

Sami Kuhmonen

One-Time Password Implementation for Two-Factor Authentication

Metropolia Ammattikorkeakoulu

Bachelor of Engineering

Health Technology

Thesis

1.03.2017

Author Title	Sami Kuhmonen One-Time Password Implementation for Two-Factor Authentication
Number of Pages Date	53 pages March 1 st , 2017
Degree	Bachelor of Engineering
Degree Programme	Health Technology
Specialisation option	
Instructor	Kari Björn, Head of Degree Programme
<p>The purpose of this work is to present different ways of implementing two-factor authentication for an online biobank system and implement one such system both on the server side and as a mobile application.</p> <p>First the threats against medical systems are presented showing the reasoning for greater security. The type of information stored in these systems and the desirability of the information for attackers is explained. General legislation is presented by examples from the United States of America as well as Finland.</p> <p>Then the possible authentication methods and basic cryptography are explained. The chosen implementation is shown with the algorithms used.</p> <p>The work succeeded in implementing the system using time-based one-time passwords as presented in existing work but extending it by using newer cryptographic methods and longer passwords to enhance the security even further. A few further enhancements are explained that could be implemented in the future.</p>	
Keywords	authentication, two-factor, one-time passwords

Tekijä Otsikko	Sami Kuhmonen Kaksivaiheisen tunnistautumisen toteutus kertakäyttösalasanoilla
Sivumäärä Päivämäärä	53 sivua 1.3.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Hyvinvointiteknologia
Suuntautumisvaihtoehto	
Ohjaaja	Kari Björn, Head of Degree Programme
<p>Työn tarkoitus on esittää toteutusvaihtoehtoja kaksivaiheiseen autentikointiin verkkopohjaisessa biopankkijärjestelmässä ja toteuttaa yksi näistä sekä palvelinpuolella että mobiiliapplikaationa.</p> <p>Ensin esitellään lääketieteellisten järjestelmien uhkia sekä syitä parempaan turvallisuuteen johtuen talletettavien tietojen arkaluonteisuudesta sekä tietojen kiinnostavuudesta hyökkääjille. Aiheeseen liittyvää lainsäädäntöä käydään läpi Suomen ja Amerikan Yhdysvaltojen osalta.</p> <p>Mahdollisia autentikaatitapoja selitetään sekä annetaan perustietoja kryptografisista menetelmistä. Esitetään valittu toteutus ja käytetyt algoritmit.</p> <p>Kertakäyttöiset salasanat toteutettiin aikapohjaisina aiemmin esitettyjen määritelmien mukaisesti, mutta käyttäen uudempia kryptografisia menetelmiä ja pitempiä salasanoja parantaen turvallisuutta entisestään. Muutamia mahdollisia jatkokehityksiä esitetään.</p>	
Avainsanat	autentikointi, kaksivaiheinen, kertakäyttösalasanat

Abbreviations

1	Introduction	1
2	Security in Medical Systems	4
2.1	Basic Security	5
2.2	Requirements by FDA	6
2.3	Legislation in Finland	7
3	Attacks on Medical Systems	9
3.1	Attack Prevalence	9
3.2	Reasons for Attacks	11
3.3	Ransomware Attacks	12
3.4	Cost of an Attack	12
3.5	Assumption of Safety	12
4	Authentication Methods	14
4.1	Available Methods	14
4.2	Token Authentication	15
4.3	Username/Password Authentication	16
4.4	Two-Factor Authentication	18
4.4.1	Definition	18
4.4.2	Two-Factor Using SMS	19
4.4.3	Two-Factor Using Pre-Generated Passwords	19
4.4.4	Two-Factor Using One-Time Password	20
5	History of Cryptography	22
5.1	Ancient Encryptions	22
5.2	Middle Ages	23
5.3	Modern Times	23
6	Cryptography Basics	25
6.1	Strong and Weak Cryptography	25
6.2	Governmental Recommendations	26
6.3	Security by Obscurity	27
6.4	Encryption vs Hashing	27
6.5	Hash Functions	28
6.5.1	SHA Family	29
6.5.2	SHA256 Algorithm	29
6.5.3	SHA512 Algorithm	30

6.6	Hash-Based Message Authentication Code	30
6.7	Random Numbers	31
6.7.1	Random Number Generation	31
6.7.2	Pseudo-Random Number Generators	31
6.7.3	True Random Number Generators	31
6.7.4	Ivy Bridge RNG	32
7	Implementation	34
7.1	Platform	34
7.2	Chosen Method	34
7.3	Requirements	35
7.4	Implementation in Short	37
7.5	Used Algorithms	37
7.5.1	Base32 Algorithm	37
7.5.2	Truncation Method	38
7.5.3	Key Generation	39
7.5.4	One-Time Password Generation	40
7.6	User Experience	41
7.6.1	Creating a One-Time Password	42
7.6.2	Logging in Procedure	42
7.6.3	Deleting Password	43
7.7	Server Implementation	43
7.8	Storing Data	44
8	Security	45
8.1	Algorithm Security	45
8.2	Server Security	45
8.3	Client Security	46
8.3.1	Storage	46
8.3.2	PIN Code	47
8.3.3	Apple Touch ID	47
8.4	Randomness of the Password	47
8.5	General Security	48
9	Conclusions	49
9.1	General Conclusions	49
9.2	Possible Enhancements	49
	References	51

Abbreviations

OTP	One-Time Password. A password that is valid only once.
SHA	Secure Hash Algorithm. A family of hashing algorithms used to determine the authenticity of a message, or other piece of data.
TRNG	True Random Number Generator. A source for aperiodic and nondeterministic random numbers
PRNG	Pseudo-Random Number Generator. A source for numbers that try to be random, but are deterministic and periodic.
HMAC	Hash-based Message Authentication Code. A method for calculating a code for authenticating messages based on a hash algorithm.
NIST	The National Institute of Standards and Technology. A governmental agency that develops and applies technology, measurements, and standards in the United States of America.
FDA	U.S. Food and Drug Administration. A governmental agency in The United States of America responsible for public health, including medical systems and products.
HIPAA	Health Insurance Portability and Accountability Act. Legislation that handles health insurances, fraud and stipulates industry-wide standards in The United States.
AES	Advanced Encryption Standard. A specification for encryption of electronic data defined by NIST.
ENISA	European Union Agency for Network and Information Security. A European Union agency for cyber security.

1 Introduction

Medical practitioners and researchers handle a lot of sensitive data on individuals in their work. These data are meant to be used only in the treatment of the patient and ensuring their wellbeing or, in the case of research, to create new treatments or ensure the efficacy of existing ones. Doctors have a need to keep specific records of the individuals and their health history, but researchers usually can handle the data as anonymized without information on who the actual individual behind the information or samples is.

Historically medical data were kept on paper in records and anyone who had access to the storage room could access them. There was no audit trail on who had read, modified or even deleted information from the records. The information was also only in one place, unless copies were made, but even in that case any possible additions or corrections were not available to others.

Nowadays medical records are kept increasingly as electronic records which allows easy access regardless of location. The information can be kept up to date and shared between healthcare providers. This creates new challenges for security and allows a whole new way for attackers to attempt to attain the information. There is no need to physically break into a hospital record storage, all that is needed is a vulnerable system connected to a network.

The reasons for attacking medical systems vary. Some may be attacking any system they can get into, some may be searching for specific information and some may be looking for a way to make money by blackmail. There have been cases of taking control of the servers and demanding money for releasing control and the media are very interested in any private information about public figures.

The records also contain other information than medical data: credit card information, social security numbers, insurance details and many other information that can be used to the attacker's advantage [22]. With enough information identity theft or insurance fraud

is possible and may cause huge problems to the individuals before it is discovered. Proving these cases may also be very difficult. Also if patients don't trust their doctors and clinics to keep their private information safe they might not want to volunteer information that might be critical for their proper care. This may also affect information gathered for research or assessing the state of public health.

Due to all this the regular methods for logging in based on a username and a password may not be enough to ensure the confidentiality of the data. The authentication must be made stronger to ensure that only the authorized people are allowed access to the data in the system.

A multinational company provides services in the biobank and medical information system sector. They produce online services which allow the storage, retrieval, anonymization and research services for a variety of medical service providers. Due to the nature of the data, customers and the system being available online they need to have strong user authentication and data protection. Their current system has a regular one-factor login system with username and password and a possibility to have a second factor authentication via SMS. Using SMS as the second factor is seen both expensive and cumbersome so another way of providing enhanced security for user authentication is needed.

The goal of this thesis is to devise a two-factor authentication system for this company. There are several two-factor authentication methods available so the purpose is not to create a completely new method. Basic principles of cryptography, examples of two-factor authentication methods and an implementation of one method will be presented.

The system will be based on one-time passwords, which are commonly used in providing a second factor authentication from social media to online banking. The choice is due to these methods being well tested and proven effective as well as easy to use. Other possible methods for second factor authentication are also discussed and the reasons for not using them are explained.

The system will be built as a two-tier application. Mobile applications will handle the user-side creation of token and server application will handle authentication into the service.

First the current state of security in medical systems is presented and then the types of attacks happening on these systems. The different ways of authentication are explained and some history of cryptography as well as basics of cryptography is given. Then the implementation is explained and a short analysis to its strength and conclusions about the work is presented.

2 Security in Medical Systems

User authentication and data security are complex problems affecting all areas of computing. It is needed from the everyday social media all the way to banking, medical information, and state secrets. We are constantly required to authenticate ourselves in one way or another and require the same from others. We have locks in our doors to let ourselves in and keep others out. At work we recognize each other automatically and can be suspicious if a stranger suddenly walks in the office. We are given passwords and PIN codes to access our bank accounts and other services. All this requires authentication and verification of the user.

Not all of the systems we authenticate to are overly important. If someone gets hold of someone else's password to a video rental company they may be able to see what someone has watched and maybe get their bonus points but not much more. Accessing a hobby forum for crocheting may be a nuisance and allow spamming but not cause any bigger issues.

Some of the systems are a lot more important. Accessing our social media profiles may allow defamation, finding out private information and causing many issues in social life, maybe even in professional capacity. If someone gets access to our bank account they may be able to take all our money, apply for loans, and other serious things. If they access our medical records they may get very personal information or maybe even affect the treatment we get the next time we go to the hospital. It could even be life threatening if a fatal allergy information is altered.

We also use some of the systems to log in to other systems. This makes it even more important to keep those systems private and only accessible to ourselves.

In the medical field a lot of sensitive and personal information is gathered. This information may include diseases, health concerns, treatment plans, family history, psychological evaluations and many other things. The access to this information should be restricted to those who really need it and there should be no reason for users to access

every kind of information. They should only be able to access the information that is required for them to do their jobs. [24]

In research users often handle large amounts of personal data. They don't usually need access to all of the personal data, only subsets of it. For example, doing tests on samples taken from a group of people may require information on the age, gender, race and if the person is a smoker. Very rarely would there be need to know the names, addresses and other information for the donors of the samples. This also makes the research much easier since the rules for handling anonymized data are much more relaxed than those governing the use of full personal data. [25]

2.1 Basic Security

All database systems should have security in place to protect their contents, but when the databases contain medical or personal information this becomes a strict requirement. The security measures that have to be in place vary depending on the customer and country where the service is provided. Every country has different legislation and every customer may have different views on how the data should be handled and presented.

For any given database there are certain basic security measures that should be in place. The users must be authenticated to make sure only authorized people are given access to the system. The users must only be authorized to see, modify or add data that they are entitled to. If a person is doing statistics of all visitors in a hospital they should not have any access to names or medical history of the users, and a nurse taking blood samples does not need to know about the psychiatric therapy the patient is going through unless it relates to the blood test. [27]

For every action taken in the system there needs to be an audit trail recorded. Every user accessing any information must be recorded and be verified in a way that cannot be altered afterwards. The audit trail must be detailed enough to have information on which data were accessed, not only which user accessed whose information. This allows anyone to validate that nobody has accessed information without a proper cause. [26]

In addition to these basic principles there are other things to consider when creating an information system. How are the users allowed to log in? What kind of passwords are acceptable to use? How often should they be changed? How to make sure the users remember the passwords and don't just write them down next to the machine? What access should an administrator have to the system? Should the administrator have access to other user accounts? How are digital signatures handled in the system?

2.2 Requirements by FDA

If a system is to be used in The United States there are guidelines to meet. Some systems and devices have stricter requirements than others but it is always good to strive to provide the largest coverage of the requirements, even if it is not legally required.

The Food and Drug Administration has several guidelines for information systems used in the medical field. For example, the 21 CFR part 11 dictates requirements for electronic signatures, including the following: [4]

- each signature must be unique to the user
- user must be verified by the organization providing the signature
- there must be at least two identification components, such as identification code and password, if the signature is not based on biometrics
- only the owner of the ID code and password may use them
- no two users may have the same combination of identification components
- possible passwords should be revised periodically
- attempts to use identification components, such as trying to log into a user's account with an incorrect password, must be detected and acted upon

2.3 Legislation in Finland

There is a lot of legislation concerning medical procedures, records, and their use in Finland. Access to this kind of information must be restricted to only those people who require the information and only partial information should be available if there is no reason to have the whole patient history. This means that, for example, if a person comes in for an x-ray of the foot the radiographer should not have access to any information about the patient's possible earlier medical history that is not relevant to the foot injury.

Electronic patient records must have user access restrictions that allow all information and functionality to be restricted based on the user. The users must be authenticated and every user must use a separate account. [10]

If the records include psychiatric or hereditary information that part must be protected with a separate permission request system if the records are to be used in other medical fields. This request does not apply to possible medication information or possible critical risk information. [10]

When services are outsourced to another entity the requirements must be upheld also at the provider and a written contract must exist between the entities describing all requirements for security. [10]

Every service provider in social services and medical fields must gather a log of all activities regarding stored information. The log must include identification of the data that were accessed, which service provider accessed them, which provider's data were accessed, why they were accessed and when. This log information must be destroyed when they are no longer pertinent for making sure all data are accessed according to the law. [11] There is however a minimum time of 12 years for which the log information must be stored securely. [10, 24 §]

All documents must include a digital signature to ensure their immutability and authenticity. When a person is signing the documents there are several requirements for the used method to be legally valid:

- The method must be based on an initial identification, which is done in person by checking identity documents (passport, personal ID card) [12, 17 §]
- There must be a log about the initial identification including required information to verify the identification, what documents were used, possible limitations, and, when using a certificate, information about the certificate authority, whose certificate it is, a unique identifier for the owner and certificate itself, validity period and an electronic signature [12, 19 § / 24 §]
- It must be possible to uniquely identify the person in possession of the identification item
- It must be possible to verify that only the person holding the identification item can use it
- The method is secure enough considering current threats against the used technology [12, 8 §]

Also the providers of authentication services must be trustworthy, which means that all their personnel must be of age, they must not be bankrupt and they may not have any other limitations to their actions. They also may not have a conviction for incarceration within the past five years, or even fined within the past three years for a crime that would make them unsuitable for work in authentication services. [12 9 §]

3 Attacks on Medical Systems

3.1 Attack Prevalence

Many medical systems have been attacked in the past and the attacks will surely just get more common since the digitalization of information provides more and more possibilities. In some places records are still on paper in cardboard folders and acquiring them requires physical entry but more and more of these are transferred to electronic systems and can be accessed from anywhere. In the United States in 2008 only 9% of hospitals had electronic health records in use whereas in 2014 the figure had gone up to 75%. [28]

The year 2016 has been deemed as the “year of data security” in healthcare, at least partially due to the issues that happened in 2015. The U.S. based Ponemon Institute specializing in research on privacy and data protection released a report stating that more than 90 percent of healthcare organizations (HCs) in their study had a data breach and more than 40 percent had more than five data breaches during the past two years. The study included 90 “covered entities” meaning “health plans, health care clearinghouses and health care providers who transmit any health information electronically” as well as 88 “business associates” (BAs) defined as “a person or entity that performs services for a covered entity that involves the use or disclosure of protected health information.” [19]

According to the report criminal attacks are a major cause for the information breaches and are up 125% compared to five years ago [19]. If records were electronic and encrypted this could be mitigated, but it is impossible to have everything converted to electronic records immediately, if ever. Then again, using electronic records will have another set of problems and possibilities for attacks.

The report further states that web-borne malware attacks caused issues for 78% of HC organizations and 82% of the BAs. 65% of HC organizations and 87% of BAs experienced electronic information-based security incidents over the past two years. What is surprising and concerning is that still only 40% of the HC organizations and 35% of the

BAs are concerned about cyber-attacks. Over half of them say they don't have enough budget and resources to counter these attacks. [19]

The most worrying security threats listed in the report amongst the HC organizations were employee negligence, cyber attackers, use of public cloud services, mobile device insecurity, and "bring your own device" activity (70%, 40%, 33%, 32%, and 29% respectively of respondents). The most reported incidents in the HC organizations were lost or stolen devices, spear phishing, web-borne malware attacks, exploits, and SQL attacks (96%, 88%, 78%, 54%, and 38% respectively). Many of these could be handled with proper information system planning and development. [19]

The Ponemon report shows that the types and amounts of attacks have changed drastically in the recent years. Previously the most attacks were through loss or theft, or other traditional methods. As shown in Table 1 in 2015 the attacks moved strongly to the electronic world and affected almost hundred times more people. Most of the individuals affected were also in only a few major attacks, like Premera Blue Cross and Anthem hacks which leaked 11 and 78.8 million customer records respectively. [22]

Medical identity theft in The United States has nearly doubled to 2.3 million adult victims between 2009 and 2014. [19]

<i>Type of Breach</i>	Individuals Affected 2014	Individuals Affected 2015
<i>Hacking or IT Incident</i>	1,768,630	111,803,342
<i>Loss or Theft</i>	7,237,157	750,802
<i>Other</i>	3,504,350	646,243
<i>Total Affected</i>	12,564,137	113,200,387

Table 1 Individuals affected by attacks in 2014 and 2015 [22]

3.2 Reasons for Attacks

The reasons for attacks are not usually to acquire medical information of individuals but to enable the attackers to use the information to either gain access to other systems or be able to claim to be another person. Identity theft and insurance fraud are possible with the information that is held in medical information system records. Attackers may be able to even acquire medical equipment or medicine by claiming to be someone else. These can then be sold for profit. [23]

A single credit card number may be worth \$1 to attackers, but health credentials can be sold for up to \$10 each based on information gathered from the underground exchanges by PhishLabs, a cybercrime protection company. [23]

3.3 Ransomware Attacks

There have been attacks known as *ransomware attacks* to hospital systems where the attacker encrypts everything with their own key and asks for the target to pay to get the key. [18] If the systems are properly backed up this might not be needed and the systems can be restored without issues but the attacker still succeeded to penetrate the system. Sometimes the hospitals even have to pay since they do not have proper backups in place. [20]

3.4 Cost of an Attack

The average cost for a single stolen patient record is 154 USD and for HC organizations it can be as high as 363 USD [19]. Since one attack can easily compromise thousands of records the costs rise quite quickly to very large sums.

The Ponemon study estimates that the cost of data breaches in the U.S. is around \$6 billion to the whole industry and the average cost of a data breach was \$2.1 million for the healthcare organizations. For the business associates a breach cost on average \$1 million. This clearly shows that security is important not only to protect the people whose information is being handled but also to protect investments and businesses. [19]

Many victims of medical identity theft report the cost of \$13500 on average to restore their credit and reimburse HC providers for fraudulent claims. Still almost two-thirds of the HC organizations and BAs have no protection services for their customers whose information has been compromised. [19]

3.5 Assumption of Safety

Encryption and security is used as a sales technique by many companies. Healthcare providers may not have enough knowledge of security internally so they trust the providers in their promises of security. Sometimes the advertising can be very misleading and cause even federal cases to be filed.

Henry Schein Practice Solutions, a leading provider of dental practice office management software, marketed their system as compliant to the HIPAA requirements for patient data encryption but actually used a simpler method than the required AES cryptography. The company had to pay \$250,000 in fines and notify all their customers that their system does not provide the advertised security required by HIPAA.

4 Authentication Methods

4.1 Available Methods

There are several methods to authenticate a user into a system. In addition to the most common username and password combination, additional methods can be used to verify the authenticity of the user. Some of the methods can be used beforehand, for example username and password, and some are used to monitor the user while using the system. [5]

Transaction authentication requires additional data to compare to the actions of the users at the moment and might cause false positives. Banks use this method to track transactions. For example, if a person's credit card is suddenly being used at a location distant from the owner's place of living the bank can monitor the transactions and contact the card owner to make sure the card is not misused. [5]

Biometrics includes fingerprints, retina scans, voice recognition, and face recognition. Voice recognition is the only simple method possible to use remotely without any additional hardware, but it might be possible to fake by recording voice and might suffer from problems with phone line quality. [5]

Tokens are physical devices used by themselves or with a username/password authentication. Tokens are often used in buildings to allow people through restricted access areas. They can also be electronic devices, such as one-time password generators. A key to one's house or car is also a simple token. [5]

Multi-factor authentication is a combination of two or more authentication methods. Usually methods from different categories are chosen to enhance security. The groups include something known (password), something possessed (token), and something inherent (fingerprint). [5]

Out-of-band authentication uses a completely separate system to authenticate the user. For example, while logging in via the Internet the user gets a second verification via SMS or a phone call. [5]

4.2 Token Authentication

Token authentication is a simple system with variable security and efficiency. The simplest token is a key or electronic badge that the user uses to gain access to the system. Usually this works mainly in physical things like buildings and vehicles. If the key has been designed to be safe the system may be safe. However, the key may be easy to copy after seeing one or one key might fit several buildings or vehicles. It is not possible to provide access to someone else without physically giving them the key or a copy of one. In information systems external hardware is also required to use them.

Sometimes tokens require a PIN code to use them, for example bank cards or smart-cards used to log in to computers. This provides another layer of security to just acquiring the token, or a copy of one. A fingerprint scan may also be used in addition to an access badge to get into buildings.

4.3 Username/Password Authentication

The most used authentication method is username and password. This may seem to contain two pieces of identification data, but only one of them is usually known only to the user. As seen in Picture 1 in many systems the usernames may be shown to other users, or is the user's email address, so practically only the password is used for authentication.



Picture 1 On Twitter the usernames are shown to everyone

If the user's password is compromised anyone can log in to the system imposing as the user. If the original user detects this they may change their password and force the impostor out. The impostor could also change the password, or set up a secondary password retrieval method within the system which allows them access at a later time.

Many systems show the user little or no information about their past actions so they may not even realize their account is being used by someone else. Some systems have information about last access time, but it might not be clearly visible or the users don't check it. Others may send a message at least when a login is detected from a new device or location. Picture 2 shows the Facebook login history giving the possibility of checking whether someone else has accessed the account.

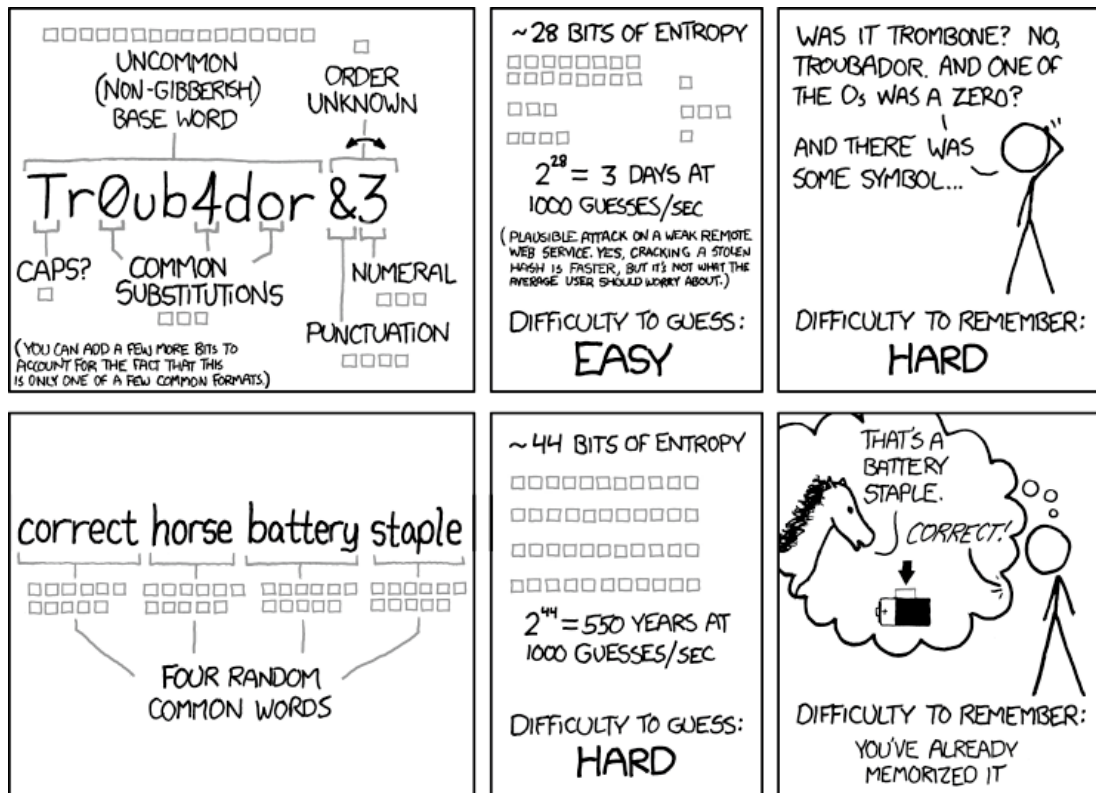


Picture 2 Facebook last login history

Passwords can be made to expire after a certain age. This will help in a situation where someone else is using the password without the user's knowledge. The impostor will have to find out the new password before being able to log in to the system. Many times passwords age quite rarely, since it would be inconvenient to memorize a new password all the time and would cause the users to either write down the passwords or use a very similar password to the previous one.

Strong passwords are very often promoted, but usually in a false way. The common misconception is that a strong password must contain numbers, capital letters, special characters etc. This will make it hard for the users to remember, which will cause them to write it down and make it an attack point. Much better passwords are for example long sequences of words that the user can make into a story and remember, maybe with some letter/number substitutions.

Picture 3 presents a famous example of this technique is presented as an xkcd comic.



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Picture 3 xkcd comic #936 promoting secure passwords

Another suggested strong password method is to use a single gibberish part which the user can memorize and add to that something different for each place where the password is used. This way it will be unique, but easier to remember.

4.4 Two-Factor Authentication

4.4.1 Definition

Two-factor authentication employs a second method to authenticate the user, usually in addition to the username/password pair, some of which were introduced earlier in this section. The second factor must be something variable or something that has to be physically obtained.

By using a second factor in authentication the user's identity is still safe even if an impostor finds out their password. This still doesn't mean that the user can be careless about their password. If it is revealed to others, the two-factor authentication becomes just one-factor authentication and no additional security is present.

4.4.2 Two-Factor Using SMS

Using SMS as a second factor authentication is actually both *out-of-band* and *two-factor* since the second piece of information is relayed via a different route than the password is sent. This method also allows only the person who has access to the device receiving the messages to log in, so partially fits into the *something possessed* category. However, sometimes SMS is sent also to other devices; for example, if the user is using Apple iPhone and has message delivery set up the SMS might also be transmitted to another device linked to the user's Apple account that may be in the possession of an impostor. There is also another attack vector via the security of the servers handling the SMS transmitting to these other devices.

Sending messages via the mobile network also imposes some costs, at least to the sender, and mobile networks are not always available. Even in larger cities there may be areas where mobile coverage for some operator is poor, or the user might be in a basement as many laboratories are.

The National Institute of Standards and Technology has recently deemed two-factor authentication based on phone or SMS as deprecated and will not consider using them as secure or advisable in the future. [29]

4.4.3 Two-Factor Using Pre-Generated Passwords

Many banks use a pre-generated set of passwords as the second factor authentication online. The passwords are printed out and they may be used sequentially or randomly. They may also be one-time or multiple-use passwords. Usually the password sheets will expire after a given time regardless of their use.

One method of generating these passwords is using a *counter-based* one-time password algorithm. This way the server does not need the whole password sheet and only needs to keep track of the secret key and the current counter value.

The major problem with these password sheets is obvious: if an attacker can have access to the sheet even once, they can copy the contents and log in any time after that using the constant pre-generated passwords. If the passwords are used sequentially, the user may keep track of which password is next and notice if there is a discrepancy. If they are used randomly, and especially multiple times, the user has no way of knowing if someone else has used their account with these passwords.

In addition to this problem the password sheets must be given to the user securely. This may cause extra costs and trouble compared to some more secure methods.

Regardless of these issues this method is used for example by major Nordic banks. In their case the login has actually three parts of secret information: login ID, password and the password sheet. Nobody else should know any of these, since banking is completely private service as opposed to collaborative services where the user's login name may be visible to others.

4.4.4 Two-Factor Using One-Time Password

One-time passwords are usually retrieved from devices without any connection to any network so they will work everywhere. There are specialized devices that only give out passwords and there are also applications for multifunction devices. The former may be more secure since to get any data out of it the attacker has to disassemble the device. Using mobile applications it is theoretically possible to get data out of the device via other installed applications or security holes in the operating system. Therefore the OTP application has to take these into consideration.

Special devices have to be present when the user logs in so they add to the items the person has to take care of. Users usually have mobile phones with them anyway so using an application doesn't require carrying anything extra. Also the application can be easily

updated and changed, unlike a hardware solution and is a lot cheaper to produce and distribute.

One-time passwords are generated with a function that takes a secret key and some variable that causes the result to be different each time. The secret must be known to both sides of the authentication and is the weak point. It must never be disclosed to anyone and preferably even the user shouldn't know it. [2]

The algorithm used to generate the passwords may or may not be public. Usually the algorithms are public since it helps determine their strength and possible weak points. An authentication method should be strong based on its algorithm and secret keys, not based on obscurity, i.e. nobody knowing how it is generated. The same applies for physical locks: they are patented and usually anyone can get a hold of the patent to read how it works. The difficulty is in creating keys that would fit that specific lock you want to open.

There are two main types of one-time passwords in use currently: *counter* and *time* based [2, 3]. Counter based method uses an incrementing counter as the input and time based uses the current time. Counter based methods may get out of sync if the user requests passwords but doesn't input them. This causes the requirement of *resynchronization* between the server and the generator. Time-based generator has no such problems, but it requires both sides to have synchronized clocks. This is usually no problem since mobile phones synchronize their clocks automatically, as do servers.

5 History of Cryptography

The field of cryptography is a very old one. It could be said that in the beginning just writing something down was a type of cryptography since very few people could read or write. Still anyone who learned to read and write could have access to these messages so normal language and writing systems didn't provide a proper way of hiding the information.

5.1 Ancient Encryptions

Almost four thousand years ago in Egypt some scribes used hieroglyphs differently to try to conceal the text from others who did not know the key to the writing. In ancient Greece texts were written on a long tape wound around a stick and the recipient had to have the same width stick to "decode" the text. Ancient Romans used the Caesar Shift cipher which shifts the letters up by a certain amount, usually three. This way A became D, F became I and X became A. Reading the message back required moving every letter back by the agreed amount.

Especially the two latter methods are very inefficient. There are only so many thicknesses of sticks or numbers of shifts one can choose from so the message can be decrypted very quickly using *brute force* method, meaning just trying every combination and seeing which one fits.

A bit more secure way to switch letters around would be to randomly select corresponding letter pairs. Then A might be R, F might be B and X might be Y. This way the attacker would have to try on average $\frac{26!}{2}$ combinations, which is already $2 * 10^{26}$ possibilities. However, there are attack methods to reduce this complexity. For example, in English language the most common letter is E so the attacker has to count how many times each letter is present in the encrypted text and can guess that the most common should be replaced with E. Apply this to a few of the most common letters and the message may start to unravel. This technique is called frequency analysis.

Also it is possible to assume that any letter by itself is probably A since there are no other single-letter words in the English language. Also two-letter words are probably some of the most common, like *if, of, to, an, in, on* and so on. With this information it is quite feasible to decrypt simple character substitution encryptions.

5.2 Middle Ages

In the Middle Ages many states had ambassadors in different countries and it was important to send and receive messages without the information getting into the wrong hands. One of the used technologies was to switch the cipher mid message, for example after every four words, to make it harder to use any frequency analysis on the whole message.

In the early 1900s a nearly unbreakable invention was made: the one-time pad. This technique has a list of ciphers that are used only once and never again. If the enemy decrypts one message they have no possibility to decrypt the next one since the cipher has changed and will not be used again in any of the future messages. By itself of course this is not enough to ensure the message wouldn't be decrypted. The encryption technique itself must also be of sufficient strength to make brute force decryption infeasible.

5.3 Modern Times

An interesting way of cryptography was to use Navajo Indians during the World War II to send messages. Their language was not known to others, so they could send messages in their own language and translate them in the other end. This, of course, goes more into the "security by obscurity" category and is not really a cryptographic method, but was still very successful during the war and wasn't broken at the time.

Also used in the World War II was the German Enigma machine. It consisted of several rotating disks that would change position on every character and depending on the starting conditions would produce a strongly secure varying output. The British tried to break

the code for a long time and finally Alan Turing built a machine that could calculate several combinations at the same time and would possibly brute force the code. Even that didn't work by itself due to the amount of different combinations.

While analyzing some messages the British had managed to decipher they noticed that many messages were regular. Some weather reports always contained predictable information. Some messages always had predictable text in them, for example "heil Hitler" in the end. With this knowledge they could make the machine try different combinations until it found the known text and then use that key to decipher any message sent before a key change happened. This kind of attack is called known plaintext attack.

The actual modern cryptographic methods use a mathematical method and a key or a set of keys to encrypt the message. Their strength is in the algorithm and the number of possible key combinations used. They can utilize a single key to encrypt and decrypt or they can use separate keys, often called *public key cryptography*. It can be used in two ways: if the message is encrypted with the *private* key then anyone who has the *public* key can decrypt it and be sure that the only person having the private key wrote it. It can also be used by anyone having the *public* key to encrypt a message and then only the person with the *private* key can decrypt it.

6 Cryptography Basics

6.1 Strong and Weak Cryptography

In cryptography the terms strong and weak are used often. The meaning is not simple and an algorithm deemed strong may suddenly become weak after new attacks or devices are designed. Therefore nothing can be said to be strong forever, only for the time being.

If a cryptographic system is strong, it means it is hard to break. An encrypted message should be very hard, or preferably theoretically impossible, to decrypt without the proper key. For a hash function it should be hard or impossible to find a different message m' that produces the same hash value as another message m .

One simple value to determine the strength of an algorithm is the number of bits used in the key. If an algorithm uses 4 bits in a key there are only $2^4 = 16$ possible keys. If testing one key takes one second, we will on average find the correct key in 8 seconds. If an algorithm uses 64 bits, we will have to test on average $2^{32} = 4294967296$ keys, taking 136 years at the mentioned speed.

Of course computers and especially dedicated hardware can test keys a lot faster and the speed depends on the algorithm used. The simplest algorithm is exclusive-or which is implemented in hardware on practically every processor available and therefore is very easy and fast to implement. Usually it is only used as a part of the algorithm and the whole algorithm consists of more complicated mathematics.

For the SHA256 algorithm used in this work the calculation takes about 12 clock cycles per byte from a regular Intel/AMD desktop processor when messages are at least 576 bytes long. [17] For shorter messages the initialization overhead causes more cycles, but the absolute time is of course shorter. This speed means that with a message of 4096 bytes it would take 49152 clock cycles to try out one key. Given a four-core processor running at 3.5GHz this would mean that 284830 keys could be tested per second.

Therefore the 64bit key length would cause an average calculation time of only four hours on one machine and in the worst case only eight hours. This obviously makes the algorithm weak.

If we use 256bit key, we will require on average $2^{128} = 2.4 * 10^{38}$ tries. This on the given desktop computer would take 2^{96} times more, resulting in a calculation time of $3.3 * 10^{29}$ years. This means that even by using an army of computers a brute force guessing method is not in general usable to break this algorithm.

Specialized hardware can of course try out keys even faster but considering the time values in question even a speedup of a billion wouldn't help to find a key in a reasonable time.

6.2 Governmental Recommendations

The National Institute of Standards and Technology (NIST) lists algorithms that should or shouldn't be used, their comparable strength based on key sizes, how often keys should be changed and other important information. [16]

European Union Agency for Network and Information Security (ENISA) has a similar report for recommendations on cryptographic algorithms and key sizes. In the 2014 report they recommend SHA-2 family for future use, as long as at least 256bit version is used. The SHA224 version is not recommended since it has less than the recommended minimum of 128 bits of security. [15]

Other recommended hashing algorithms are SHA-3 (with at least 256 bits) and Whirlpool. [15]

The HMAC algorithm is deemed safe for future use also but since it can use several hash functions internally the actual algorithm must be chosen properly. HMAC-SHA1 and HMAC-MD5 should not be used in any new implementation due to the insecurity of the hash algorithms. [15]

6.3 Security by Obscurity

A common problem in cryptography or other security is to rely on *security by obscurity*. This means that the details of the encryption or other security methods are not disclosed and trust is in the difficulty of finding out this information. This does not provide good security, since when the method is found out, it may not be very strong and can be circumvented easily. It can be related to using a special hiding place for a key to your house and trusting no outsiders will know where to find it.

Regularly used cryptography methods are publicly explained and anyone can analyze them. This way the strength relies on an actually proven method and usually only the keys must be kept secret. This also means that anyone can try to devise a method for calculating the encryption or decryption values faster than before.

6.4 Encryption vs Hashing

Many times people talk about encryption when they are actually using *hashing*. Hashing is a cryptographic method, but it is very different from encryption in usage and function. Both methods use keys to ensure authenticity and/or secrecy but in a different way. [30]

Encryption is taking a message M and using some function F to produce a message M' . There always exists a function F' that can be used to convert M' back into M . This means that encryption is a two-way function and can be used to hide or secure information from others. [30]

Hashing is taking a message M and using some function F to produce a hash H , usually of a small size compared to the message. The operation is one-way, meaning there is no inverse function F' that would produce M from H . There may also be many M s that produce the exact same H . There are ways to try to infer the original message, or any of the messages that would produce a certain hash, but the hash function and its applications should make this very hard. Hashing is used for example to make sure a document

is not tampered with in-transit since altering it would make the hash result not match the one sent with it. [30]

6.5 Hash Functions

Hash functions are used to transform variable sized input data into a usually fixed size output [6]. The output is shorter in length than the input and can be used for example to verify the authenticity of the input or to search values in a large dataset faster. Since the input is variable and output is fixed size, several input values will produce the same output value. This is called a collision and means that unlike compression or encryption methods hash values are not reversible. [31]

Hash functions need to have certain features to be considered strong:

- *Uniform distribution*, which means that for any given input, any output value is as probable as another. This reduces collisions and makes the function less predictable. [32]
- *Avalanche effect*, which means that when any input bit changes, the probability for any output bit to change is 0.5. This makes the function less predictable and finding a collision much harder. [32]

It is important to reduce the probability of a collision. If an attacker finds a way to produce a collision, they can find another input data that produces the same output as another. If the hash function is used to provide a digital signature, the attacker can change the input data into another and still produce the same output. If this happens the hash function is not suitable for digital signatures and must be discarded. [32]

If the hash function is used in providing faster search results inside large datasets the consequences of collisions are not as drastic as for digital signatures. In search hash functions data is usually divided into sections based on the hash function and several input data will produce the same output regularly. The hash function is not used to validate the data, only to find in which section the data resides. If the hash function produces

collisions more often, the only adverse result is that more data is saved in certain sections and finding it takes more time. [33]

Even if a hash function is considered safe when introduced, it may be found lacking later on. MD5 is one algorithm that was very widely used, but later on cryptanalysis found ways to produce collisions on demand and the algorithm is very fast so brute force attacks are feasible. SHA-1 algorithm was also previously a standard but is now deprecated. Old implementations may still rely on these algorithms, but all new applications must use other hash functions. [34]

6.5.1 SHA Family

In 1995 the National Institute of Standards and Technology defined a Secure Hash Algorithm 1 that provides hashing of any data into a 160bit result. The algorithm was widely used but it has also been proven to have possible collision problems. Therefore in 2001 a new family of hash functions were defined in Secure Hash Standard [1]. These are called SHA-2 and contain such algorithms as SHA-256 and SHA-512. They provide larger output values, which has to be taken into account when implementing methods using them.

6.5.2 SHA256 Algorithm

The SHA256 algorithm is a hash function that has internal state of eight 32bit words and operates on blocks of 512 bits each. Each block is run through 64 cycles of the mathematical formula and each time a different constant is used. The end result will be the concatenation of this internal state to produce a 256bit hash value. [1]

Currently there are no known actual attacks against the SHA256 algorithm, but theoretical weaknesses are being assessed and some possible problems have been identified, like preimage resistance (when value x hashed produces the same value x), which might make it easier to find collisions for the algorithm. [14]

6.5.3 SHA512 Algorithm

The SHA512 algorithm is similar to the SHA256 algorithm but operates on eight 64bit words internally, running the algorithm 80 times and producing a 512bit hash value. The apparent security strength of the algorithm is twice that of SHA256 and should be used when better security is required. [1]

The algorithm has the same theoretical attacks as the SHA256 algorithm, but due to more cycles it is more secure.

6.6 Hash-Based Message Authentication Code

A *hash-based message authentication code* (HMAC) is a way of generating an authentication code for any sized data to verify its authenticity. The algorithm requires a hashing function (like MD5 or SHA256) for operation. The main algorithm is $HMAC(K, m) = H((K \oplus opad) || H((K \oplus ipad) || m))$, where H is the hash function, m is the message, K is the secret key, $||$ is concatenation operator and \oplus is exclusive-or operator. Constants $opad$ and $ipad$ are outer and inner padding values consisting of repeating bytes of 0x5c and 0x36 respectively. [13]

When applied to the message the algorithm will generate a single hash value as a result. The value will be different if the message or the key is changed. To validate its authenticity the secret key must be known.

The reason why there are two hashes inside each other is because for a single hash of type $H(m||K)$ there might be a way to find a message m_2 that provides the same initial conditions as the original message. Also using $H(K||m)$ would allow the possibility of lengthening the message with other data until a collision is found. [13] There have been suggestions to use the algorithm $H(K||m||K)$, possibly with two different keys, but this also has been shown to have possible security issues. [14] Using two hashes one inside the other makes it very hard to find a collision.

6.7 Random Numbers

6.7.1 Random Number Generation

Cryptography relies heavily on random numbers. The keys generated to secure data must be very hard for others to guess or generate, which is why so called *true random numbers* should be used. Usually this is not feasible and algorithms have been invented to produce *pseudo-random numbers*. The main difference between true and pseudo-random number generators is that the former uses some inherently random source as a seed for the number generation whereas the latter uses mathematical algorithms.

The properties for random number generators are:

- *Efficiency*: how many random numbers you can get from the generator in a specific time [8]
- *Determinism*: how easily can the sequence of numbers be repeated [8]
- *Periodicity*: how many numbers can you get from the generator before they repeat themselves [8]

6.7.2 Pseudo-Random Number Generators

Pseudo-random number generators are *efficient*, since they only rely on a mathematical formula. They can generate huge amounts of random numbers, but they are *deterministic*, since they rely on a fixed algorithm and seed values. They are also *periodic* and will repeat the same numbers after a certain amount of numbers produced. [8]

6.7.3 True Random Number Generators

True random number generators usually have *poor efficiency*, since they need an external source which is sampled. There can only be a certain amount of random data sam-

pled per time period. The reason why they are still preferred is because they are completely *nondeterministic* and *aperiodic*. This means that they cannot be guessed and even if you took billions of numbers, the sequence you get will never repeat itself. [8]

The sources for true random numbers can include the key typing or mouse movements of a computer user. These sources are not really truly random and don't produce data at all times, so they are not always useful. Also in server systems there are no users to actually generate data to use. [8]

More usable sources are, for example, atmospheric noise and radioactive materials. The decay of radioactive sources is completely unpredictable, so the detectors will get signals at truly random times. These signals can then be used to generate random numbers. [8]

Some computers have processors that include an analog circuit that inherently produces noise, which is amplified and used as a source for random numbers. This way it is possible to generate a better randomness than would be feasible with just mathematical algorithms. One of these systems is Intel's Ivy Bridge.

6.7.4 Ivy Bridge RNG

The RNG in Ivy Bridge consists of a self-oscillating digital circuit with feedback. The circuit has two latches which are in a metastable state and will output 0 or 1 depending on the thermal noise in the system. It is capable of working around 3GHz frequency, which means it can produce 3 billion random bits per second. [9]

These bits are fed into another system working at 800MHz, which collects them into 256bit random numbers. These numbers are subjected to a "health check" that determines whether the randomness is good enough. Tests count the appearance of certain bit sequences in the number (for example, the number of 1s) and if the value is not within a certain range, the number is discarded and a new one is taken. About 99% of numbers will pass the test. After the test there are some more processing, testing and reseeding before the actual number is presented to the requesting application. [9]

Intel claims the RNG to have at least 0.5 bits of entropy per bit, which was tested independently. [9] This means that the generated random numbers have at least half the randomness of a truly random source.

7 Implementation

7.1 Platform

The two-factor authentication system was implemented in two parts: a server system and mobile client applications. The server system was implemented using Microsoft's ASP.NET MVC framework in C# language to integrate with the existing system. The mobile client applications were implemented with Java for Android and Swift for iOS devices.

The server side system can be adopted to work on any authentication method since it is not tied to a specific framework. It is used as a secondary part after the regular password authentication and is therefore completely separate. The only thing needed is for the server-side system to keep track of the success of password authentication and invoke the one-time password system before actually letting the user in.

The mobile applications are implemented so that they work on a large variety of Android and iOS devices. The functions needed for cryptography are available from very early versions of these platforms so in practice all devices currently in use should be able to use the application. Also due to the reason for the whole system being security it is not advisable to keep very old mobile operating systems in use.

7.2 Chosen Method

The chosen method was time-based one-time password as defined in RFC 6238 [3] but substituting the use of the possibly insecure SHA-1 algorithm with SHA-256 and without any master secret. Also the truncation method was modified to take into account the larger output size of the hash function. No global master key is defined so if a single secret key were to be disclosed, other keys would not be affected.

The password is derived from a shared secret K , and a time value T that are fed into an SHA-256 HMAC algorithm. The value T is derived from the current UNIX time by dividing

the value with a predetermined time constant D . This constant will determine the time window in which the algorithm will provide the same value before moving on to the next.

Having a large D will cause the password to remain the same for a longer time and also affects the time the server will accept the password. The constant D was chosen to be 20 seconds since that is a reasonable time for the user to input the password and it to be sent to the server.

When the server validates the password, the same method is used, but there is also a tunable constant C that determines how many past time slots are accepted. Since the password is valid within a discrete D second timeslots, the user may have received the password at $D-1$ seconds and the validity period has passed before the password is sent to the server. Therefore the server will check for the current password at the time of receiving and also C previous timeslots.

Having a large C will allow the user to input the password at a longer timeframe and also accounts for slightly non-synchronized clocks, but will allow for an eavesdropper to use the same password for a longer time. Therefore the constant C was chosen to be 1 to allow on average $D*1.5$ seconds of time to send the code to server, i.e. 30 seconds.

The 256 bit result of the HMAC algorithm is fed into a truncation method, which will result in a 32 bit output. This value is further truncated into an 8 number integer code ready to be used. This truncation is done so that the code is easier for the user to enter into the system. If we used the whole 256bit result the user would need to enter a longer or more complicated code into the system making it more cumbersome to use.

7.3 Requirements

The RFC 6238 [3] defines seven requirements for the algorithm:

1. Both sides must be able to derive the current UNIX time. As long as both the server and the mobile device have their clocks synchronized to a reliable source

they can determine this. Also by using the D and C values slight differences between the server and the mobile device clocks do not interfere with the password generation notably.

2. *Both sides must share the secret or have an algorithm to generate it.* In this implementation the secret key is shared manually from the mobile device to the server so both have access to it.
3. *The algorithm must use HOTP as a key building block.* The algorithm is directly derived from RFCs 4226 and 6238 and uses HOTP as the basis.
4. *Both sides must have the same constant D.* The value of D is hard-coded and the same on both sides.
5. *There must be a unique secret for each provider.* The mobile application generates a new random secret key for each user and system. There is a probability of the secret key being the same, but the probability is about one in $1.46 * 10^{48}$ due to the random secret being a 160bit value.
6. *The keys should be randomly generated or derived using key derivation algorithms.* The implementation uses platform-provided random number functions that have been designed to be used in cryptographic systems.
7. *The key should be protected against unauthorized access and usage.* The secret key is encrypted using an unknown length PIN code and there is no way to determine the secret key without knowing the PIN code.

Therefore all requirements placed by the RFC 6238 have been met and the algorithm can be assumed to be secure and implemented in the way intended.

7.4 Implementation in Short

The client application generates a secret key by using cryptographically strong random numbers. The secret is shown to the user in base32 encoded form for convenience and the user saves this secret key on the server via an encrypted channel. The secret key is saved on the client encrypted with a user-defined numerical PIN code. The secret key is never shown to the user again on the client application and the application does not know the PIN code. This way if the application data storage is compromised, it contains only opaque encrypted data and there is no possibility to determine which PIN code should be used to decrypt it and retrieve the secret key.

When the user requires a one-time password to access the server they input the PIN code and the secret key is decrypted and used for calculating the time-limited code. The code is then input to the server which verifies the validity by calculating the same values with their copy of the secret key.

7.5 Used Algorithms

The algorithms used in the implementation are well-known and nothing new was invented. This was to make sure there are no weak links due to badly designed algorithms.

7.5.1 Base32 Algorithm

The base32 algorithm converts input binary data into human readable text using 32 alphanumeric characters. This means that for every five bits of input data one eight bit character will be produced, making the output data 60% larger than the input. The character set used is *0123456789ABCDEFGHIJKLMNPQRSTUVWXYZ*. The character set specifically skips certain letters like *O* and *I* due to the possibility of a mix-up with numbers *0* and *1* as well as letters *L* and *U* as unneeded to reach 32 characters.

If the input data size is not divisible by 5 bits zero bits are appended. The implementation could also add a checksum into the output to make sure the data is input correctly but in this case this was not considered important.

7.5.2 Truncation Method

The truncation is used to convert the resulting 256bit hash result into a more user-friendly one-time password. The truncation method in the RFC 6238 [3] is meant to be used for 160bit values so I altered it to allow the usage of the whole 256bit value. The algorithm uses a part of the hash result as an offset to the hash to take a 32bit part of it. Using it this way ensures that the result is as random as it can be and doesn't make it easier to determine.

Example of the algorithm: [Figure 1]

1. The 256bit hash value received from previous steps:
cd4dda3993d5e1756c65f357da2c320fbfd2856107254f1f6499d0c72acff68c
2. Take the four lowest bits of the first byte (*cd*) which is **d** or **13**
3. Take the three lowest bits of the tenth byte (*65*) which is **5**
4. Add these values together to get **18**
5. Use this as an offset to the hash and take four bytes starting from position 18, getting the value **d2856107** or **3531956487**
6. Take modulo 100000000 of this number to get **31956487**. This is the one-time password calculated from this hash

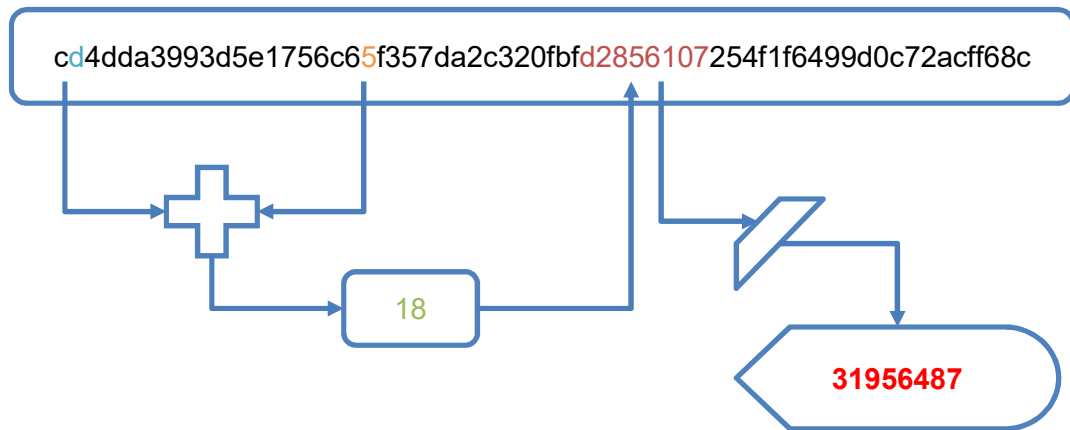


Figure 1 Deriving the one-time password from the hash value

Since the algorithm uses a 4bit and a 3bit number to calculate the offset the possible range is 0-23. Four bytes are taken so the possible byte range is 0-27. This means that the last four bytes of the hash are never used in the formation of the one-time password. This has some effect on the randomness of the result, but it is not assumed to be that significant.

7.5.3 Key Generation

The client generates the secret key using the following algorithm [Figure 2]:

1. A 160bit cryptographically secure random number **N** is generated on the client via system provided functions (*java.security.SecureRandom* on Android, *SecRandomCopyBytes* on iOS).
2. A user-provided numeric PIN code of user-chosen length is run through HMAC-SHA256 algorithm with a predefined constant key to produce a 256bit number **P**
3. The number **N** is xor'ed with number **P** to produce the saved secret **S**
4. The user is shown a base32 encoded version of the number **N** to store into the server

- The saved secret S is stored on the client device

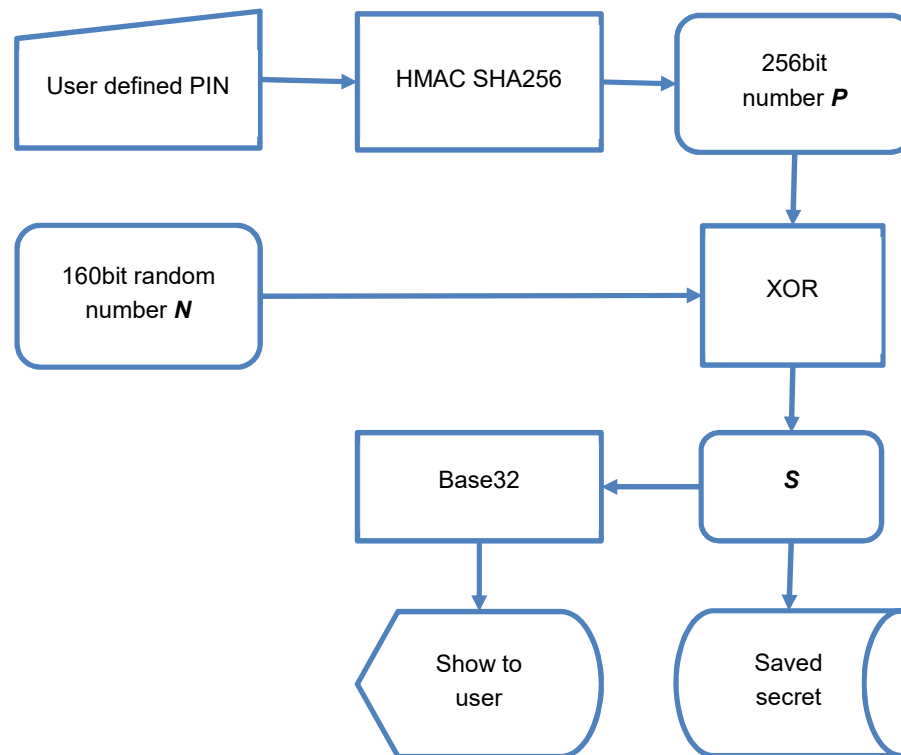


Figure 2 Key generation algorithm

7.5.4 One-Time Password Generation

When a one-time password is required the following algorithm is used: [Figure 3]

- The saved secret S is retrieved from the client storage
- A user-provided numeric PIN code is run through HMAC-SHA256 algorithm with a predefined constant key to produce a 256bit number P
- The saved secret S is xor'ed with number P to restore the secret key N
- Current UNIX timestamp T is acquired from a synchronized clock
- The timestamp T is divided by the constant D to produce the value O

6. The secret key N is combined with the 32bit little-endian representation of the value O and run through HMAC-SHA256 algorithm to get a 256bit value X
7. An offset F is calculated by taking the four lowest bits of the first byte from X and adding them to the three lowest bits of the tenth byte from X
8. A 32bit value Y is constructed by taking four consecutive bytes in big-endian form from X starting from byte F
9. The final shown password W is formed by taking the modulo 100000000 of value Y causing the output to be an eight number string

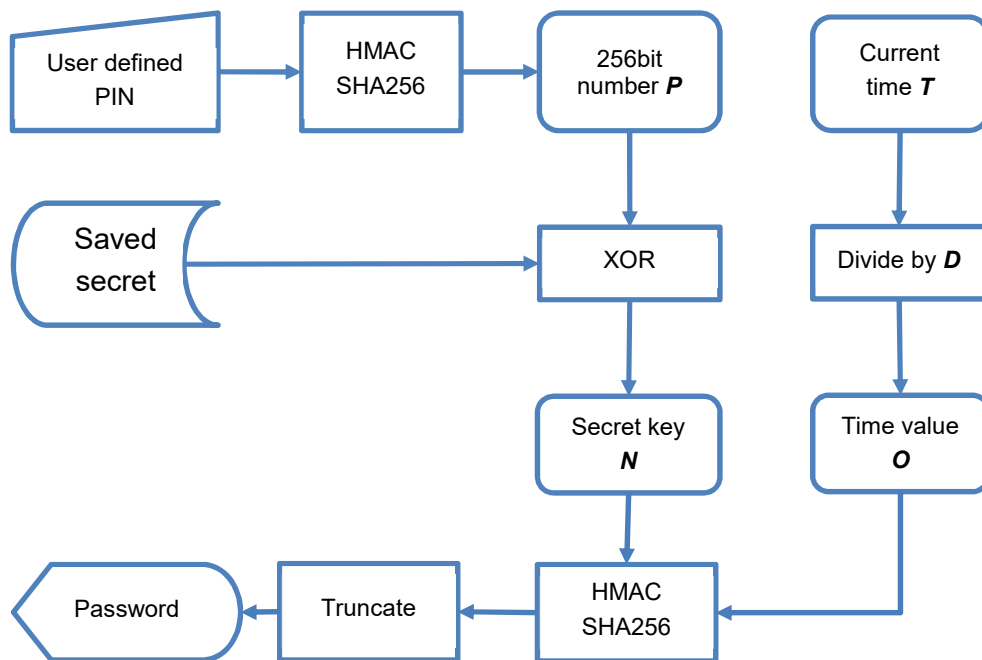


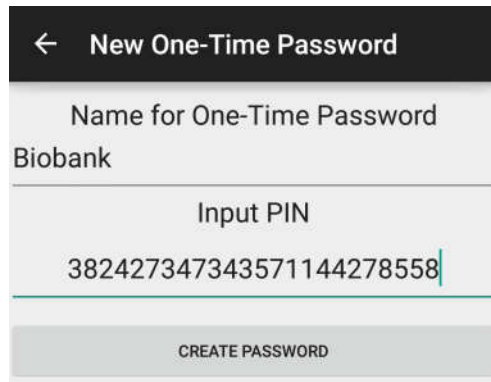
Figure 3 One-time password generation

7.6 User Experience

The mobile application implemented has the possibility to create a new one-time password provider, get a new password and delete an existing password provider.

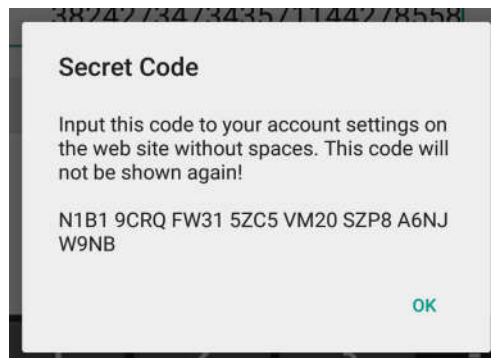
7.6.1 Creating a One-Time Password

1. User selects *Create New Password*
2. User gives the password a name and a PIN code [Picture 4]



Picture 4 creating a new password provider

3. The application presents the secret code for the user [Picture 5]



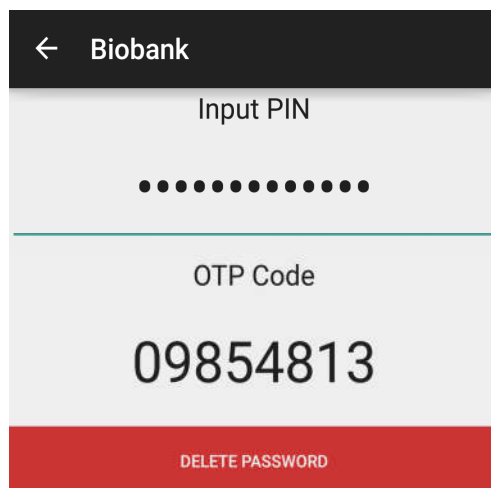
Picture 5 Presenting the secret code to the user

4. The user inputs the secret code to the server via a web page

7.6.2 Logging in Procedure

1. User logs into the server providing their username and password
2. The server asks for the one-time password
3. User launches the client application, selects the desired password and inputs their PIN number

4. The client application shows the 8-number password [Picture 6]



Picture 6 Retrieving the one-time password with PIN code

5. User sends the one-time password to the server
6. Server generates one-time passwords based on the same secret key and predefined time values D and C and compares the generated passwords to the user-given password
7. Depending on the validity of the PIN code and the time taken to send the password to the server the user may or may not be allowed to log in

7.6.3 Deleting Password

When the user wants to delete a password provider they select the password, choose delete and confirm the action. The operation is irreversible immediately and all information of the password are removed from the device.

7.7 Server Implementation

When a user logs in their username and password are first checked. Then if a two-factor authentication is enabled the server will prompt the user to input the one-time password. After receiving the password the server creates a password by itself based on the same secret key and D value and checks if it matches. If it does, the user is logged in. If not,

then C previous time values are used to create passwords and if any of those matches the user is logged in. If none of these match the user is presented with an error message and the invalid login is recorded in the server logs.

If the user inputs a wrong password too many times the account is disabled. This prevents the attacker trying more than a couple of PIN codes on the account.

7.8 Storing Data

The secret key is stored using encryption based on a free-length PIN code on the mobile device. The secret key is exclusive-or'ed with an SHA-256 HMAC of the PIN code. This means that the stored data is an opaque binary blob and there is no way to determine what the PIN code is, for example using known plaintext attack. This adds to the security since the attacker cannot run through all possible PIN codes and get an indication on which one is correct.

On the server the secret key is stored in the database and the database settings define if the data are encrypted on disk or not. When requesting the key it is transmitted in plain text so the connection to the database should be encrypted.

The implementation also doesn't take into account the theoretical possibility of accessing the memory of the server or the mobile device directly and finding the secret key that way. This is due to the fact that if an attacker can do that on the target device they can also probably read the information directly from other sources and security has already been compromised.

8 Security

8.1 Algorithm Security

According to the RFC 4226 [2] the original algorithm with counter has the probability of one in 999,760 of guessing the correct number in one go. With knowledge of previous values and possibility to try older values also the probability of guessing increases by the product of these numbers.

It is also shown that brute force is the best possible attack against this algorithm. There are no smarter attacks against the algorithm. Even if an attacker would see 2^{40} authentication attempts (which would take an impossible long time to gather) they still wouldn't have any better success at predicting the next password.

With the modified algorithm the number of possible keys has been increased by a factor of 100, so the probability is also decreased by the same factor. The values for possible previous keys accepted was set as two, so the probability of finding the correct key is close to one in $5 * 10^7$. Since the server also closes the account after too many wrong attempts on the one-time password there is not much possibility of using brute force to determine the password.

8.2 Server Security

The server stores the secret keys in the database without encryption, since the database must be secured and a breach of security on the server would in any case provide information about the encryption of the keys. If the database security cannot be ensured, the keys can also be saved encrypted and the encryption key will be stored with the application. Then the application server security has to be ensured.

Using a typical lockdown of user accounts after n invalid password attempts for both the regular password and the one-time password will make sure the attacker cannot try out passwords until a proper one is found.

8.3 Client Security

The shared secret key is generated on the client application. The client application has no connection to any network and the shared secret is input by hand into the server system. This way there is no possibility of eavesdropping on the connection between the mobile device and the server. There is, however, a possibility for eavesdropping between the browser used and the server. Since this channel will be secured with industry standard security measures (SSL, TLS), the possibility of acquiring the key is minimal.

To further minimize the possibility of eavesdropping the shared secret key can of course be sent via other methods than using a browser, if required.

Since the secret key is stored as an opaque binary blob and no information of the PIN code is stored the attacker cannot determine which PIN code of the infinite number of possibilities is the correct one.

Since a platform provided random number generators and encryption implementations are used there is very little possibility of an error in the implementation of these. There might be an error but this is deemed very improbable since the functions have been thoroughly tested and many times the implementations are public.

8.3.1 Storage

The client stores only the user-given name of the password and the encrypted secret code. These cannot be used to determine the actual secret key or the PIN code even if they were compromised.

On Android devices the data are stored in the internal memory which is encrypted and by default no other application can access the data stored by this application. On iOS devices the internal memory is encrypted by default and also only accessible by the application itself.

IOS devices are not regularly *jailbroken*, so the possibility of getting data out of the storage is quite small. Android devices are regularly *rooted* or include applications that have access to the data. Since the keys are encrypted without storing the encryption key, even direct access to them will not reveal the actual keys to the attacker.

8.3.2 PIN Code

The PIN code is a free-length numeric code used to encrypt the secret key in storage. Since there is no fixed length for the code, the amount of possible codes is very large, practically infinite. Every PIN code input is accepted as a valid code and a one-time password is output in constant time. This way there is no possibility to guess the PIN code from responses or error messages.

8.3.3 Apple Touch ID

The application will not use Apple's Touch ID fingerprint technology, since this would actually reduce security. The fingerprint locking does not provide any encryption or real security, only information whether the fingerprint or PIN code input is valid. Therefore the secret key would have to be saved unencrypted and might be possible to extract from the device.

8.4 Randomness of the Password

Since the truncate algorithm includes a 32bit value modulo 100,000,000 there is a slight bias towards certain numbers. A 32bit value can have a value between 0 and 4,294,967,295 which in modulo 100,000,000 is 94,967,295. This means that values between 0 and 94,967,295 can occur 43 times (probability of one in 99,882,960) whereas values between 94,967,296 and 99,999,999 can occur 42 times (probability of one in 102,261,126). This bias is very small and doesn't play a role in helping break the password.

Considering this implementation has an eight number one-time password compared to the six number password used in the RFC 4226 [2] the bias is a lot smaller.

8.5 General Security

There is a possibility for someone else to log in during the same timeframe using the same username, password and one-time password during the time period the one-time password is valid. This is highly unlikely and if this would be perceived as a possible threat the server can be easily modified to accept the password only once by storing the recently used passwords for the duration of the acceptance period. Then the first one to log in using a certain time-limited password would invalidate that password from being used again and the attacker would have to acquire a new password.

9 Conclusions

9.1 General Conclusions

The goal of this thesis was to evaluate two-factor authentication methods and implement one of them for real-world use. Several authentication methods were evaluated and explained and a suitable version was implemented for a live environment. The method enhances the security of the system by not allowing anyone to log on with just a static password.

The time-based one-time password mechanism is simple to implement and very efficient in adding another layer of security on top of a regular username and password authentication. It provides the possibility to mitigate the possibility of an attacker getting hold of the user's password and accessing sensitive information.

The implementation succeeded in adding a second layer of security on the system while causing minimal extra work for the users. The mobile application is simple and easy to use and runs on the devices the users carry around every day already. There was no need for specific user training since the users learned to use the system by written instructions.

9.2 Possible Enhancements

There is a possibility that an attacker would get a hold of the one-time password the user uses to log in and would use it to log in to the system. This could be easily circumvented by implementing a recent one-time password memory in the server that automatically blocks any duplicate use. In this case it wasn't seen as an important addition and would've added complexity to the system so it wasn't implemented.

This implementation also restricts the user to only use a single device to produce the passwords. It would be possible to modify the server to store multiple secret keys and try all of them for acceptable codes. This also would've added complexity to the system

and effectively reduced the amount of possible codes since several combinations would have been accepted. Usually the secret key is created by the server and set to the mobile devices by the user so there can be several devices with the same key producing the same one-time password without reducing security.

At the moment the server tells the user if the regular username/password combination is correct before requesting the one-time password. If the server opaquely accepted any regular password and only after getting the one-time password gave a response the security would be a bit better. The attacker couldn't be sure that the regular password is correct and couldn't use any social attacks to administrators or other personnel, for example to get the secret key changed to their key by presenting the correct username and password combination in some recovery system.

The truncation algorithm leaves the last four bytes of the hash unused. It would be simple to add another two-bit value from some part of the hash to the offset to make its range be 0-27 which would mean we would use the whole byte range of 0-31.

To conclude, this thesis presented the requirements for information systems containing personal medical information, the threats that are present and a method to implement a two-factor authentication to enhance the security of such systems.

References

- 1 National Institute of Standards and Technology, Federal Information Processing Standard 180-4: Secure Hash Standard, 2012. <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>, 1.1.2017
- 2 The Internet Engineering Task Force, Request for Comments 4226: HOTP: An HMAC-Based One-Time Password Algorithm, 2005. <https://tools.ietf.org/html/rfc4226>, 1.1.2017
- 3 The Internet Engineering Task Force, Request for Comments 6238: TOTP: Time-Based One-Time Password Algorithm, 2011. <https://tools.ietf.org/html/rfc6238>, 1.1.2017
- 4 U.S. Food and Drug Administration, Code of Federal Regulations, Title 21 Chapter 1, part 11: Electronic Records: Electronic Signatures, 2014. <http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/CFR-Search.cfm?CFRPart=11&showFR=1>, 1.1.2017
- 5 Akilesh Sharma, The Most Common Authentication Methods Used Today, 2012. <http://www.tweakandtrick.com/2012/06/most-common-authentication-methods-used.html>, 1.1.2017
- 6 Wolfram MathWorld: Hash Function. <http://mathworld.wolfram.com/HashFunction.html>, 1.1.2017
- 7 Bret Mulvey, Hash Functions, 2007. <http://bretmulvey.com/hash/>, 1.1.2017
- 8 Random.org, Introduction to Randomness and Random Numbers. <https://www.random.org/randomness/>, 1.1.2017
- 9 Hamburg, Kocher, Marson: Analysis of Intel's Ivy Bridge Digital Random Number Generator, 2012. https://www.cryptography.com/public/pdf/Intel_TRNG_Report_20120312.pdf, 1.1.2017
- 10 Sosiaali- ja terveystieteiden ministeriön asetus potilasasiakirjoista, 298/2009. <http://www.finlex.fi/fi/laki/alkup/2009/20090298>, 1.1.2017
- 11 Laki sosiaali- ja terveydenhuollon asiakastietojen sähköisestä käsittelystä, 9.2.2007/159. <http://www.finlex.fi/fi/laki/ajantasa/2007/20070159>, 1.1.2017
- 12 Laki vahvasta sähköisestä tunnistamisesta ja sähköisistä allekirjoituksista, 7.8.2009/617. <https://www.finlex.fi/fi/laki/ajantasa/2009/20090617>, 1.1.2017

- 13 The Internet Engineering Task Force, Request for Comments 2104: HMAC: Keyed-Hashing for Message Authentication. <https://tools.ietf.org/html/rfc2104>, 1.1.2017
- 14 Khovratovich; Rechberger; Savaliev, Biclques for Preimages: Attacks on Skein-512 and the SHA-2 family, 2011. <http://eprint.iacr.org/2011/286.pdf>, 1.1.2017
- 15 ENISA: Algorithms, key size and parameters report, 2014. http://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/algorithms-key-size-and-parameters-report-2014/at_download/fullReport, 1.1.2017
- 16 NIST: Recommendation for Key Management, 2012. http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf, 1.1.2017
- 17 Virtual Application and Implementation Research Lab: eBACS: ECRYPT Benchmarking of Cryptographic Systems. <http://bench.cr.yt.to/results-hash.html>, 1.1.2017
- 18 Aurora Aguilar, Two Calif. Hospitals hit by ransomware attacks, 2016. <http://www.modernhealthcare.com/article/20160326/MAGAZINE/303269991/two-calif-hospitals-hit-by-ransomware-attacks>, 1.1.2017
- 19 Ponemon Institute LLC, Fifth Annual Benchmark Study on Privacy & Security of Healthcare Data, 2015. https://media.scmagazine.com/documents/121/healthcare_privacy_security_be_30019.pdf, 1.1.2017
- 20 Joseph Conn, Hospital pays hackers \$17,000 to unlock EHRs frozen in 'ransomware' attack, 2016. <http://www.modernhealthcare.com/article/20160217/NEWS/160219920/hospital-pays-hackers-17000-to-unlock-ehrs-frozen-in-ransomware>, 1.1.2017
- 21 FTC, Dental Practice Software Provider Settles FTC Charges It Misled Customers About Encryption of Patient Data, 2016. <https://www.ftc.gov/news-events/press-releases/2016/01/dental-practice-software-provider-settles-ftc-charges-it-misled>, 1.1.2017
- 22 HelpNetSecurity, Why Cybercriminals Target Healthcare Data, 2016. <https://www.helpnetsecurity.com/2016/01/28/why-cybercriminals-target-healthcare-data/>, 1.1.2017
- 23 Reuters, Your medical record is worth more to hackers than your credit card, 2014. <http://www.reuters.com/article/us-cybersecurity-hospitals-idUSKCN0HJ21I20140924>, 1.1.2017
- 24 U.S. Department of Health & Human Services, Summary of the HIPAA Security Rule. <http://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html>, 1.1.2017

- 25 Stanford Medicine, Obtaining de-identified data for research. <https://med.stanford.edu/researchit/consultation-service/scci-compliance-processes/anonymized-data-for-research.html>, 1.1.2017
- 26 Ofni Systems, 21 CFR 11.10(e): Audit Trails. <http://www.ofnisystems.com/21-cfr-11-10e-audit-trails/>, 1.1.2017
- 27 U.S. Department of Health & Human Services, Minimum Necessary Requirement. <http://www.hhs.gov/hipaa/for-professionals/privacy/guidance/minimum-necessary-requirement/index.html>, 1.1.2017
- 28 The Office of the National Coordinator for Health Information Technology, Adoption of Electronic Health Record Systems among U.S. Non-Federal Acute Care Hospitals: 2008-2014. <https://www.healthit.gov/sites/default/files/data-brief/2014HospitalAdoptionDataBrief.pdf>, 1.1.2017
- 29 National Institute of Standards and Technology, Digital Authentication Guideline, 2016. <https://pages.nist.gov/800-63-3/sp800-63b.html>, 1.1.2017
- 30 Security Innovation Europe, Alan Pearson, What's the Difference Between Hashing and Encrypting? 18.12.2014. <http://www.securityinnovationeurope.com/blog/whats-the-difference-between-hashing-and-encrypting>, 1.1.2017
- 31 Learn Cryptography, Hash Collision Attack. <https://learncryptography.com/hash-functions/hash-collision-attack>, 1.1.2017
- 32 Neustar Research, Choosing a Good Hash Function. <https://research.neustar.biz/2012/02/02/choosing-a-good-hash-function-part-3/>, 1.1.2017
- 33 Michigan Technical University, Introduction to Data Structures, Hash Table Collision Handling. http://www.csl.mtu.edu/cs2321/www/newLectures/17_Hash_Tables_Collisions.html, 1.1.2017
- 34 National Institute of Standards and Technology, NIST's Policy on Hash Functions, 5.8.2015. <http://csrc.nist.gov/groups/ST/hash/policy.html>, 1.1.2017