometric identification. This is due to the access control on the cryptographic keys. If biometric identification is not successful after two attempts, the student will be logged out of DCApp. To try again he must log back in, and is given another two tries. In other applications, a passcode can usually replace biometric identification, this is not the case for DCApp however, as biometric identification is one of the major requirements for the system.

The student interface displays a list of received documents, where each element is a reference to the location of the data in iCloud Drive. When tapping a document reference the student is given a list of options, and he must provide a biometric identification in order to execute the actions. Figure 4.2 shows the design of the interface.



Document: Masters diploma
Date: 1-6-2019

Document: Diploma for sending
Date: 1-6-2019

Document: Final exam
Date: 26-5-2019

Document: Grades 2017
Date: 14-5-2019

Document: Bachelor grades
Date: 2-5-2019

Document: Lab test
Date: 25-4-2019

Document: Midterm evaluation
Date: 10-4-2019

Document: Masters diploma
Date: 1-6-2019

Document: Diploma for sending
Date: 1-6-2019

Document: Final exam
Date: 26-5-2019

Document: Grades 2017
Date: 14-5-2019

Document: Bachelor grades
Date: 2-5-2019

Document: Lab test
Date: 25-4-2019

Document: Masters diploma
Please select an option

Example document. This would normally containt the contents of the requested file.

FIGURE 4.2: The student interface contains a list of all encrypted documents in their iCloud Drive. Upon tapping a document the student can perform actions.

Lastly, the student needs to be able to sign σ_d, creating σ_s, and send the desired document to a third party. To initiate this process, the student must provide a valid biometric identification to gain access to the private key needed to create a digital signature. In a globally distributed system, the data would be sent to a verifier by sending the data via email, for example. This is a prototype application, and thus, the verification will be performed on the student's device. A more detailed explanation of the verification process will be given in Section 4.9.

4.7 Administrator Process

The main purpose of the university is to verify the identity of students on the first encounter, and to sign documents. Before a student can request any documents, the student must meet an administrative user to prove his identity, and deliver a copy of the public key through AirDrop. A copy of the public key is only released to the university after the administrative user observes an authentic Face ID or Touch ID on the student's device, paired with a physical student ID card. The university immediately signs the copy of the student's public key, creating σ_1, and stores it as a **`verifiedStudents`** object in the database, along with the corresponding student id. The **`verifiedStudents`** collection contains all verified students, and is accessible by all administrative users.

Upon transcript requests from a student, the university has three options:

- Accept the request. This option could be used in larger systems where some administrators have the role of handling requests and forwarding or assigning them to the correct administrators to sign and send the documents. However, this logic has not been prioritized in DCApp.

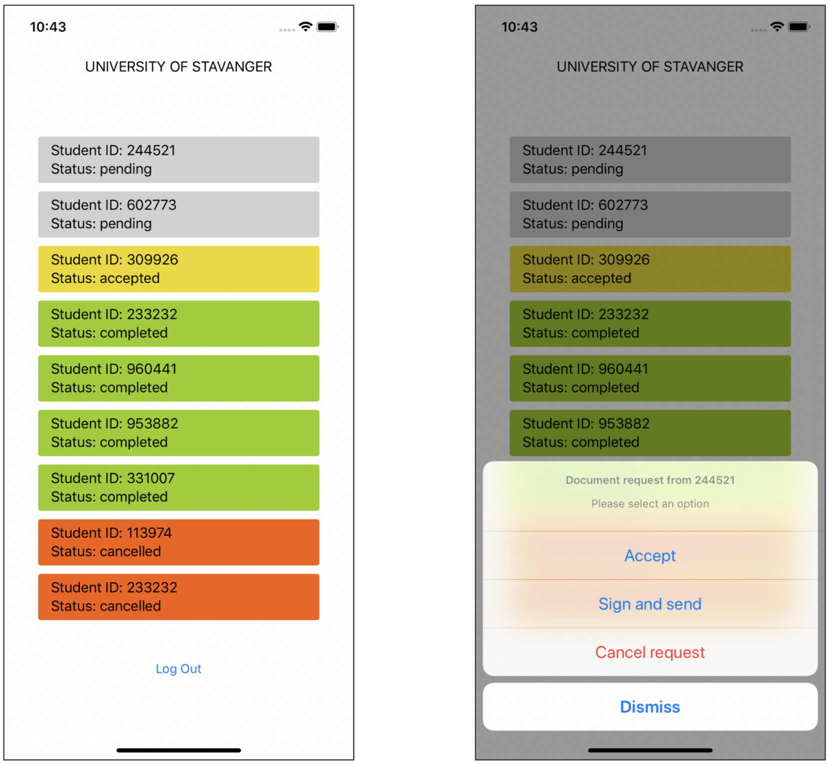
- Cancel the request. In DCApp the request is marked as cancelled, but no logic has been implemented to further inform the student of this, as it is not a significant part of the fundamentals of the system.

- Sign and send the document. When an administrator is ready to sign and send a document, the administrator will find the requested document in their database, sign it, creating σ_d, and send the data (document, σ_1 and σ_d) to the student. In DCApp a sample text document is used as the document.

The university's administrative users must provide an authentic Face ID or Touch ID in order to perform such operations. The same logic for handling failed identification attempts is implemented for administrative users as for students, and the user is signed out if a valid identification is not provided within two attempts.

4.7.1 Administrator Actions

For administrative users in DCApp, a user-friendly interface shown in Figure 4.3 has been designed and implemented. When a request is tapped from the list, the administrator can perform actions depending on the status of the request. If the request is pending, for example, the administrator can accept it, sign and send it, or cancel the request. For any of the actions to be completed, the administrator must provide a valid biometric identification.



Student ID: 244521
Status: pending

Student ID: 602773
Status: pending

Student ID: 309926
Status: accepted

Student ID: 233232
Status: completed

Student ID: 960441
Status: completed

Student ID: 953882
Status: completed

Student ID: 331007
Status: completed

Student ID: 113974
Status: cancelled

Student ID: 233232
Status: cancelled

Student ID: 244521
Status: pending

Student ID: 602773
Status: pending

Student ID: 309926
Status: accepted

Student ID: 233232
Status: completed

Student ID: 960441
Status: completed

Student ID: 953882
Status: completed

Document request from 244521

Please select an option

FIGURE 4.3: The administrative interface contains a list of all request, and upon tapping a requests the administrator can choose to perform an action.

4.7.2 Creating New Administrators

This application will have the possibility to create more administrative users, also known as signers. This is a feature that is useful in order to create a more realistic experience. In reality, there is usually more than one person who has to sign a diploma before it is granted to the student. To achieve this feature, the application has one main administrator who has the possibility of creating new users with administrator privileges, in this case signing documents and verifying students. In DCApp, documents need only be signed by one administrative user before sending it to the student. The process of creating a new administrator is as follows:

1. The main administrative user creates a user by entering the new administrator's email address and a simple password that will be updated to a more secure password at a later time.

2. On the first login for the new administrator, he or she will be asked to update the password to a more secure password.

3. The new administrator has to meet the main administrator in person to identify themselves, authenticate themselves with biometrics on their

device, and deliver a copy of their public key for signing through AirDrop. The new administrator is now a verified signer in the system, and the signed key is added to the administrator object in the database.

4. The main administrator gives the new administrator the signed key in return, which will be used for verification by any third party. It verifies that the signer has been verified by the institute.

These new administrators are not able to create new administrators; this is only possible for the main administrator.

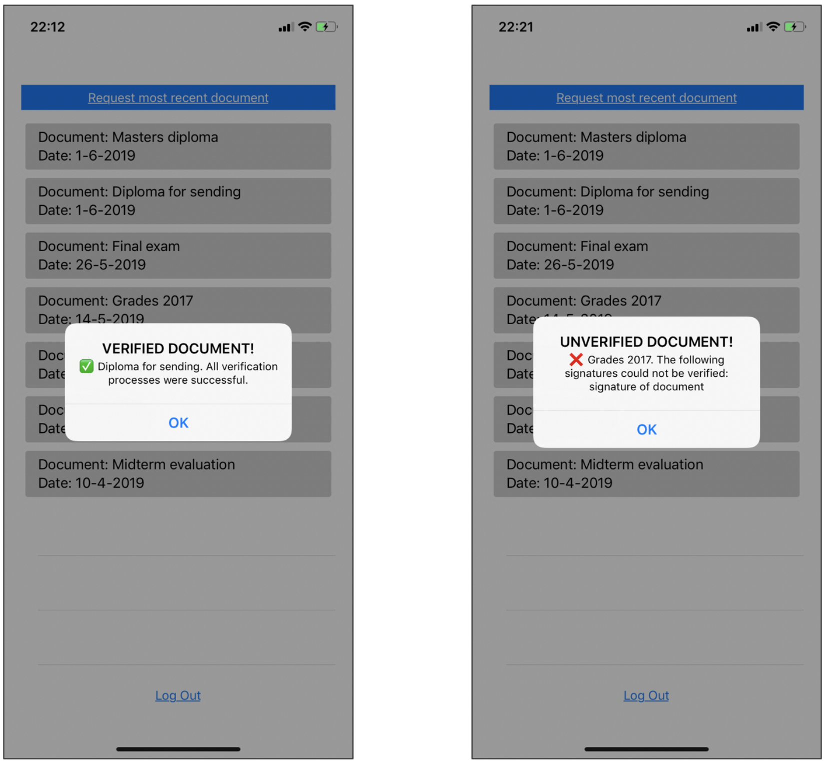
4.8 Generating Key Pair

Both the students' and administrators' key pairs are generated at the face-to-face meeting before they are verified by an administrator or main administrator, respectively. As mentioned earlier, a copy of the public key is delivered to the administrator who is verifying the new user before the user is considered verified. The key pair is generated when the new user provides a valid biometric identification on their device. This biometric identification is required before sending a copy of the public key to the administrator who is verifying the user. If the AirDrop of the copy of the public key is not successful for some reason, or the student is not successfully verified, a new key pair will be generated the next time the user tries to transfer the copy of the public key for verification, overwriting the old key pair.

4.9 Third Party

The third party, also referred to as the verifier, will receive data for verification by the document owner. In order to verify the document, the verifier must verify the signatures σ_1, σ_d and σ_s. The student will send these signatures along with his public key and the document. To verify σ_1 and σ_d, the verifier must obtain a certificate issued by a CA directly from the university, which contains a copy of the signer's public key. The software would only need to be able to perform the verification function described in Section 5.7.

As DCApp is a prototype application where the goal is to prove the concept of a document verification system, a student logged into the application will have the possibility to verify the document. A copy of the public key of the university will be sent along with the document to the student, since a PKI is not implemented. This should never be done in a real world system. The application will perform the necessary verification steps, verifying the signatures σ_1, σ_d and σ_s. Two screenshots, one for a verified document and one for an unverified document, are shown in Figure 4.4.



22:12

Request most recent document

Document: Masters diploma
Date: 1-6-2019

Document: Diploma for sending
Date: 1-6-2019

Document: Final exam
Date: 26-5-2019

Document: Grades 2017
Date: 14-5-2019

VERIFIED DOCUMENT!
✅ Diploma for sending. All verification processes were successful.
OK

Document: Midterm evaluation
Date: 10-4-2019

Log Out

22:21

Request most recent document

Document: Masters diploma
Date: 1-6-2019

Document: Diploma for sending
Date: 1-6-2019

Document: Final exam
Date: 26-5-2019

Document: Grades 2017

UNVERIFIED DOCUMENT!
❌ Grades 2017. The following signatures could not be verified: signature of document
OK

Document: Midterm evaluation
Date: 10-4-2019

Log Out

FIGURE 4.4: Result of verification process in student interface.

In a larger and more realistic system, there would be more than three computations. As explained in Section 4.7.2, it is likely that an institution will assign the signing role to more than one user. In this case, the signers would all have to be verified by one main administrative user, who would also sign the copy of their public key. If more than one signer has the role of signing the same document, the verifier would have to perform more computation in order to complete the verification process. Copies of these public keys would be retrieved directly from the institute, and the verifier would have to verify that all of the signers' public keys have been signed by the main administrative user in the institution. The main administrator's public key would be retrieved from the certificate issued by a CA, which would also confirm the identity of the institution. Thus, the number of verification computations needed to be performed by the verifier is three times the number of administrative signers, plus two.[2] The cost of verification computations is considered to be cheap, and the effort required is not very significant.

[2] For every signer of a document, the verifier must verify σ_d, σ_s, and the public key of the signer. In addition, the verifier must verify σ_1 and the certificate from the main administrator once.

Chapter 5

Implementation

This chapter will present the implementation of DCApp in Swift. The implementation is based on the design described in Chapter 4, and uses some of the concepts and technologies introduced in Chapters 2 and 3.

5.1 Class Structure

Classes are building blocks for a larger system [38]. All functionality of DCApp is performed within different classes, which all have different purposes. The class structure will be presented in a class diagram in the Unified Modeling Language (UML) shown in Figure 5.1. This figure shows the most important classes in DCApp, as well as what functions and variables they contain.

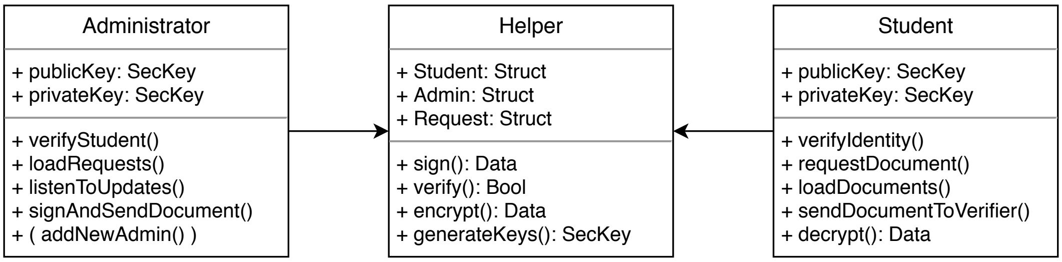


FIGURE 5.1: UML class diagram representing the most important classes, functions and variables.

The UML class diagram above is a simplified representation of the classes used for the implementation of DCApp, but includes the most essential parts. For a signed in user, they either have an administrator or student interface, and can reach certain functions depending on their role. The diagram shows the most important functions and variables that are reached from each interface. The **`Helper`** class contains functions and structs that are common between students and administrators. A struct acts as a data type and ensures that all objects of that type have the required information. Some of the functions in Figure 5.1 are explained in more detail in this chapter.

5.2 User Authentication

Firebase Authentication provides the backend services to authenticate users in DCApp. It handles both the sign up for new users and signing in for existing users. `Auth.auth()` is the authentication service provided by Firebase, and it includes functions such as `createUser` and `signIn`. Both of these functions require the specification of what type of credentials that are used for authentication, which is email and password in DCApp. Also, on sign up for students the student number, name, and whether or not the student has been verified is stored, which is always set to `false` upon sign up. A struct, **`Student`**, is defined to ensure that all students have these attributes. The struct can be seen in Listing 5.1.

```swift
struct Student {
    var name: String
    var id: String
    var verified: Bool
}
```

LISTING 5.1: The struct for student objects.

The listing below shows the code for creating a new user. Only students can sign up on their own, as regular administrative users are created by the main administrative user.

```swift
Auth.auth().createUser(withEmail: email, password: password,
 ↪  completion: { (user, error) in
    if error == nil {
        let data = Helper.Student(name: name, id: studentNumber,
         ↪  verified: false)
        self.saveUserToDB(data: data, uid: (user?.user.uid)!)
    } else {
        //handle error
    }
})
```

LISTING 5.2: Function that handles the sign-up of new students.

The function called in line 4 in Listing 5.2, `saveUserToDB` saves the user information as a new **`student`** object in the database. Every Firebase user has a unique user ID (UID), which is used as the document ID in the database.

Administrative users will have other attributes such as role, and if they are the main admin or not. Thus, an **`Admin`** struct is used for administrative users.

Upon a sign in to the application, the code in Listing 5.3 is called. This code checks if the credentials match a user object, and sequentially if the user is a student or administrative user by checking the UID to the objects in the database. The user will be directed to the appropriate interface depending on their role.

```swift
Auth.auth().signIn(withEmail: email, password: password) { (user, error) in
    if let error = error {
        //handle error
    } else if user != nil {
        let currentUser = Auth.auth().currentUser
        let db = Firestore.firestore()
        var ref = db.collection("users").document(currentUserUID)
        //if currentUser is in the "users" collection, it is a
        //student. Perform actions to display student interface.

        //else, check is the currentUser is an admin user.
        ref = db.collection("admins").document(currentUserUID)
        //perform actions to display admin interface.
    }
}
```

LISTING 5.3: Function that handles sign in.

5.2.1 Database Rules

In order to create a secure system, the database needs a set of rules which define what users have read and write permissions to certain objects. Firestore allows rules to be written for the database as a whole, and specifically for collections and documents.

The most essential restriction in the database is that a request has to come from an existing, authorized user. Students will have access to read the data connected to their user object, and can write to the **`requests`** collection. All rules are written in a syntax similar to JavaScript in a JSON structure, and the language is based on the Common Expression Language (CEL) [39]. This syntax is shown in Listing 5.4.

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /students/{student} {
    allow read, write: if request.auth.uid == student ||
     ↪  exists(/databases/$(database)/documents/admins/
     ↪  $(request.auth.uid))
    }
  }
}
```

LISTING 5.4: Database rules for students.

Administrative users will have access to read and write to **`student`** objects, but only read their own object. The main administrative user will have access to read and write to any object in the database, including other administrative users. This is the user that can create new administrative users and must verify their identity.

Without proper restrictions to the database, unauthorized users could create and delete users, create new administrative users, or alter user objects.

5.3 Biometrics

For any action requiring the use of a private key, the user has to provide authentic biometric identification. This includes all signing processes by both the university and student. Also, both regular administrative users and students need to provide an authentic biometric identification upon sign up. The code in Listing 5.5 shows how to use the biometrics API[1] from Apple. First, the code checks if the device is compatible with biometrics. If the device supports biometrics, an identification attempt is required, and an action will be performed depending on whether the identification attempt fails or succeeds.

[1] API: Application Programming (or program) Interface: a set of protocols used by programmers to create applications for a specific operating system or to interface between the different modules of an application [40].

```
if context.canEvaluatePolicy(.deviceOwnerAuthentication, error: &error) {
    let reason = "Authenticate"
    context.evaluatePolicy(.deviceOwnerAuthentication, localizedReason: reason ) { success, error in
        if success {
            //perform action for successful identification
        } else {
            //perform action for failed biometrics attempt
        }
    }
} else {
    //device does not have Face ID or Touch ID
}
```

LISTING 5.5: Code that handles biometric identification upon performing actions in DCApp.

When biometric identification is required, the user has a few attempts to present an authentic identification, and if the identification fails, the user is logged out of the application. No actions can be performed, and no information can be retrieved, if the user cannot identify themselves through biometrics. This means that even if someone knows a user's password, they cannot gain access to any keys, or perform actions on behalf of someone else.

5.4 Communication Between Participants

Communication between participants in DCApp is done mainly via AirDrop. When any user is created, apart from the main administrative user, they need to provide a copy of their public key. This key is delivered via AirDrop, and a similar solution would be necessary for real life applications as well, because a physical first time meeting is required.

The requesting of documents by students is done by creating **`request`** objects in the database. These objects are created by a student when he chooses this alternative in the student interface, but cannot be read or written to by students. Only administrators can read and write to **`request`** objects. The administrators receive immediate changes to the **`request`** collection by implementing a Snapshot Listener that listens to the collection. The Snapshot Listener is shown in Listing 5.6

below. If a new request is added, modified by another administrator, or deleted, all administrators get the instant update.

```
func listenToUpdates() {
    let db = Firestore.firestore()
    db.collection("requests").addSnapshotListener { querySnapshot,
    ↪  error in
        guard let snapshot = querySnapshot else {
            //throw error
        } snapshot.documentChanges.forEach { diff in
            if (diff.type == .added) {
                //add new request and reload
            }
            if (diff.type == .modified) {
                //modify request and reload
            }
            if (diff.type == .removed) {
                //delete request and reload
            }
        }
    }
}
```

LISTING 5.6: Function called upon admin sign in to listen to real time request updates.

The last communication option that is implemented in DCApp is the transfer of documents from the administrator to the student. In a real world application, the solution would have to work with global participants, but due to restrictions that are mentioned in Section 6.1.2, the document transfer is performed via AirDrop in DCApp as a demonstration. When a document is transferred to a student in DCApp it is first encrypted with the copy of the student's public key to ensure that it can only be decrypted by the intended student. The student number in the document request is used to find the student in the **`verifiedStudents`** collection in the database, which contains all students that have been verified by the university. This is done by performing queries to the database, and is shown in Listing 5.7. After the data has been encrypted by the signer, it is sent to the student.

```
db.collection("verifiedStudents").whereField("studentId", isEqualTo:
 ↪  id).getDocuments() {
    (querySnapshot, err) in
    if let err = err {
        //throw error
    } else {
        for document in querySnapshot.documents {
            //take public key attribute from object, and use it to
             ↪  encrypt the data
        }
    }
}
```

LISTING 5.7: Database query to obtain a student's public key.

Solutions for global participants will be discussed in Section 6.1.2.

5.5 Signing Process

Both the university and students compute some digital signatures. To create the digital signatures the same helper function, shown in Listing 5.8, is called.

```
public func sign(_ digest: Data) throws -> Data {
    var error : Unmanaged<CFError>?
    let result = SecKeyCreateSignature(privateKey,
     ↪  .ecdsaSignatureMessageX962SHA256, digest as CFData, &error)
    guard let signature = result else {
        //throw error
    }
    return signature as Data
}
```

LISTING 5.8: Signing function used for all signing processes.

The signing process is the same for all rounds of signing, which is why the same helper function can be called for any signing process by any party. To create a digital signature three parameters are needed: a private key, data to be signed, and hashing algorithm. These are the three variables that are taken as parameters in the build in function, `SecKeyCreateSignature`. This function creates the cryptographic signature for a block of data using a private key and a specified algorithm (`.ecdsaSignatureMessageX962SHA256` is used with ECC and SHA256) [41]. In DCApp the hashing algorithm chosen is SHA256, which is considered to be secure.

In DCApp the university signs the copy of the public key of the student once, and signs all documents that are to be granted to the student. For every document the student has received, he must sign the signature of the document before sending it to any third party. This is to verify that it is the student who is the sender of the document.

5.6 Storing and Retrieving Documents

After the student receives a document it is stored in the student's personal iCloud Drive. For this implementation it is expected that the student has enabled iCloud, otherwise, the document will be disregarded. If iCloud is enabled, the application stores the document in a directory called DCAppDir. This directory is created the first time a document is saved to iCloud Drive from the application, DCApp. The pseudocode for saving data to iCloud Drive is shown in Listing 5.9 below.

```
func saveToiCloud(data: String) {
    if iCloud enabled then
        if DCAppDir does not exist then
            create DCAppDir directory
        create empty file
        insert the data into the file
        copy file to DCAppDir in iCloud Drive
}
```

LISTING 5.9: Pseudocode for saving a document to iCloud Drive.

After a file has been saved to iCloud Drive, it will be available on any device where the user has their iCloud enabled. However, the document can only be encrypted by DCApp, due to its protection from the encryption. The private key that decrypts the data can only be used within DCApp.

The documents are displayed in the student interface, but are only file references to the documents in iCloud Drive. If the student wishes to see the contents of a document, it must be retrieved from iCloud Drive and decrypted. Pseudocode for accessing the file references and downloading documents are shown in Listings 5.10 and 5.11, respectively.

```
func findFiles() {
    if iCloud Drive enabled on device and DCAppDir exists then
        retrieve contents of directory
        create variables with reference to files
        display variables in student interface
}
```

LISTING 5.10: Pseudocode for importing the file paths of iCloud Drive documents.

```
func openFile(fileReference: String) {
    find and retrieve file at fileReference path
    decrypt file with private key - need valid biometric
     ↪  identification
    display file to user
}
```

LISTING 5.11: Pseudocode for displaying a document from iCloud Drive.

5.7 Verification Process

The verification process consists of performing several hashing and verifying operations. A diagram outlining the verification process has been shown in Figure 2.4. In order to verify that the document has not been modified after it was signed, and that it belongs to the correct student, three verification processes need to be computed. The code used to validate a signature is provided in Listing 5.12.

```
1 func verify(signature: Data, digest: Data, publicKey: SecKey) throws
   ↪ -> Bool {
2     var error : Unmanaged<CFError>?
3     let valid = SecKeyVerifySignature(publicKey,
       ↪ .ecdsaSignatureMessageX962SHA256, digest as CFData,
       ↪ signature as CFData, &error)
4     guard valid == true else {
5         //throw error
6     }
7     return valid
8 }
```

LISTING 5.12: Function for verifying the signature of some data.

`SecKeyVerifySignature` in line 3 in the listing above is a function provided in Swift that verifies a cryptographic signature using a public key and a specified hashing algorithm, and returns a `Bool`[2] as return value [42]. It takes four parameters:

1. The signature it is trying to recreate.
2. Digest: the data that was hashed in the signing process (for example the document).
3. Public key. This is the corresponding public key to the private key that was used to create the signature in 1.
4. Hashing algorithm. It is important that this is the same algorithm used in the signing process. In this implementation, SHA256 is used.

The signature and digest are converted to type `CFData`, which is a mutable data type that is used in order to perform the verification computation [43].

The three verification computations needed in this sample application are:

- verify(σ_1, studentPublicKey, universityPublicKey)
- verify(σ_d, document, universityPublicKey)
- verify(σ_s, σ_d, studentPublicKey)

In a real world application, third parties would verify documents by retrieving the certificates containing the signers public key directly from the institutes. However,

[2] True or false value

in DCApp, a PKI has not been used, and the verification process is invoked by the document owner. The verification process takes place right after the student signs the document signature, σ_{d}, producing σ_{s}.

Chapter 6

Evaluation

In this chapter, the results, benefits, and risks involved with implementing the solution outlined in this thesis will be discussed. The use cases mentioned in Section 1.2 will also be used in the discussion.

The degree of validity in documents and authentic identification differ vastly between countries. Comparing all methods practiced by different countries will be cumbersome; hence, the systems in Norway will be the primary focus in this chapter.

6.1 Improvements

In this section, some of the shortcomings in this thesis will be discussed, and possible methods to solve these in a real world solution.

6.1.1 Importance of the Private Key

In order to build a secure system based on the technologies outlined in this thesis, the private key is crucial. A very secure method is, as mentioned earlier, to create and store the private key in the Secure Enclave. This method has a significant drawback in the context of this system, which is that the private key is locked to a single device. Although this would provide the most significant security for the private key, it is not very user-friendly. If this was the chosen implementation, the user would have to generate new key pairs every time a device was lost or replaced. This would also mean that they would have to meet in person to identify themselves and share their new public key with the relevant institute.

Instead, the private key should be encrypted and stored in the keychain, where it is stored and can be accessed on new devices that belong to the same user. This is the chosen method for the sample application as well, and is considered secure and user-friendly, as the user has to use biometrics in order to use their private key. The Secure Enclave carries out this authentication, and upon a pass result, it gates keychain item access [23]. Although the keychain does allow the private key to be imported on new devices, it has to be done from an encrypted iTunes backup [44]. This means that the user is required to perform some actions, namely to take regular, encrypted backups in iTunes. This is not a realistic expectation, and thus in order to implement the document verification system that has been explained, a more user-friendly version of the keychain would have to be designed.

The concept of such a keychain is similar to the existing keychain, but would also be used to synchronize cryptographic keys over iCloud, and not just passwords and credit cards. This could be a feature that will be available in the future, as two-factor authentication[1] provides a very high level of security to the encrypted keychain [46]. However, this would require a solution that is too advanced to research in this thesis, and is out of the scope of this project. Also, for a scalable application, there would need to be a solution that could work cross-platform, not only for iOS.

6.1.2 Sending and Storing Documents

The document storage is important in the document verification system explained throughout this thesis. Documents need to be easily accessible to the user, as well as securely stored. For DCApp, which is only intended for iOS, a natural solution is to store encrypted documents in the document owner's personal iCloud Drive. This allows the document to be synchronized across devices, and can only be decrypted in DCApp. The main advantage of using iCloud Drive as file storage is that it saves storage cost, and each document owner is responsible for having enough storage.

Document transfer is performed by using AirDrop in DCApp. This is the implemented method because document transfer is not the main focus in the concept of the document verification system, and DCApp is designed under a time constraint. It allows documents to be transferred between users, but only within Bluetooth range of each other.

Firebase is used for many of its backend services, but its Cloud Messaging service is not ideal for document transfer. This is because it does not allow messages with a payload greater than 4KB to be transmitted [47]. A solution to transfer larger payloads with Firebase is to use its storage service. However, this would require DCApp to be responsible for the document storage, which is one of the main reasons why iCloud Drive has been the implemented storage method.

[1] Two-factor authentication: an extra layer of security designed to ensure that only the account owner can access their account, even if someone knows the password. When signing in to a new device for the first time, two pieces of information is required— password and a six-digit verification code that's automatically displayed on an already trusted device. [45]

iCloud Drive presents the possibility of sharing documents with a shareable link, thus providing the possibility of transferring documents between globally distributed participants. The signer would encrypt the data with the document owner's public key, save the data to iCloud Drive, share it with the document owner by sending the shareable link (payload less than 4KB). The document owner could then copy it to their iCloud Drive, and finally the signer could delete their copy, freeing up storage. This is a solution which has not been implemented because it is not possible to retrieve the shareable link programmatically, meaning that the result would not be very user-friendly. Also, iCloud Drive could not be an option on a real world application as it is not compatible cross-platform. From this information, it can be deduced that iCloud Drive would not be the best storage option in a real world application. Instead, some other options will be considered.

Alternatives

The first alternative is a solution that would integrate well with DCApp, known as CloudKit [48]. CloudKit is a framework that acts as a Backend-as-a-Service, and is used to keep application related data up to date across devices. This solution would be suitable because it allows data to be uploaded to cloud storage and shared between users, and can use users personal iCloud Drive space. Thus, data can be shared globally. CloudKit offers a lot of the same services as Firebase, but mainly focuses on the iOS platform [49].

The second and most realistic option in a real world application, is to use a cloud-based solution that does not rely on a specific operating system. There are many options available, including Firebase's storage, AWS, and Azure. Cloud storage is considered to be secure, and there is extra security considering that the document is encrypted with the document owner's public key. The costs of these cloud storage options are not excessive, making it a suitable alternative for a real world application.

In the following sections, we will go into more detail into the use cases presented in Section 1.2, and how a document verification system will benefit these cases.

6.2 Degree Certificates

DCApp is an implementation of the document verification system, and the use case of the application is to verify degree certificates, also known as grade transcripts. According to [50], there are several cases every year in Norway where national and foreign applicants forge different types of diplomas, and it is also safe to assume that a large number of forgeries go unnoticed.

6.2.1 National Applicants

National students usually supply their university application with a Norwegian high school diploma. These diplomas are standard in Norwegian high schools, and for some students who completed high school in the year 2000 or later, the diplomas are electronic [51]. Still, not all high schools offer electronic diplomas, and students who take subjects privately need to provide these grades themselves. This means that students often have to scan and upload documents when applying for universities, meaning that they can easily be modified and go undetected.

At some point during the school year, universities take a sample of the accepted students' grade transcripts to verify, and the chosen students must present their original grade transcripts upon request. The verification process is done manually, and includes matching fonts, resolution, dates, etc. From this, they can reveal some forgeries, but chances are that some forgeries even pass this inspection. [50]

The consequence of this verification process is that students who have forged grade transcripts steal the spots in study programs from applicants who were better qualified. Even if the forgery is noticed throughout the school year, the qualified applicant still lost their spot in the study program.

6.2.2 Foreign Applicants

Foreign applicants to Norwegian universities have also been known to forge diplomas or English tests. According to [50], NOKUT[2] detected 15 forgery cases in 2017, a 36% increase from the previous year. It is naive to assume that all forgery cases by foreign applicants are detected.

Foreign diplomas do not follow a certain standard, and thus require further investigation to verify the documents. Methods that are used to verify such documents are to ensure that the institution exists and has been accredited, and to detect red flags such as students with low or mediocre high school results that has a university degree from a top rated university.

6.2.3 Solution

A solution to the challenges mentioned earlier in this section is to implement a document verification system in the education sector. There has already been some improvement to the verification process in recent years with the introduction of

[2] Norwegian Agency for Quality Assurance in Education (NOKUT): Handles applications about recognizing foreign education.

a service called the Diploma Registry[3]. This is a service where students in higher education can collect their results and share them with other education institutions, employers, or other participants [52]. The grade transcript can be shared either via a link, or can be downloaded as a PDF document, and then shared as a file. When it is shared as a link, the third party can be certain that the results have not been altered, but with a PDF document, there is room for modification by the student. The most secure solution would be to implement a system like the one in DCApp. This would especially be useful in the Norwegian application process from high school to university, and it is realistic to expect a similar system in the future, as the Diploma Registry has been a start of the digitalization of the education sector. Advantages with a system in this sector include less use of physical paper, it is easier to detect forgery, all applications can be verified, and no students will lose their place in a study program to someone who has forged their application.

Ideally, a document verification system as implemented in DCApp would be integrated into all educational institutions in the world, but considering the difference in modernization and maturity makes this problematic and unrealistic. However, it could be expected that similar solutions could be implemented across some countries that would prioritize this. It would provide a more efficient verification process in all institutions who have to acknowledge educations from countries with these systems.

6.3 Banking

The system described throughout this thesis could be an ideal implementation in the banking sector. This section will outline the current situation, and what improvements this system could bring.

6.3.1 BankID

In Norway, there is an electronic form of identification known as BankID. BankID is available in two forms: on mobile devices and as a separate BankID device. For BankID on mobile devices, the user has to confirm a random two-word phrase that is presented on the login page and provide their personal 4 to 8 digit code. For the separate BankID device, the user has to press a button on the device that presents them with a randomly generated 6-digit sequence. These two methods are used in combination with the user's social security number, and in some cases a password. This form of identification is considered as authentic as identifying a person using their driving license or passport, and can be used to purchase goods online, sign

[3] Diploma Registry is known as Vitnemålsportalen in Norwegian

documents, buy and sell stocks, tax return submission, and more [53]. BankID has become a widely used form of identification in Norway, and has contributed to automating many processes, saving companies both time and money. It also makes it easy for individuals to identify themselves, and often eliminates the need for in-person meetings.

Although this electronic form of identification has contributed to several improvements in society, there are some threats associated with this method. The following sections will address the issues regarding identity theft and document forging.

6.3.2 Identity Theft

Identity theft has been limited to some degree with the electronic identification system. An example of where BankID must be used for identification are cases where an individual wishes to apply for a loan. In some extreme cases, a loan will be granted with the presentation of a copy of an ID and social security number belonging to a person. This can be obtained by anyone who gets a hold of a person's credit card in Norway. However, in most cases, some further form of authentication is required to avoid fraud, and this is where BankID is used to validate the identity.

The required use for BankID has made it more troublesome to commit identity theft, but far from impossible. Most cases where BankID has been used to commit identity theft, the crime is committed by someone who has access to the victim's BankID device. Particularly these crimes are committed by people who live, or have previously lived, in the same household. In these cases, the criminal would have easy access to the victim's BankID, and a great deal of harm could be caused before the victim is made aware of the situation. Replacing the current BankID solution with a solution based on the use of biometrics and signing methods described in this thesis would eliminate most of the identity theft situations we see today. This would protect individuals from being put in a difficult financial situation as a result of identity theft, and also protect insurance companies from having to cover the costs of the financial damage.

6.3.3 Forging Documents

BankID is primarily used for identification, and does not provide any significant protection against document forgery. There are several scenarios where BankID is used to identify a person, and following this, the user is to upload some documents. Using banks and loans as an example again, this is a process normally required when applying for a loan. Relevant documents for this process are pay slips, tax returns, and bank statements. The way the system works today is that it is sufficient to upload a screenshot or file containing the required documentation. A problem with this is that documents can easily be manipulated using simple computer software. Forgeries can include changing the income on a pay slip or balance in a bank statement, and often goes unnoticed until it is too late.

A system where documents could easily be checked for unauthorized modifications would provide a great contribution to society. It would effectively stop processes that include forged documents, saving banks time and money. An efficient process for distributing signed documents, and for eliminating document forgeries, would improve several sectors, some of which will be discussed in the following sections.

6.4 Health Sector

Several issues could be solved in the medical sector with the introduction of a document verification system explained in this thesis.

6.4.1 Health Insurance

Medical journals are used in cases such as applying for health insurance, surgeries, and when accidents or illness occur abroad. Taking the application process for health insurance as an example here, a more trustworthy system would provide great benefits for insurance companies. According to [54], fraud cases concerning health insurance were discovered for approximately NOK 140 million in 2018. These cases consist of individuals reporting false or inconclusive information about their medical history. Fraud makes insurance more expensive for honest clients, as the insurance companies' revenue needs to cover their payouts. Although insurance companies can detect some cases, there is likely a large number of fraud cases that go undetected, costing the companies, and other clients, millions. Without a proper document verification system, the process is manual, requiring several man-hours, and thus very prone to human errors. Having a document verification system, as explained in this thesis, would improve the document verification process drastically, increasing the level of detected fraud cases, as well as making the process much more efficient. The system would ensure that all medical history was confirmed by a medical institution. In addition to saving companies millions, it would also protect the individuals who might, otherwise, present incomplete medical information by accident.

Granting all of this, the system will not be 100% secure if individuals have medical records from different countries, and not all countries have implemented the system. However, this is not the case for most Norwegian residents, and the system would contribute to considerable savings both in wrongful payments and man-hours.

6.4.2 Medical Prescriptions

Another example that was mentioned in Section 1.2.3 in regards to the health sector is the forgery of prescriptions. Several news reports published in 2016, [55, 56], address cases where citizens have forged prescriptions to obtain an excessive amount of prescription drugs. By forging these prescriptions, they were able to obtain large amounts of pills worth hundreds of thousand Norwegian kroner. These drugs could be distributed illegally to other parties, or contribute to addiction. The cases from these articles are just some of the exposed forgery cases regarding medical prescriptions, and there likely are many more that have not been revealed.

The verification process with prescriptions on paper is manual, and forgeries may not be detected at first glance, or at all. From 2013 electronic prescriptions started to substitute paper prescriptions, which according to the Norwegian Pharmacy Association has reduced the reported number of forged invoices by 50% from 2013 to 2015 [55]. This is a positive trend, but there are still improvements to be made to this system to reduce the number of forgeries further. The document verification system explained in this thesis would contribute to a more secure system where the number of forgeries would be diminished. It would protect the government from subsidizing expensive medications to people who are not eligible.

6.5 Oil and Gas Industry

The massive flow of documents in the oil and gas industry indicates that a secure document verification would be beneficial in this sector. Although digitalization has provided many benefits to this sector, it has also created a need for more sophisticated security systems. A system as the one outlined in this thesis could provide much needed security to many areas in this industry. It is critical to protect companies against fraud, hacker attack, and to establish trust.

6.5.1 Protection Against Hackers

With the continuing digitalization, there is an increase in automated processes, including drilling, operation of wells and pumps, and utilizing industry robots. These automated processes have and will continue to provide a great financial benefit, both in better utilization of human resources, and more precision of the work performed. However, having systems that operate these processes also pose a danger of being hacked and abused by cybercriminals. According to several articles, [57, 58, 59], systems can be vulnerable to attacks by hackers. There have been cyberattacks that have caused great damage, and there is a danger that systems are more vulnerable when processes are automated, and machines are connected to a larger network. Cyberattacks mentioned in the articles include hackers stealing credentials, intrusions into control systems, and sabotage. Article [57] addresses a case where there was an attack on a shared data network, forcing pipeline operators to temporarily shut down computer communications as a response. Without the quick response, the attack could have lead to data loss, explosions, spills, or fires. The attack mentioned in [58] was designed to sabotage the operation and trigger an explosion. However, due to a mistake in the attackers' computer code, the explosion was prevented.

To increase security, the use of biometrics should be used as a form of identification before any actions can be done to a system. Such actions can be requesting the starting of a machine, shutting down power, and sharing, modifying or deleting data. If a platform receives a request (document) to start a machine, the platform should be able to verify that the request is authentic. A system based on the principles in DCApp ensures that only people that have been authorized can perform actions, as the requests can easily be verified as authentic or not.

6.5.2 Forged Invoices

One issue that the oil and gas industry has been a victim of, amongst other industries, is receiving forged invoices [60]. This scam is most common in Sweden, but has quickly moved to Norway. Criminals create companies, often with similar names as well known suppliers, and sends invoices to whomever they wish to scam, in the hope that the scam will go unnoticed. If this is not detected in time, it could result in large costs for the affected companies. A document verification system implemented across industries could easily notify companies if an invoice is not from an authorized supplier, or if it has been modified. Suppliers, as their role as the signer, signs invoices with their private key by identifying themselves through biometrics. Their public key would be available through a certificate signed by a CA, and the invoice could easily be verified by the invoiced party.

Another example of internal issues regarding invoices, is employees sending forged invoices to customers, and taking the money as payment to themselves. According to [61], a hired consultant for Equinor was indicted on corruption and fraud in 2010. He had received approximately NOK 100,000 by falsely invoicing a customer. There are no precise numbers on how widespread fraud cases of this sort are, but these cases could potentially end up costing companies millions, especially when they go unrevealed. To prevent similar cases, a document verification system could secure companies against these types of vulnerabilities by requiring the signer of a document to be an authorized signer, or even require more than one authorized signer to sign documents before they are considered valid.

6.6 Associated Risk

The implementation of a document verification system explained would provide many benefits to companies. However, even with such a system, there are still threats and vulnerabilities. This section will address some of these issues.

6.6.1 Loss of Biometric Data

As the security of this system is based on biometrics, it is essential that this is unique for individuals. In Apple's Secure Enclave, a mathematical representation of the device owner is stored, and is used to verify new Face ID or Touch ID attempts. Some of the most important security features of the Secure Enclave are that it is strongly encrypted, no information leaves it, and it is its own separate system. Today there is no indication that the Secure Enclave will face security issues, but this may not be the situation forever. Hopefully, all technologies used to handle biometric data will be secured against all types of attacks at all times, but if this fails, then there is a chance of people's biometrics getting stolen. The risk of stolen biometric data is minimal, but is worth mentioning to enhance the importance of the security required in the document verification system. If biometric data is stolen, it cannot be changed, and the document verification system would lose trustworthiness.

6.6.2 Loss of Private Key

The validity of this system is also based on the private keys. Every person in the system has a secret, which is their personal private key. If somehow, a private key is lost or stolen, the key has become compromised, meaning that it may not be a secret anymore. A consequence of a compromised private key is that people with dishonest intentions can use it to sign documents such that it appears as the original private key owner signed them. It is therefore crucial that the private key is well protected. DCApp has the necessary protection of the private key, as the private key is protected by access control that requires a valid Face ID or Touch ID to use it, and items in the keychain are always encrypted. It is crucial for any real world implementation of a similar document verification system to have the same degree of security for the private keys.

6.6.3 Trustworthiness of Signers

One vulnerability, which is challenging to solve, is the case where trusted people act dishonestly. If an individual has been granted an authorized signing position, the system relies on the individual's actions to be legitimate. If any document only requires one signer, that signer has the possibility to forge a document, sign it, and hence it will appear valid to any party wishing to verify it. This is why it is important in a real world application of the system to usually require more than one signer to sign a document for it to be considered valid. The more signers that are required, the higher the credibility of the document. Although it is not impossible, it is more unlikely for a number of signers to collude to create forged documents that will appear valid.

6.6.4 Trustworthiness of Certificate Authority

The document verification system is designed to easily verify if a document is valid, and signers need to provide proof of being an authentic institute. Taking the example in Section 6.5.2 with forged invoices, the system would be weakened if any party could contact a CA and claim that they are a legitimate supplier, and thus start issuing signed invoices that will appear valid. To avoid this, it is necessary that the CA takes their role seriously, and that they ensure that they are distributing the public keys of genuine establishments. This has to include some manual work, but it is necessary to maintain a good reputation. The verifier should be observant to ensure that the CA is considered trustworthy.

Chapter 7

Conclusion

This thesis has introduced a document verification system with biometrics as the form of identification. The principles of the system have been explained, and a sample application, DCApp, has been built based on the available technology. Although DCApp is a sample application where not all features of a complete system have been implemented, it demonstrates the principles of the system. Combining the features of DCApp and the additional system requirements explained in this thesis constructs a complete document verification system based on biometrics in theory.

A more digitalized document verification system solves a lot of the issues that we face today, mostly regarding document forgery and identity theft. It would solve the vulnerabilities concerned with digital identification, and create a system that is much more difficult to manipulate. The integration of the document verification system based on biometrics explained in this thesis would lead to higher security, less manpower needed for verification processes, increased trust, and a much more efficient process. In a society where most of the document flow and communication is digital, it is essential to have a system that can verify identity and document validity, which this system does. Replacing all current systems immediately is not a realistic expectation, but as the need for these security features increases, it will become more attractive for sectors to implement a similar system. In addition to increasing security, it also allows for the verification process to become automated, making it less prone to human error. Utilizing a system that is less prone to errors and fraud will also result in substantial savings for sectors that are susceptible to this.

Of course, the implementation of such a system could not completely replace other systems today, as the use of devices supporting biometric identification is not currently mandatory in any of the mentioned use cases, or similar cases. However, from the findings presented in Chapter 6, it can be concluded that industry sectors and companies have a need for a more sophisticated document verification system, as well as a more secure method for identification. If they are dedicated to integrating a secure document verification system, and are prepared to take the necessary steps to implement one, it would result in a solution to these problems.

List of Figures

List of Listings

List of Tables

Appendix A

Accessing Code

A.1 Prerequisites

- A Macintosh is required to run the code.
- Xcode. Xcode is an Integrated Development Environment (IDE) for macOS, and can be downloaded from the App Store.
- An Apple Developer account in a Developer Program is required (this is a paid account).
  - Without a paid account, the code can still be viewed in Xcode and run in the Xcode simulator. In the simulator the biometric features are replaced with a passcode. AirDrop is not available in the simulator.

A.2 Download and Run Code

Go to the GitHub repository, DCApp. Follow the instructions in the README.md file.

Link to GitHub repository: `https://github.com/elisabethbratli/DCApp`

References

[1] Tor Gunnar Tollaksen. *Tre av fire oljeselskap har vært utsatt for alvorlige dataangrep*. URL: https://sysla.no/teknologi/tre-av-fire-oljeselskap-har-vaert-utsatt-alvorlige-dataangrep/ (visited on 05/07/2019).

[2] Blockgeeks. *What is Blockchain Technology?* URL: https://blockgeeks.com/guides/what-is-blockchain-technology/ (visited on 04/23/2019).

[3] Hein Meling. *Efficient Trustworthy Computing with Blockchains and Biometrics*. URL: https://prosjektbanken.forskningsradet.no/#/project/NFR/274451 (visited on 05/28/2019).

[4] SSL2BUY. *Symmetric vs. Asymmetric Encryption – What are differences?* URL: https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences (visited on 05/28/2019).

[5] Norton Symantec employee. *What is a man-in-the-middle attack?* URL: https://us.norton.com/internetsecurity-wifi-what-is-a-man-in-the-middle-attack.html (visited on 05/28/2019).

[6] Rafael Almeida. *Symmetric and Asymmetric Encryption*. URL: https://hackernoon.com/symmetric-and-asymmetric-encryption-5122f9ec65b1 (visited on 05/28/2019).

[7] Sectigo Limited. *What Is a Digital Signature?* URL: https://www.instantssl.com/digital-signature (visited on 04/23/2019).

[8] Jscrambler. *Hashing Algorithms*. URL: https://blog.jscrambler.com/hashing-algorithms/ (visited on 04/23/2019).

[9] Julie Olenski. *ECC 101: What is ECC and why would I want to use it?* URL: https://www.globalsign.com/en/blog/elliptic-curve-cryptography/ (visited on 05/29/2019).

[10] Techotopia. *An Overview of Public Key Infrastructures (PKI)*. URL: https://www.techotopia.com/index.php/An_Overview_of_Public_Key_Infrastructures_(PKI) (visited on 04/18/2019).

[11] Thomas Pornin. *How does the digital signature verification process work?* URL: https://security.stackexchange.com/questions/8034/how-does-the-digital-signature-verification-process-work (visited on 05/14/2019).

[12] Wikipedia. *Biometrics*. URL: https://en.wikipedia.org/wiki/Biometrics (visited on 02/06/2019).

[13] Margaret Rouse. *MD5*. URL: https://searchsecurity.techtarget.com/definition/MD5 (visited on 05/28/2019).

[14] Brandon Vigliarolo. *Brute force and dictionary attacks: A cheat sheet*. URL: `https://www.techrepublic.com/article/brute-force-and-dictionary-attacks-a-cheat-sheet/` (visited on 05/28/2019).

[15] Foudil Belhadj. *Biometric system for identification and authentication*. URL: `https://hal.archives-ouvertes.fr/tel-01456829/document` (visited on 02/06/2019).

[16] Margaret Rouse. *File Storage*. URL: `https://searchstorage.techtarget.com/definition/file-storage` (visited on 02/28/2019).

[17] theiphonewiki. *Secure Enclave*. URL: `https://www.theiphonewiki.com/wiki/Secure\_Enclave` (visited on 02/01/2019).

[18] Apple Inc. *iOS Security*. URL: `https://www.apple.com/business/site/docs/iOS_Security_Guide.pdf` (visited on 05/08/2019).

[19] Apple Inc. *Storing Keys in the Secure Enclave*. URL: `https://developer.apple.com/documentation/security/certificate_key_and_trust_services/keys/storing_keys_in_the_secure_enclave` (visited on 04/17/2019).

[20] Apple Inc. *kSecAttrSynchronizable*. URL: `https://developer.apple.com/documentation/security/ksecattrsynchronizable?language=swift` (visited on 04/29/2019).

[21] Rajkumar. *Securing and Encrypting Data on iOS*. URL: `https://code.tutsplus.com/tutorials/securing-and-encrypting-data-on-ios--mobile-21263` (visited on 06/09/2019).

[22] Apple Inc. *LocalAuthentication*. URL: `https://developer.apple.com/documentation/localauthentication` (visited on 04/23/2019).

[23] Apple Inc. *Accessing Keychain Items with Face ID or Touch ID*. URL: `https://developer.apple.com/documentation/localauthentication/accessing_keychain_items_with_face_id_or_touch_id` (visited on 04/17/2019).

[24] Wikipedia. *Touch ID*. URL: `https://en.wikipedia.org/wiki/Touch_ID` (visited on 02/28/2019).

[25] David Cardinal. *How Apple's iPhone X TrueDepth Camera Works*. URL: `https://www.extremetech.com/mobile/255771-apple-iphone-x-truedepth-camera-works` (visited on 02/28/2019).

[26] Jason Cipriani. *Apple Face ID: Everything you need to know*. URL: `https://www.cnet.com/how-to/apple-face-id-everything-you-need-to-know/` (visited on 02/28/2019).

[27] Daniel Nations. *What Is AirDrop? How Does It Work?* URL: `https://www.lifewire.com/what-is-airdrop-how-does-it-work-1994512` (visited on 03/25/2019).

[28] Kailash Mondal. *What is the relationship between Xcode, Swift and Cocoa?* URL: `https://www.quora.com/What-is-the-relationship-between-Xcode-Swift-and-Cocoa` (visited on 02/28/2019).

[29] Wikipedia. *Swift (programming language)*. URL: `https://en.wikipedia.org/wiki/Swift_(programming_language)` (visited on 02/28/2019).

[30] Wikipedia. *Cocoa Touch*. URL: https://en.wikipedia.org/wiki/Cocoa_Touch (visited on 02/28/2019).

[31] Doug Stevenson. *What is Firebase? The complete story, abridged*. URL: https://blog.usejournal.com/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0 (visited on 02/27/2019).

[32] Firebase. *Cloud Firestore*. URL: https://firebase.google.com/docs/firestore/ (visited on 02/27/2019).

[33] Quackit. *Querying a Database*. URL: https://www.quackit.com/database/tutorial/querying_a_database.cfm (visited on 05/10/2019).

[34] Firebase. *CFData*. URL: https://firebase.google.com/docs/firestore/security/overview (visited on 04/18/2019).

[35] Wikipedia. *Zero-knowledge proof*. URL: https://en.wikipedia.org/wiki/Zero-knowledge_proof (visited on 02/27/2019).

[36] Apple Inc. *kSecAttrAccessibleWhenUnlocked*. URL: https://developer.apple.com/documentation/security/ksecattraccessiblewhenunlocked (visited on 05/10/2019).

[37] Firebase. *Users in Firebase Projects*. URL: https://firebase.google.com/docs/auth/users (visited on 05/13/2019).

[38] Tutorialspoint. *Swift - Classes*. URL: https://www.tutorialspoint.com/swift/swift_classes.htm (visited on 05/10/2019).

[39] Firebase. *Security Rules language*. URL: https://firebase.google.com/docs/rules/rules-language (visited on 05/09/2019).

[40] Dictionary.com. *API*. URL: https://www.dictionary.com/browse/api (visited on 05/09/2019).

[41] Apple Inc. *SecKeyCreateSignature*. URL: https://developer.apple.com/documentation/security/1643916-seckeycreatesignature (visited on 04/18/2019).

[42] Apple Inc. *SecKeyVerifySignature*. URL: https://developer.apple.com/documentation/security/1643715-seckeyverifysignature?language=objc (visited on 04/17/2019).

[43] Apple Inc. *CFData*. URL: https://developer.apple.com/documentation/corefoundation/cfdata-rv9 (visited on 04/17/2019).

[44] Timo Hetzel. *iOS Backups & Device Migration*. URL: https://hetzel.net/2014-09-20/ios-backups-device-migration/ (visited on 04/26/2019).

[45] Apple Inc. *Two-factor authentication for Apple ID*. URL: https://support.apple.com/en-us/HT204915 (visited on 05/07/2019).

[46] Apple Inc. *iCloud security overview*. URL: https://support.apple.com/en-us/HT202303 (visited on 05/07/2019).

[47] Firebase. *Firebase Cloud Messaging*. URL: https://firebase.google.com/docs/cloud-messaging (visited on 04/26/2019).

[48] Apple Inc. *CloudKit*. URL: https://developer.apple.com/icloud/cloudkit/ (visited on 05/08/2019).

[49] Max Chuquimia. *I need a cloud database: Firebase or CloudKit?* URL: `https://medium.com/the-lair/firebase-vs-cloudkit-1b2fcaef2d2a` (visited on 06/04/2019).

[50] Jørgen Svarstad. *Fikk studieplass etter å ha forfalsket sju karakterer fra videregående*. URL: `https://www.forskerforum.no/fikk-studieplass-etter-a-ha-forfalsket-sju-karakterer-pa-vitnemalet/` (visited on 05/13/2019).

[51] Kunnskapsdepartementet. *Elektronisk vitnemål*. URL: `https://www.samordnaopptak.no/info/soke/elektronisk_vitnemal/index.html` (visited on 06/09/2019).

[52] Kunnskapsdepartementet. *Diploma registry*. URL: `https://www.vitnemalsportalen.no/english/` (visited on 05/13/2019).

[53] BankID. *Things to use BankID for*. URL: `https://www.bankid.no/privat/bruksomrader/` (visited on 04/16/2019).

[54] Martine Holøien. *Arrangerer bilkollisjoner og lyver om egen helse*. URL: `https://www.hegnar.no/Nyheter/Naeringsliv/2019/03/Arrangerer-bilkollisjoner-og-lyver-om-egen-helse?r=refresh` (visited on 05/16/2019).

[55] Eivind Kristensen. *Svindlet apotek over hele Sørlandet*. URL: `https://www.fvn.no/nyheter/lokalt/i/EkJwP/Svindlet-apotek-over-hele-Sorlandet` (visited on 05/16/2019).

[56] Anette Stensholt. *Svindlet til seg piller for nær 160 000*. URL: `https://www.nrk.no/vestfold/svindlet-til-seg-piller-for-naer-160-000-1.12923359` (visited on 05/16/2019).

[57] Clifford Krauss. *Cyberattack Shows Vulnerability of Gas Pipeline Network*. URL: `https://www.nytimes.com/2018/04/04/business/energy-environment/pipeline-cyberattack.html` (visited on 06/04/2019).

[58] Nicole Perlroth and Clifford Krauss. *A Cyberattack in Saudi Arabia Had a Deadly Goal. Experts Fear Another Try.* URL: `https://www.nytimes.com/2018/03/15/technology/saudi-arabia-hacks-cyberattacks.html?module=inline` (visited on 06/04/2019).

[59] David E. Sanger. *Utilities Cautioned About Potential for a Cyberattack*. URL: `https://www.nytimes.com/2016/03/01/us/politics/utilities-cautioned-about-potential-for-a-cyberattack-after-ukraines.html?module=inline` (visited on 06/04/2019).

[60] Marit Kaarbø Løvold. *Se opp for falske e-poster og fakturasvindel!* URL: `https://www.regnskapnorge.no/faget/artikler/bransjeaktuelt/se-opp-for-falske-e-poster-og-fakturasvindel/` (visited on 05/07/2019).

[61] VG. *Statoil-konsulent tiltalt for grov korrupsjon*. URL: `https://www.vg.no/nyheter/innenriks/i/4Mo0V/statoil-konsulent-tiltalt-for-grov-korrupsjon` (visited on 05/07/2019).