Linköping University | Department of Computer science
Master thesis, 30 ECTS | Computer Science and Engineering
2016 | LIU-IDA/LITH-EX-A--16/028--SE

Design and implementation of a collaborative secure storage solution

Fredrik Kangas
Sebastian Wihlborg

Supervisor : Ulf Kargén
Examiner : Nahid Shahmehri

External supervisor : Fredrik Sjöstedt

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

In the modern enterprises it is common that support and maintenance of IT environments are outsourced to third parties. In this setting, unencrypted confidential data may pose a problem since administrators maintaining the outsourced system can access confidential information if stored unencrypted. This thesis work, performed at ELITS, presents a solution to this problem; a design of a collaborative storage system where all files at rest (i.e. stored on disk) and in transit remain encrypted is proposed.

The design uses a hybrid encryption scheme to protect the encryption keys used. The keys can safely be stored in a centralized database as well as sent to the clients without risk of unauthorized parties gaining access to the stored data. The design was also implemented as a proof of concept in order to establish that it was possible to realize.

Keywords: *Hybrid encryption system, Secure storage, Collaborative encrypted storage*

Acknowledgments

We would like to thank our supervisors, Ulf Kargén at LiU and Fredrik Sjöstedt at ELITS, for all the support, excellent feedback and rewarding discussions. We would also like to thank our examiner, Professor Nahid Shahmehri, for giving us the opportunity to conduct this thesis. Finally we would like to thank Youtube user Knifoo and Bob Ross for the enlightening background noise of the 10 hour edition of Happy Little Clouds.

Contents

List of Figures

1 Introduction

1.1 Motivation

Many computer systems and applications use large amounts of data, either user provided data or static content in the system. These systems require that the data can be saved and later accessed when needed. This is usually handled by a database or in larger applications, a distributed database consisting of several interconnected databases. It is not unusual that the data is stored unencrypted within the database and that the server encrypts the data before it is being transmitted to the clients. By doing so, the data is only protected against attacks that intercept the connection between the client and server with the intention to steal sensitive data.

Most companies view their data as assets and it is something they want well protected. Thefts of confidential data is something that costs companies a large amount of money yearly [19]. Keeping the data unencrypted in the database opens up for a range of different security breaches that might lead to confidential company data being stolen. If a malicious user gets access to the system it would be no problem to read the data stored in the company's databases if it is stored unencrypted. This could happen mainly in two ways. The first is a so called *inside attack* where an employee of the company, such as an system administrator, uses its access to the system to do harm. These kind of attacks are one of the hardest to mitigate and answer to 40% of the attacks performed against companies [9]. The second scenario is attacks from outside the company. An attacker then uses some security flaw in order to gain access to the targeted company's systems and its database.

It is common that companies outsource the hosting and maintenance of their IT environments. In these situations, it is common that the company responsible for the environment has full access to the system, including any information and files stored within. Considering that most companies view their data as assets, this can be an unwanted drawback of outsourcing. Keeping data unencrypted in a database is therefore a serious security problem.

1.2 Problem statement

Storing confidential data unencrypted can pose a problem. The stored unencrypted data can be accessed by the administrators managing the system or attackers bypassing the system's security. This can greatly compromises the confidentiality of the stored data. The following questions will be addressed in order to propose a solution to the problem:

Question 1. *How can a system for file storage be designed to achieve full encryption from storage to client?*

The main assignment of this thesis is to propose and design a solution for secure file storage in a distributed system. The purpose of the system is to keep files encrypted at all times except at the client which will perform the actual encryption and decryption.

Question 2. *How can such a system be designed to securely manage cryptographic keys?*

In order for the system to work in a collaborative fashion, keys have to be shared between multiple parties. In order to enforce security the keys have to be managed in a secure fashion.

Question 3. *How can such a system be designed to prevent administrators from accessing confidential information?*

Administrators maintaining the database and the file storage should not be able to gain access to confidential information. Even if an administrator has full access to the database, it should not be possible to modify the data and thereby gain access to confidential information.

1.3 ELITS

This thesis will be performed at ELITS, which is an IT company with a focus on management of IT environments [4]. ELITS operates in three main areas: managed services, consulting services and IT services. This thesis will be performed within the third area, IT services, where ELITS provide IT solutions for companies with focus on availability and security. One of the IT solutions that ELITS provide is a storage solution where ELITS manages the data storage of companies' businesses. This thesis will aim to expand ELITS current storage solution to offer a secure storage module.

1.4 Aim

The main purpose of this thesis is to create a theoretical foundation for secure file storage in a distributed system. A system design of such a system will also be proposed. The system design aims to enable the system to keep all data encrypted within the server and its database and let the clients handle the encryption and decryption of data. The system design will be implemented as a proof of concept with limited functionality in order to validate the proposed system design.

1.5 Method

This thesis work consists of four parts: a literature study, requirement analysis, a combined design and threat mitigation phase and finally an implementation phase. The system design phase was carried out as an iterative process. This can be seen in figure 1.1

The first part, the literature study, will be used to gain knowledge primarily about encryption and key generation. This includes different types of encryption, different algorithms to perform encryption, as well as key generation and strengthening. Encryption will be a central part of the system and will be prioritised in the literature study. In order to take advantage of the encryption algorithms proper keys have to be used, which is why key generation and strengthening will have to be researched. The literature study will also contain topics related to secure network traffic, file storage and existing file encryption software. Publications in the field of computer science as well as Internet resources will be used to gather information about theses topics.

Then we will proceed to the next part, which is the requirement analysis. During this part of the thesis work we will specify and analyse the requirements of the system. The requirements will be derived by analysing the main features of the system in cooperation with our supervisor at ELITS. The requirements will then be analysed to break them down in order to find out what they entail for the design and implementation of the system.

During the system design phase, the architecture of the system will be proposed. This phase will be an iterative process. The architecture shows how all the parts of the system will be related and structured. During this phase the database used by the system will also be modeled. An *Enhanced Entity Relationship Diagram*, EER diagram, will be used in order to model the relations between entities of the system. Since one of the system's main functions is to store files, a model for the file storage will also be proposed in the design phase. This includes how files are referenced in the database as well as how they will be physically stored. Also, the communication flow of the system's main features will be described in the proposed design. The flows will show the data sent between the different parts of the system, as well as what is stored in the database, in order for the system to carry out a certain task. Along with the system design a threat analysis will be done. For each iteration of the system design phase the design will change according to the findings from the treat analysis. The iterations will stop once no more realistic threats are identified. At this stage an attack tree [21] will be created in order to analyse the overall security of the proposed system. The iterative process described above has been inspired the by the essence of many agile software development methodologies such as SCRUM [31] and OpenUP [16]. What all agile software methodologies have in common is the goal of iterating and reworking a solution until it is finished, rather than developing the solution in a linear fashion [17]. The flexibility of the agile methodologies is what inspired the iterative process presented above.

The final phase of the thesis is an implementation phase. During this phase a proof of concept of the system will be developed in the *Go* programing language. The product of this phase will serve to validate that the correct system was designed.

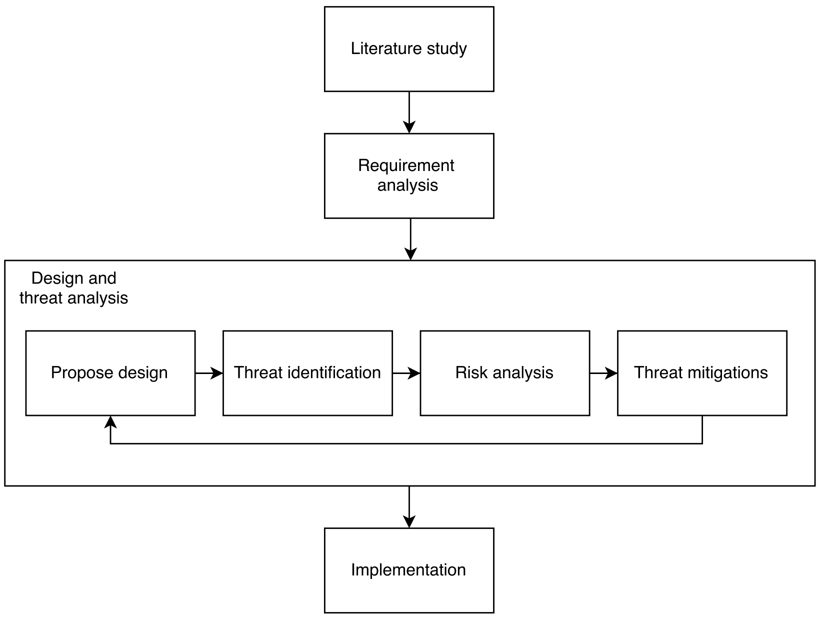


Figure 1.1: Method phases

1.6 Delimitations

The clients in the system will at some point keep the user's encryption key in the memory. We will not go into details on how to protect against potential extraction of keys kept in memory. It is assumed that the memory used by the client is protected by the environment that runs the client. Furthermore, attacks that rely on extraction of keys will not be considered.

The system flows that will be presented as part of the design chapter will only be concerned with the basic functionality of the system. The flows described will have certain security aspects to consider, which is why they will be presented in detail. Not all functionality have similar security aspects and will therefore not be presented in this way.

The platform for the proof of concept clients will be OS X which is the platform provided by ELITS. Other parts of the system has no such limitations. The proof of concept will only contain basic functionality based on the requirements specified by ELITS. To handle the local encryption of downloaded files, VeraCrypt [29] will be used as a part of the client. In addition, the proof of concept will not be a distributed system.

2 Theory

The following sections will give the needed background knowledge and the relevant concepts in order to fully grasp the thesis work. For the following sections the book *Cryptography and network security: principles and practices* [25] has been used as a reference in addition to the ones provided.

2.1 Attacks

In information security an attack is an attempt to violate confidentiality, integrity or availability of data. This can be accomplished by stealing or destroying data as well as denying access to the service which provides the data. The following sections will explain a few attacks, all of which are relevant to this thesis.

Reverse engineering

Reverse engineering is the act of extracting knowledge or design information from software. Based on this information the software could be reproduced as is or with alterations to the original software. While reverse engineering could be used with good intentions, for instance to reproduce legacy software without documentation, it can also be used with malicious intent. Applications which use cryptographic keys to encrypt or decrypt sensitive data could be subject for reverse engineering attacks. By reverse engineering the software an attacker could get hold of these keys and use them to decrypt sensitive data from a local repository for instance. Another scenario where a reverser engineering attack could be used maliciously is to extract information about certain algorithms such as encryption and key generation algorithms. Knowledge about the algorithms used will enable an attacker to replicate the software, but with intentional flaws that compromise the security of the software.

Eavesdropping

To get hold of secret information, eavesdropping can be used. An eavesdropping attack is when an unauthorized party secretly picks up information shared between two parties, without their consent. In the scenario depicted in figure 2.1, the attacker can listen to the communication without the sender and the recipient knowing. Eavesdropping can be carried out in many different ways. The most basic way is secretly listening in on a conversation between two persons, but it can also be done in more advanced ways. This involves, tapping in on a phone line to listen to people's calls or secretly listen to traffic on a computer network. The act of eavesdropping is completely passive and the eavesdropper just listen to the communication, not altering or manipulating the information. To protect against eavesdropping, encryption can be used. By encrypting the information sent over the channel, only authorized parties can read it. This makes the information far less useful for a potential eavesdropper.

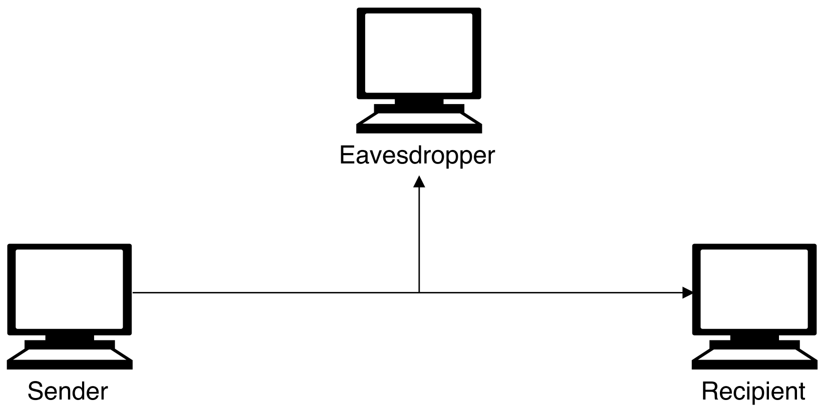


Figure 2.1: Eavesdropping attack

Man in the Middle

Man in the middle, MITM, is an attack that is widely known in communication security. In a MITM attack, the attacker intercepts the communication between two parties without their knowledge. This might seem like the same as an eavesdropping attack, but in a MITM the attacker have full control over the data sent between the two parties. The attack can be seen as an active eavesdropping attack. A MITM attack can be used in different ways. It can be used to eavesdrop by passing on the information unchanged. There is also a possibility for the attacker to alter the information passing through. In the scenario in figure 2.2, the sender and the recipient think they are communicating with each other. In fact, they are both communicating with the attacker, who is impersonating the sender or the recipient. By tricking the users to think the attacker is a trusted party, the attacker can ask for secrets that should only be revealed to trusted parties. The secrets revealed might be a password or an encryption key. To protect against MITM attacks, it is of importance that the sender and the recipient can verify the identity of the other party. This can be done by using digital signatures with protocols such as SSL/TLS.

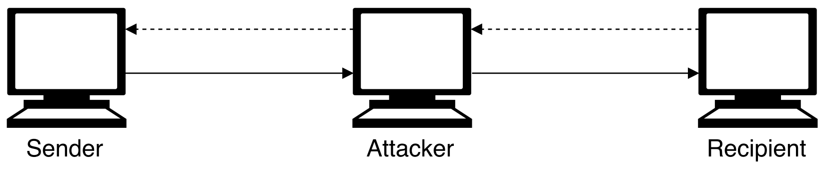


Figure 2.2: Man in the middle attack.

2.2 Encryption

Encryption is the process of protecting data from being read by unauthorized users. This is mainly done by mathematically altering data so that only authorized users can access it.

The unencrypted data, usually know as *plaintext*, is transformed by an encryption algorithm. Transformed data is often called *ciphertext* and needs to be decrypted before it can be accessed. The encryption algorithm uses an specific *encryption key* to encrypt the data and only an authorized user with the correct *decryption key* can access the data. There are two main types of encryption, symmetric and asymmetric encryption.

Symmetric encryption

Symmetric encryption is a type of encryption where the same cryptographic key is used for both encryption and decryption. The authorized parties agree on a symmetric key which is usually generated by a so called *pseudo-random key generator*. The selected key then needs to be kept secret in order for the encryption to remain secure. The parties can now securely exchange information. The sender encrypts its messages with the shared key and receiving parties decrypt the message with their key. The main drawback of this type of encryption is that all parties need to have access to the key, which requires some kind of key exchange. This can e.g. be handled by sending the symmetric key over a secure communication channel, typically achieved by asymmetric encryption. However, the performance of encryption and decryption with symmetric encryption is much better than that of asymmetric encryption and symmetric encryption is therefore widely used.

There are two categories of symmetric encryption algorithms, block ciphers and stream ciphers. In block ciphers a predefined number of bits, a block, are passed to the algorithm and encrypted at the same time. Block ciphers is widely used in algorithms used for encrypting large data files. They often operate in a number of rounds, where the block is transformed by simple permutations and substitutions. In stream ciphers on the other hand, each bit is passed to the algorithm and encrypted individually. Stream ciphers is often used in scenarios where the amount of data to be encrypted is unknown, like a wireless network or a phone call. Stream ciphers use a random bit stream as the encryption key. Each bit of the plain text is encrypted, typically by XOR:ing, with the corresponding bit of the encryption key stream. The key stream used in a stream cipher needs to be fully random and is only to be used once.

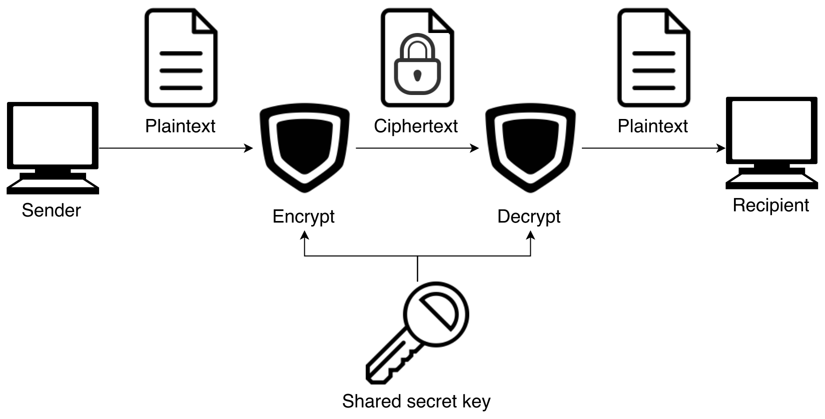


Figure 2.3: Symmetric Encryption

Advanced Encryption Standard

In the late 90s there was a need to replace the current encryption standard at the time, *Data Encryption Standard*. This was done by publicly holding a competition to find a new encryption algorithm that was more efficient and secure than the current standard. The winning algorithm was *Rijndael*, a symmetric block cipher. The new standard was called Advanced

Encryption Standard, AES, and was the first openly published encryption algorithm to be approved by National Security Agency of the United States in 2002.

AES uses a block size of 128 bits and is able to use an encryption key length of either 128, 192 or 256 bits [26]. The algorithm operates on a block by running a series of operations on it. For how many rounds the series of operations is performed is determined by the key length. For keys of 128 bits, 10 rounds is used, and keys of lengths of 192 and 256 bits uses 12 and 14 rounds, respectively.

When a block is encrypted with AES-128, the encryption key is used to generate 10 unique round keys that will be used in each round. The block is then organized as 4x4 matrix, where each position contain 1 byte of the block. The matrix is then transformed by having each byte XOR:ed with the first round key. In each round the matrix is then transformed by substitution and permutation of the bytes. The first step of a round is substitution for each of the bytes according to a substitution table. The rows of the matrix are shifted cyclically to the left in the next step. The first row is not shifted at all, while the other rows are shifted 1,2 and 3 bytes respectively. After the rows of the matrix have been shifted, the columns are mixed. These two steps provides *diffusion* in the cipher. This means that changing one byte in the input causes several bytes of the output to change. Diffusion causes potential patterns to scramble and greatly increases the amount of data needed to break the cipher by analysis. In the last step of a round the matrix is XOR:ed with the round key for the next round. These four operations are then repeated for the remaining number of rounds.

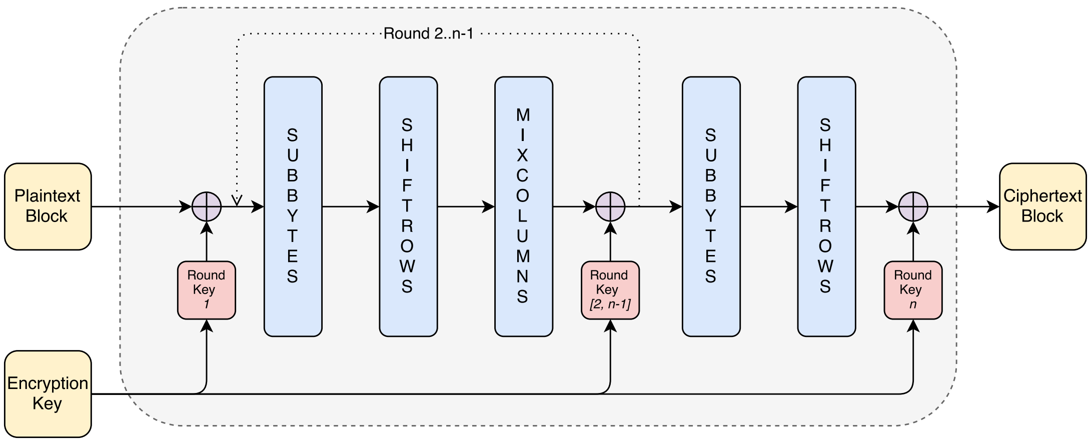


Figure 2.4: Block diagram of AES

AES is widely used because of its performance and security. Since the operations performed in the rounds are simple and can be done in parallel by hardware, manufacturers of processors have included hardware acceleration for AES in their processors. This means that the processors is shipped with instructions to perform encryption with AES, which greatly increase the speed of the encryption and decryption.

Other algorithms

There exists other symmetric block cipher algorithms. The most common is the other finalists from the public competition, *Serpent* and *Twofish* [14]. For the implementation of this system AES will be used since it is the current standard. Therefore, the other common symmetric block cipher algorithms will only be mentioned briefly.

Serpent uses the same block size and key length as Rijndael, but uses 32 rounds [10]. The algorithm uses permutations and substitutions and is designed for all operations to run in parallel. By using 32 rounds, it actually provides a higher security margin than Rijndael, but Rijndael is faster and easier to implement. This was the main reasons which made Rijndael the new standard algorithm.

Twofish also uses the same block size and key length as Rijndael and uses 16 rounds [22]. Twofish is based on a *Fiestel network*, which is a structure that transforms any function into a permutation. The function in the Fiestel network is often called the *F function*. The F function is a key-dependent mapping of an input into an output, which is always non-linear. Twofish was slightly slower than Rijndael when implemented on most platforms and was not selected as the new standard.

Asymmetric encryption

Asymmetric encryption is a type of encryption where key pairs are used instead of a single key. The keys in the key value pair consist of a *public key* and a *private key*. The *public key* is used for encryption while the *private key* is used for decryption. One central aspect of asymmetric encryption is that data encrypted with the *public key* cannot be decrypted using that same key, only the *private key* can be used for this purpose.

Asymmetric encryption is based upon mathematical problems for which there exists no current efficient solutions, for instance integer factorization, discrete logarithms and elliptic curves. These problems are easy to use in order to generate key pairs while it is difficult, close to impossible, to calculate the *private key* from the *public key*. Therefore you can safely keep the *public key* published without compromising security. However the private key **still** has to be kept secure.

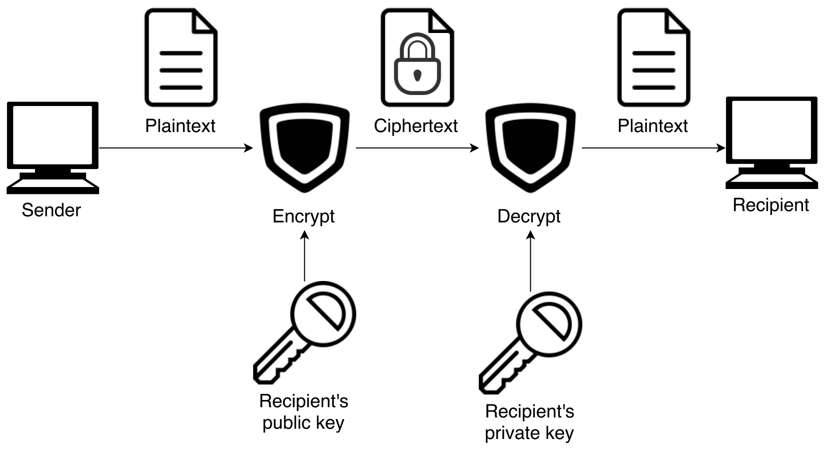


Figure 2.5: Asymmetric encryption

RSA

RSA is an asymmetric encryption algorithm first presented in 1977 by Rivest, Shamir and Adleman [20]. The security in RSA relies on the difficulty of factoring the product of two large prime numbers. Plain RSA is not semantically secure. This means that it is possible for an attacker to separate two encryptions form each other if the attacker knows the corresponding plaintext. In order to make RSA semantically secure a padding scheme needs to be added. This will be further explained in it's own section.

A RSA cryptosystem consists of three steps, key generation, encryption and decryption. If Bob wants to receive messages encrypted with RSA he will generate the keys in the following way.

1. Bob selects two different prime numbers p and q.
2. Bob will compute $n = pq$ which will used as the modulus for the keys.
3. Bob calculates the value of *Euler's totient function* for n: $\phi(n) = \phi(p)\phi(q) = (p-1)(q-1) = n - (p+q-1)$. This is a private value.
4. Bob chooses a integer e which fulfills $1 < e < \phi(n)$ as well as $gcd(e, \phi(n)) = 1$.
5. Bob then calculates d so that $de = 1 \; mod \; \phi(n)$

The public key is then the values $[n, e]$ while the private key is the values $[n, d]$. However the values p, q and $\phi(n)$ also needs to be kept secret since they can be used to calculate d. If Alice wants to send an encrypted message to Bob she will do the following:

1. Alice obtains Bobs public key consisting of $[n, e]$.
2. Alice generates the ciphertext c by calculating $c = m^e mod \; n$ where m is the message.
3. Alice sends c to Bob.

When Bob wants to decrypt the ciphertext c sent by Alice he will use his private key $[n, d]$.

1. Bob calculates $c^d = (m^e)^d = m \; mod \; n$.
2. Bob can read the plaintext message m.

The procedure described above is plain RSA. As mentioned earlier plain RSA is not semantically secure and a padding scheme needs to be added to achieve semantically security. The padding scheme would be applied to the message before encrypting and after decryption.

Optimal asymmetric encryption padding A padding scheme is commonly used to expand plaintext to a specified length. This is the case for symmetric encryption algorithms which require plaintext to be a multiple of the block size. However, in asymmetric encryption algorithms the purpose is different. The padding scheme used with asymmetric encryption algorithms aims to add structured, randomized padding to the message before encryption. This means that a message, once padded, will encrypt to one of a large number of different ciphertexts. This entails that attackers can't distinguish between encryptions even if they have knowledge about the corresponding plaintexts.

Optimal asymmetric encryption padding (OAEP) is a padding scheme that is commonly used together with RSA [5]. OAEP can also be used to build an all-or-nothing transform which means that you need to have the entire message in order to reverse the padding. When used together with RSA, OAEP consists of a number of components:

- The RSA modulus n.
- Two integers k_0 and k_1.
- The plaintext message m consisting of $n - k_0 - k_1$ bits.
- Two cryptographic hash functions G and H.

The procedure to encode a message m is as follows.

1. Pad m with k_1 zeroes to create $m0..0$.
2. Generate random k_0 bits string r.
3. Apply G to r to generate a $n - k_0$ bits string.

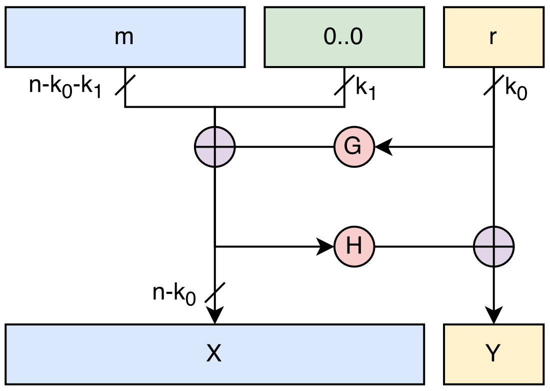


Figure 2.6: Optimal asymmetric encryption padding diagram

4. Calculate $X = m0..0 \oplus G(r)$.
5. Apply H to X to generate a k_0 bits string.
6. Calculate $Y = r \oplus H(X)$
7. The encoded message now consists of $[X, Y]$.

To decode the encoded message, $[X, Y]$ together with H and G is used as follows:

1. Calculate $r = Y \oplus H(X)$.
2. Calculate $m0..0 = X \oplus G(r)$.

The decoded message will still contain the zeroes added in step one of the encoding process and will have to be removed before using the message.

If OAEP used together with RSA is commonly refereed to as RSA-OAEP. This variation of RSA is semantically secure as opposed to plain RSA.

ElGamal

ElGamal encryption is an asymmetric key encryption algorithm, based on the older Diffie-Hellman key exchange, first described by Taher Elgamal in 1985. The security in ElGamal is based upon the difficulty of solving certain problems involving discrete logarithms [3].

ElGamal encryption is composed of three components: the key generator, the encryption algorithm and the decryption algorithm. If Bob is the one who wants to be able to receive encrypted messages he will generate the key pair in the following way:

1. Bob selects a large prime p.
2. Bob selects a primitive root $\alpha \ mod \ p$
3. Bob calculates $\beta = \alpha^x \ mod \ p$ where x is a random integer.

The public key is the values $[p, \alpha, \beta]$ while x is the private key. If Alice wants to send an encryption message to Bob the following procedure will be used:

1. Alice acquires Bobs public key consisting of $[p, \alpha, \beta]$.
2. Alice converts the message m into an integer representation M.
3. Alice generates a random integer k.
4. Alice then generates $a = \alpha^k \ mod \ p$ and $b = \beta^k M \ mod \ p$.
5. Alice then sends Bob the ciphertext containing $[a, b]$.

The different components of the ciphertext have different purposes. a is used to transmit Alice's secret k and b is used to transmit the actual message m. Furthermore k is supposed to be used only once and not be the same in succeeding encryptions. This essentially means that the cipher text will not be the same for consecutive encryptions of the same plaintext.

When Bob wants to decrypt the message from Alice he will be using his secret x and the ciphertext $[a, b]$. In order to decrypt the message:

1. Bob will calculate $M = \frac{b}{a^x}\ mod\ p = ([b\ mod\ p][a^{-1}\ mod\ p]^x)\ mod\ p$
2. Bob will then transform the integer M into the correct encoding of the original message m.

The encoding of m needs to be known on beforehand in order to transform the integer representation M. The transformation step can however be omitted if the message that is supposed to be encrypted already is an integer.

Furthermore the ElGamal encryption is considered to be semantically secure [23]. This means it is secure against a passive eavesdropping adversary. The fact that it is semantically secure makes it infeasible to derive meaningful information about the plaintext of a message from the ciphertext and the public encryption key.

Elliptic curve cryptography

An elliptic curve is the graph of the equation $y^2 = x^3 + bx + c$ where a and b are real numbers [27]. In cryptographic uses, elliptic curves modulo a prime, are most suitable. This is due to the fact that those curves have a finite set of points. Elliptic curves are useful in cryptographic applications because it is possible to add any two points on the curve to produce a third point on the curve. This property is used to apply elliptic curves to existing cryptosystems. In general one of two procedures is used to do this:

1. Change modular multiplication to addition of points on an elliptic curve.
2. Change modular exponentiation to multiply a point on an elliptic curve by an integer.

The second procedure is only a special case of the first. Exponentiation is equal to multiplying a number by itself multiple times while multiplying a point by an integer is to add the point to itself multiple times.

Elliptic curve versions exist for multiple cryptosystems (ECC), for example elliptic curve ElGamal. For Bob to create an asymmetric key pair he will need an elliptic curve E which needs to be know by all parties in the system. He then chooses a point α on E and a secret integer a and computes $\beta = a\alpha = \alpha + \alpha + .. + \alpha$. The public key consists of $[\alpha, \beta]$ and the private key is a. For Alice to send a message she will express her message as a point M on E. She chooses a random integer k and computes $y_1 = k\alpha$ and $y_2 = M + k\beta$ and sends the pair $[y_1, y_2]$ to Bob. Bob decrypts by calculating $M = y_2 - ay_1$.

Generation of elliptic curves is time consuming since it requires the computation of all points on a curve. Therefore there exists a number of standard curves that can be used. The use of standard curves does not influence the security of elliptic curve cryptography. The security of ECC relies on the fact that performing point multiplication is possible while it is infeasible to calculate the multiplicand from the original and product points. This is unaffected by knowing the elliptic curve used.

Digital Signature

A digital signature is a scheme for proving authenticity of a digital message or file. A digital signature allows a recipient of a message to validate that the message originated from the sender while also validating the message integrity.

Most asymmetric encryption schemes can be used to create digital signatures by using public and private keys. In order to create a digital signature, the signer will encrypt the message with his private key which results in the digital signature. To validate the digital signature, the verifier needs to have the digital signature, the plaintext message and the signer's public key. The verifier will decrypt the digital signature and verify that the decryption and plaintext coincide, in which case the validation was successful. The digital signature is therefore tied to both the signer and the message. This entails that the digital signature can not be copied and used together with another message by an impostor.

Hybrid cryptography

Symmetric and asymmetric encryption algorithms have different advantages and disadvantages. Symmetric algorithms are in general significantly faster than asymmetric algorithms, however they require all parties to share a key that needs to be kept secret. Asymmetric algorithms on the other hand, allow a public key which can be safely distributed at the cost of performance [12]. A hybrid cryptosystem uses the advantages of each type of encryption and reduces the impact of the disadvantages. The most common approach to hybrid cryptosystem is to first generate a symmetric key that will be used to encrypt a message with a symmetric encryption algorithm. The secret key is then encrypted with an asymmetric encryption algorithm and the public key of the recipient. Both the encrypted secret key and the encrypted message is then sent to the recipient [8]. For example, to send a message to Bob using such a system Alice does the following:

1. Alice obtains Bobs public key.
2. Alice generates a new symmetric key.
3. Alice encrypts the message using the symmetric key.
4. Alice encrypts the symmetric key with Bob's public key.
5. Alice sends both encryptions to Bob.

To decrypt the ciphertext Bob does the following:

1. Bob uses his private key to decrypt the symmetric key.
2. Bob uses the symmetric key to decrypt the actual message.

Since the message can potentially be large, the more efficient symmetric encryption algorithm is used to perform the bulk of the work in encrypting and decrypting the message. The inefficient asymmetric encryption algorithm is only used to distribute the secret key used for the symmetric encryption. Hence the hybrid cryptosystem uses the two types to the best of their advantages.

2.3 Hash functions

A hash function is a function that can be used to transform data of arbitrary size into data of a fixed size. The value returned from a hash function is usually referred to as a *hash*. Cryptographic hash functions can be defined by three resistance properties the hash functions need to comply with.

- *Pre-image resistance:* It should be hard to find a message that results in a given hash.
- *Second pre-image resistance:* Given one message, it should be hard to find another different message that result in the same hash.
- *Collision resistance:* It should be hard to find two different messages that results in the same hash.

To describe the resistance properties the term hard is used. Hard, in this context, means that it is almost certainly impossible for an adversary to circumvent the resistance properties for as long as the security of the system is deemed important.

It is common to also provide a *salt* to the data when using a hash function. A salt is random data that is appended to the original data before applying the hash function. The salt makes it harder for an attacker to use pre-computed tables for e.g. various possible passwords and their hashes, since the attacker also have to take the salt into account. It also prevents equal data to have equal hashes since the added salt will make the data different before applying the hash function.

2.4 Key Management

Key management is the management of keys used in cryptographic applications. The concept of key management includes how to generate, exchange and store keys securely. Good key management is an important part of the security in a cryptographic application. This is due to keys being used to enforce security throughout the cryptographic application, hence they need to be managed properly.

Key generation using /dev/random

For key generation, a *random number generator (RNG)* is commonly used. It can be either a computational or physical device with the sole purpose of generating sequences of numbers without any pattern, i.e. random numbers. */dev/random* and */dev/urandom* are two files in Unix based operating systems which serves as an interface to the kernel's RNG [18]. The RNG collects environmental noise from device drivers as well as other sources and stores it in an entropy pool. The RNG also keeps track of how many bits of noise there is in the entropy pool. From the entropy pool the random numbers are then created when requested.

The difference between */dev/random* and */dev/urandom* is that the previous is a blocking RNG. That means that when the entropy pool is empty, reads from */dev/random* will block until more environmental noise has been collected. */dev/urandom* on the other hand will not block if the entropy pool is empty. Instead it will use a *pseudorandom number generator (PRNG)* to create the requested bytes. This entails that */dev/random* should be suitable for most applications that requires high quality randomness, such as key generation [7]. Since */dev/urandom* uses a PRNG if the entropy pool is empty the values returned will not have as high quality randomness. The reduced quality of randomness given by */dev/urandom* could potentially be vulnerable to cryptographic attacks on the algorithms used by the PRNG. */dev/random* will return at most 512 bytes while */dev/urandom* will return at most 32 MB.

Despite the drawback, */dev/urandom* is preferred when accessing the entropy pool. This is due to the fact that */dev/random* blocks which can cause a program to be blocked for a long time while new entropy is collected. */dev/random* is only recommended for keys which need a long life time, such as keys for SSL. More about SSL in section 2.5. However OpenSSL uses */dev/urandom* as default [15].

Key derivation function

A *key derivation function (KDF)* is used to derive secret keys from a secret value, such as a password. KDF can be used to transform passwords into keys of suitable length. This is called stretching.

One KDF is *PBKDF2* which stands for password-based key derivation function 2 [28]. It applies a hash function to the input together with a salt and repeats this process multiple times in order to produce the *derived key*. This key can then be used as an encryption key or other cryptographic applications.

2.5 Secure Communication (SSL/TSL)

Secure Socket Layer / Transport Layer Security (SSL/TLS) are cryptographic protocols designed to provide a security on top of a reliable transport layer [2]. The transport layer is usually the Transmission Control Protocol (TCP) which provides reliable, ordered and error-checked delivery of data streams. Therefore, SSL/TLS is only concerned with privacy and data integrity between two communicating parties. SSL/TLS uses symmetric and asymmetric encryption as well as certificate authorities (CA) to provide this security.

During the initial setup of a SSL/TLS session the client and server negotiates which version of SSL/TLS, cipher suite and algorithms to use during the session. Next, the server sends a certificate to the client. This certificate has been signed by a CA and is used by the client to verify the identity of the server. After the client verifies the certificate, the client and server have to agree on a cryptographic key. This key is for the symmetric encryption used to encrypt the data transmitted. The client generates *nonce*, a random number, which will be used to generate the key for the symmetric encryption. The client encrypts the nonce with the server's public key, which is part of the certificate. The client sends the encrypted nonce to the server which can decrypt it. Now both the client and the server have the nonce and can generate the cryptographic key for the symmetric encryption. This key is then used for the lifetime of the session.

2.6 Object Storage

Object storage is an alternative storage architecture to file and block storage. In a file storage, files are structured in a file system. The files are kept in a folder hierarchy and metadata is stored in the file system. Block storage on the other hand, divides files into blocks and these blocks can be stored individually. When a file is requested, the individually blocks are located by their address and combined to assemble the file again. The block addresses are kept within the block storage system and is the only metadata stored about the files.

In an object storage, an object is defined as a file together with all related metadata [13]. Unlike files in a file system, objects are kept in a flat structure called the *storage pool*. Objects can only be kept in the storage pool and an object cannot contain another object. Both files in a file system and objects in an object storage have metadata associated with the data they contain. Object storages, however, does not have a limit on the amount of metadata that can be associated with the data. The developer can freely store metadata they need for their application. When an object is created it is assigned a unique identifier, usually generated by the content of the object. To retrieve an object, the only information needed is the identifier. This allows a server or end-user to retrieve an object without knowing the physical location of the data, unlike a file system where a path is needed.

The fact that the object storage keeps all files in the storage pool allows for great scalability. The storage pool eliminates the overhead of keeping track of large amounts of directory metadata which is a typical bottle neck in file systems. The flat structure of the object storage allow continued horizontal scalability, practically without a limit on data quantity. This is accomplished by simply adding new servers to the object storage rather than improving existing hardware.

3 Requirements analysis

In this section all the requirements for the system in this thesis will be listed and described. The list of requirements was acquired by discussions and meetings with our supervisor at ELITS. The requirements will be divided into functional and non-functional requirements. The functional requirements describe the behavior of the system while the non-functional requirements are concerned with attributes such as security and scalability.

3.1 Functional requirements

Requirement 1: *All encryption and decryption shall be performed at the clients.*

A central aspect of the system is that all files in the storage shall be encrypted. This entails that all data going in to the system will be encrypted at the client before being sent to the storage. In the same fashion, data retrieved from the storage will be transmitted encrypted and will be decrypted by the client. That means that no encryption and decryption of data will be, and shall not be, performed at any other part of the system than the clients.

Requirement 2: *The user shall be able to use the system collaboratively.*

The files stored within the system shall be accessible by a group of users. A user shall be able to create rooms where encrypted files can be stored. Only users authorized shall be able to see and access the files in the room. The owner of the room shall be able to invite other users to access the room. The system shall also be able to handle if a owner requests to remove a room or remove a user's access to the room. The server needs to keep track of which encrypted files are accessible by which users.

Requirement 3: *A version control system for files shall be present.*

Since the system shall allow collaboration of files it is necessary to provide version control of the files. In cases where multiple users modifies the same file it shall be possible for the system to detect conflicts. For instance, a conflict could be detected if two users modifies the same area of a file. When a conflict is detected the system shall try to solve the conflict by itself or alert the last user who made modifications to the file. That user will then have to solve the conflict manually. Furthermore it shall be possible to see the alteration history of the files and if desirable rollback to a previous version of the file.

Requirement 4: *Files that have been received by the client shall be stored encrypted at the client.*

The data requested by the client will be received encrypted and will be decrypted by the client. The client shall not store the data in plaintext but will encrypt the data with its own local encryption. The reason is that the data shall be protected at the client as well as in the storage. The benefits of using a local encryption for this purpose is that no encryption key used for the files in the storage will have to be kept at the client. Furthermore, this allows a user to keep working while not connected to the system and still keeping the data protected.

Requirement 5: *The system shall be able to run in two different modes.*

When the system is setup, the user shall be able to choose which of the following modes for the system:

Mode 1: *Private Mode*
When the system is set to run as private mode all the files encrypted can only be accessed by authorized users. If a user does not want to share the encrypted information there is no way to access it. There is no way of accessing a user's files without having the encryption key. This means that there is no recovery from lost keys.

Mode 2: *Enterprise Mode*
This mode is, as the name implies, mainly for use in enterprises. A system in this mode always adds access for a chosen user to files created and encrypted with the system. This means that e.g. an enterprise that uses the system always have access to all the information. This prevents from scenarios where data is locked down because of lost passwords, black mailing etc.

Requirement 6: *It shall be possible to change the encryption keys used in the system.*

The system shall be able to change the encryption keys used for already encrypted information. There shall be a way to force a change of the cryptographic keys. This can be used if the users feels that the security of a key have been compromised. This feature could also be used to change the keys continuously as a precaution for compromised key security.

Requirement 7: *To gain access to the system, a secure registration process shall be performed.*

Before a user gains access to the system, there shall be a registration process. An access administrator of the system shall register an account for the user, which have to activate the account. During the activating process it shall be possible to verify that the user activating the account is in fact the user who requested the account from the access administrator.

Requirement 8: *The system shall be able to handle different user privilege levels.*

The system shall include three different privilege levels: owner, write and read. These privilege levels shall not be determined by a simple entry in the database, for example an integer representing the level, as that would makes it possible for a system administrator to manually change the privilege level of a user. Instead the system shall be designed in a way that makes it infeasible to manually change the privilege level of a user.

- *Owner:* Shall be able to perform all actions available in the room.
- *Write:* Shall be able to read and modify existing files as well as uploading new files.
- *Read:* Shall only be able to read files.

Furthermore the system shall be designed to include the possibility to add new privilege levels.

3.2 Non-functional requirements

Requirement 1: *No encryption keys shall be stored on the clients.*

Since the clients alone handle the encryption and decryption, they will also handle the encryption keys. It is of importance that these keys are not stored directly on the clients, e.g. in a file or hard coded in the clients. By using this approach the security of the encryption may be compromised. If the keys are stored directly on the clients, there is a possibility that an attacker will be able to get hold of them. This can be avoided by generating the encryption key just before usage and keeping the encryption keys in memory no longer than needed. After the key is used, the memory should be correctly emptied before deallocated.

Requirement 2: *System administrators shall not be able to get access to the data if not explicitly allowed to.*

The system shall be designed in a way that makes it infeasible for the administrators of the system to get access to any of the stored data. By having access to the database, there shall be no way to inject data or change records to give unauthorized users the ability to access stored data. Only an authorized user of the right privilege level shall be able to give another user access to its data.

Requirement 3: *The system shall be possible to run as a distributed system.*

The system shall be able to deploy as a distributed system. The system shall be able to handle a large amount of requests together with large files. A solution to that problem is to have the storage solution in a distributed system together with a load balance server. Therefore the system shall be designed in such a way that it is possible to do so.

4 Design

4.1 Introduction

In the following chapter, the proposed design will be presented. The requirements from chapter 3 are used as the foundation for the design overview, which presents the general functionality of the system. Furthermore a system architecture will be described, presenting the individual component of the system. Following the system architecture, the database structure will be presented where individual tables are explained. In order to get a better understanding of how the system operates, essential system flows will be explained in detail. The system flows will show data transactions in the system as well as communication between the components. The chapter will end with a discussion of the proposed design.

The design took shape over a total of four iterations. The changes ranged in size from alternating information flow to completely redesign modules of the system. When performing the iterations a proposed design was analyzed and threats identified, more about this process in chapter 5. Mitigations to the identified threats were then proposed and adapted to the design, ending the iteration. This process resulted in the design that will be presented in this chapter.

4.2 Definitions

This section contains definitions of terms that will be used throughout the design chapter and subsequent chapters.

- *Room:* A room can be considered a group of users. A room will have encrypted files associated with it, which members of the room can access.
- *Room key:* The room key is the cryptographic key used to encrypt and decrypt all the files in a room. In the figures the room key will be denoted by sym_R.
- *Local key:* The local key is the cryptographic key used to encrypt and decrypt the files stored locally when downloaded by the client. In the figures the local key will be denoted by sym_L.
- *User's asymmetric key pair:* The user's asymmetric key pair is generated from the user's password. The public and private parts of the asymmetric key pair will be called the

user's public key and the *user's private key* respectively. In the figures they will be denoted by $asym_{pub}$ and $asym_{priv}$.

- *Multiple asymmetric key pairs:* In some contexts there will be multiple asymmetric key pairs mentioned. To keep them apart, they will be referred to in a similar fashion to the user's asymmetric key pairs, e.g. *x's public key* where x is another participant. In the figures they will be denoted by $asym_{pub\_x}$ and $asym_{priv\_x}$.
- *Encrypted room key:* An encrypted room key is a room key that has been encrypted with a public key. These will be denoted by $asym_{pub}(sym_R)$ in the figures.
- *Local encryption module:* A module that handles the encryption and decryption of the local files when downloaded by the client. It will also mount the files as a file system that can be displayed and interacted with by the client. The module can e.g. be a third party software or developed to fit the user's needs.

4.3 System overview

Based on the functional requirements from chapter 3, a system that operates in the following way has been designed.

A new account is registered in the system by an *access administrator*. An activation code for the account is sent by SMS to the user. The new user can activate the account in the system by using the received activation code. To interact with the system, the user will use a client. When a user is logged in at a client, all the available rooms for that user will be shown. Each user in the system can be a member of several rooms. The user can navigate through the rooms to see all the files contained in the room. The user can then choose to either download files from a room or upload files. In order for a user to use a file, it has to be downloaded and encrypted locally on the client. Furthermore the system contains a version control mechanism. When a user uploads a modified file, the server will check when the file was initially downloaded to the client as well as when the last modification of the file occurred in the object storage. If a modification has been made to the file in the object storage while the client used it locally, a conflict will be reported. The file in the object storage will then be sent to the client which has to resolve the conflict before initiating a new upload.

The client communicates with the server of the system. It is important for the server to verify that a user requesting an action, is in fact the user and not an attacker impersonating the user. Therefore, the user needs to be authenticated by the server before an action can be carried out in the system.

When a new room is created the user who created it will be the owner of the room. It is only the owner who can invite other users to the room for collaboration. When a new user is invited to the room, the owner will choose the privilege level of the user. Invited users will get access to the files in the room and can download them for local use, depending on the privilege level the user may also write files to the room. It is also possible to remove a user from a room. When a user is removed from a room, the files contained in the room can not be accessed by the user. The user removed might have files locally that was downloaded from the room before. If so, the server will request that the client removes the data when the client connects. An owner of a room can of course also delete the room itself. When the room is deleted all the files associated with the room will be removed from the storage.

If a room encryption key were to be considered comprised it is important that the room key can be changed. In order to change the room key all files associated with the room will be replicated and encrypted with a new room key. When the replication is complete the old room will be removed. When a user belonging to the old room requests an arbitrary operation concerning the old room, the server will send a request to the client to verify itself. When this process is complete the server will continue with the original request.

System Modes

The secure file storage system shall be possible to operate in two different modes, private and enterprise. Which mode to use will be determined at the initial configuration of the system. Enterprise mode is a compromise between availability and confidentially while private mode has confidentially as highest priority. It is up to the end-user and to decide which mode they believe is most suitable for their operations.

In enterprise mode there will be a *keywarden*. This is a special user, with maximum privileges, that will automatically be added to the room when it is created and which cannot be removed. The purpose of the keywarden is to avoid potential data loss if a room where to be rendered inaccessible. An inaccessible room could be the result of a room owner leaving the company or a malicious employee removing all other users from the room. In that case no additional users can be added to the room and no data can be accessed. However, in enterprise mode the keywarden can prevent a room from being inaccessible. The keywarden can simply assign a new owner. In private mode, however, there will be no such recovery mechanism. If a room is somehow rendered inaccessible there will be no way to make it accessible.

4.4 System Architecture

The system consists of five different parts: the clients, the server, a database, a replication server and an object storage. An overview of the system and how the parts communicate can be seen in figure 4.1. Each part will be described in more detail in the following sections.

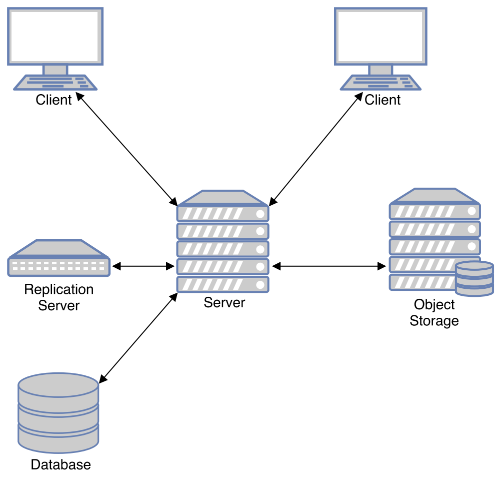


Figure 4.1: Architecture of the system.

Client

The clients are responsible for generating the user's keys as well as encrypting and decrypting the downloaded data. The client will use a local encryption key to encrypt and decrypt the local data. It will also use an asymmetric key pair to encrypt and decrypt the room keys. It is important that each user that uses the client have a unique set of these keys. The user's keys also need to be the same each time they log in to the system. Therefore, the clients generate the needed keys based on the user's password and username. The client is also responsible for generating the room encryption key when a new room is created. This is done by randomly generating a room key to be used. Before it is sent to the server, it is encrypted with the creator's public key.

The clients use a local encryption module to make sure that the data downloaded from the system stays encrypted. When a file is downloaded from the system, the client will decrypt it using the room key of the room containing the file. The file will then be encrypted by the local encryption module using the user's local encryption key. The files can then be mounted as a file system and displayed within the client. When the user wants to upload the file to the system, the client encrypts the file with the room key and sends it to the server.

Server

The server communicates with the clients and is the communication hub in the system. It handles all the requests and makes sure that users requesting an action have the right privilege to perform it. The server also authenticates the users to make sure they are not being impersonated. In this process the server is responsible for generating random tokens that will be used to verify a user. When data is needed to perform a request, the server queries the database for the needed information. When parts of the system needs to store data in the database, they go through the server, which in turn writes it in the database. If a user wants to download a file, the server will download it from the object storage and serve it to the user.

Database

The database contains all the data and relations between the data that is required by the system. All the details of users and rooms will be stored in the database. The database will also keep track of which files are contained in each room as well as which users have access to the rooms. The database only communicates with the server and all the queries and writes are requested by the server. The structure of the database will be described in detail in section 4.5.

Replication Server

To change the room key used to encrypt the data within a room, all the data need to be decrypted and encrypted with the new room key. This is handled by the replication server (RS). The RS will generate a new random room key and replicate all the data in the room. While the key change is in process the users can only read the files. This allows the users to continue using the files encrypted with the old room key until the replication is complete. After the replication is complete the RS will verify and invite users that were in the old room for a set period of time. If the users have not requested anything from the room during this time period, they have to be reinvited to the room by the owner. In the case where multiple key changes have been requested in a short time period there will be several active instances of the replication server. The server will use the oldest, still active, instance to validate that the user was a member in the room before the key changes happened. The server will continue validating until the user has the latest room key. How the verification and invite process of the old users is preformed is described in chapter 4.6.

Object Storage

All the files uploaded to the system will be stored in the object storage. All the metadata associated with a file will also be stored in the object storage. The object storage will assign each of the uploaded files a unique hash that will be used as an id for the file. The hash will be created from the content of the file and its metadata. The id will then be passed to the server, which will store it in the database. When a user requests a file, the client will send the unique id for the file to the server. The id will be passed to the object storage, which will find and return the correct file. The file will then be returned to the user by the server.

4.5 Database Structure

The database is a central part in the system. It contains all necessary information for the system to operate. An *enhanced entity relationship (EER)* diagram for the database can be found in figure 4.2. The *id* attribute present in all tables is the primary key, a unique identifier for each entry in the table. The other attributes in each table will be explained in the following subsections.

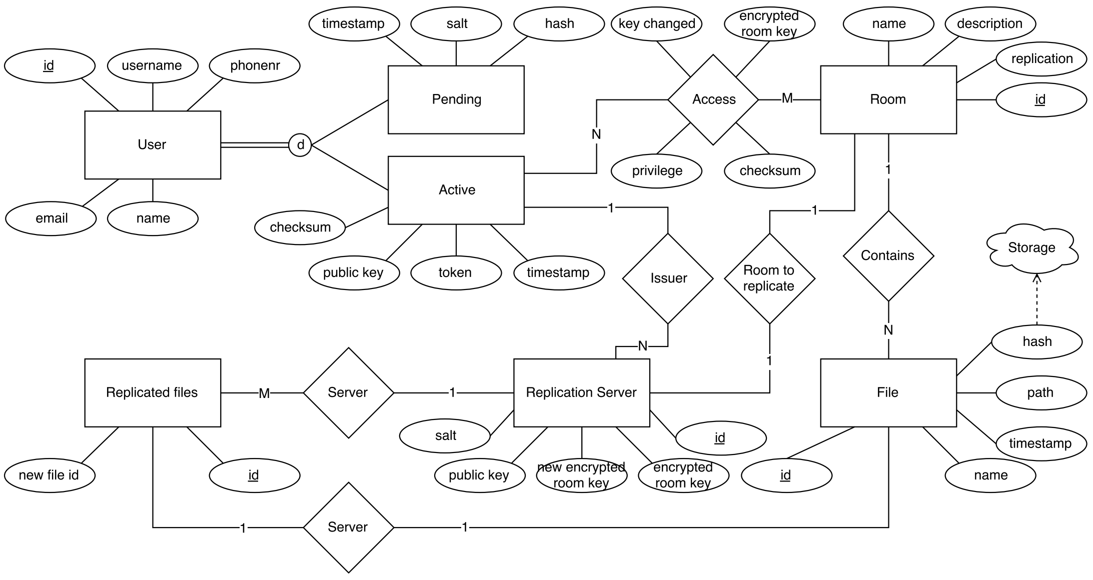


Figure 4.2: EER diagram for the database in the system.

Active and pending user table

The two user tables will contain information about each user. The tables will store username, phone number, email address and the name of the user.

The active user table keeps information about all the active users in the system. It has four additional attributes: checksum, timestamp, token and public key attribute. A checksum is a small sized data value which is used to verify data integrity. The checksum attribute will store a checksum of all the privileges the user currently have in all the rooms. The purpose of this checksum is to offer detection if privilege levels where to be changed manually in the database, this will be further explained in chapter 5. The token and timestamp are attributes used for authentication of the user. The token stores a randomly generated value and the timestamp the time of generation. How they are used is described in chapter 4.6. The final attribute, the public key, is the user's public key and is used when inviting a user to a room, for instance.

The pending user table keeps information about users who have still not activated their accounts. The pending user table has three additional attributes: hash, salt, and a timestamp. The hash attribute stores a hash generated by applying a hash function to a random value and a salt. The salt is stored in the salt attribute and the timestamp represents when the hash was generated. These attributes are used when a user wants to activate the account. How these attributes are used is further explained in section 4.6.

Room table

In the room table, general room information will be stored. The attributes include name, description and replication. The attributes name and description will be stored encrypted with the room key. This prevents system administrators to derive what the room might be used for from the name and description. It is still possible to see which users that are members in a room. The replication attribute is a flag that is set when the room is being replicated. If the flag is set, only read operations will be allowed until the room has been completely replicated. The purpose of the flag is to prevent data loss when a file has already been replicated, but someone makes modifications to the original file.

Access table

The access table keeps information about which users that are members in a room and what privilege level the different members have. The access table contains two foreign keys, user and room. A foreign key is an attribute that references a row in another table, in this case the user table and the room table. The encrypted room key attribute will contain the user specific encrypted room key, created by encrypting the room key with the user's public key. This encrypted room key is used when a user wants to perform actions on a file in the room. The privilege attribute represents the privilege level. The checksum is used to validate that the privilege attribute has not been changed manually, e.g. by a system administrator. The checksum is generated by applying a hash function on the user's privilege level, the user's public key and the room id. The final attribute is the *key changed* attribute. This is an additional flag attribute that is used to inform the server that the room key has been changed, and that the user has to verify that he has the old room key. How this flag is used is described in section 4.6.

File table

In the file table, all files uploaded to the object storage will have an entry. The file table has five different attributes: name, path, timestamp and hash. The name and path attributes contain the name and the virtual path of the file when mounted by the local encryption module. These two attributes will be encrypted using the room key. This prevents system administrators from gaining information about data kept within the system. The timestamp attribute represents the time of the last modification of the file. This attribute is used when a user wants to upload a modified file to ensure that the user has the latest version. If not the user has to merge the modified file with the one in the object storage before uploading. The final attribute, the hash, contains the unique identifier for the file in the object storage. This hash is used to fetch and upload files to the object storage.

Replication server table

The replication server table keeps track of all active RS instances. The table have two foreign keys, to a user issuing the request to use the RS and to the room to be replicated. Furthermore it has a public key, a salt, a new encrypted room key and a encrypted room key attribute. The public key attribute contains the RS's public key used for this replication. The other two attributes contain the old and new room key respectively encrypted using the RS's public key. The salt attribute is used when generating the RS's asymmetric key pair for the RS instance. The keys and the salt stored have an active role in the replication process and is further explained in section 4.6.

Replicated files table

The replicated files table is used to keep track of the files that have been replicated by a RS. It has two foreign keys. One to the replication server table for the RS responsible for the replicated file and one to the file that has been replicated. It only have one attribute, new file id, which is the new file id given to the replicated file.

4.6 System Flows

In this section the flows corresponding to the basic functions of the system will be described. The system flows describes what happens in the system when a specific action is carried out. For each system flow, the parts of the system involved in the action is shown. The flows also describe how the individual parts of the system handle the requests of a specific action.

Register and Activate an Account

The process of creating a new account in the system is divided into two step: registering the account and activating the account.

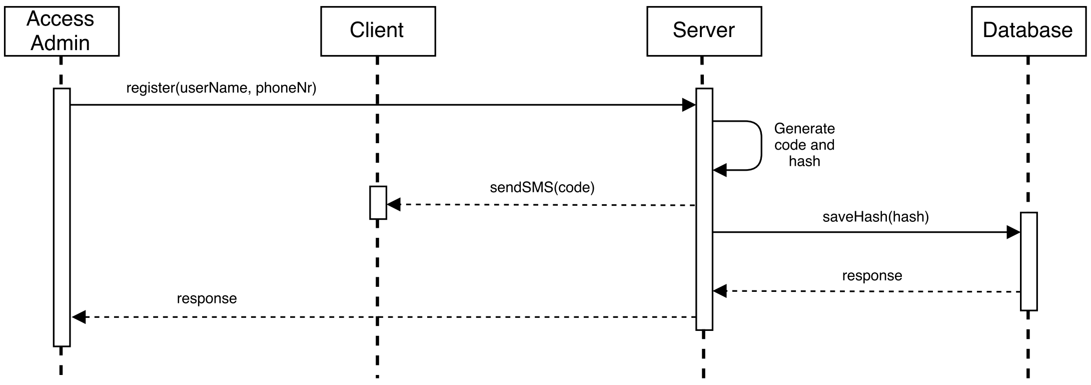


Figure 4.3: System flow for registering an account.

A new account is registered in the system by an access administrator. In this part of the process a username and a phone number for the new user is given to the administrator. These are then sent to the server. The server generates a random activation code and runs it through a hash function. The activation code is sent by SMS to the phone number provided and the hash of the activation code is stored in the database. The registration part of the process is now complete. This part of the flow can be seen in figure 4.3.

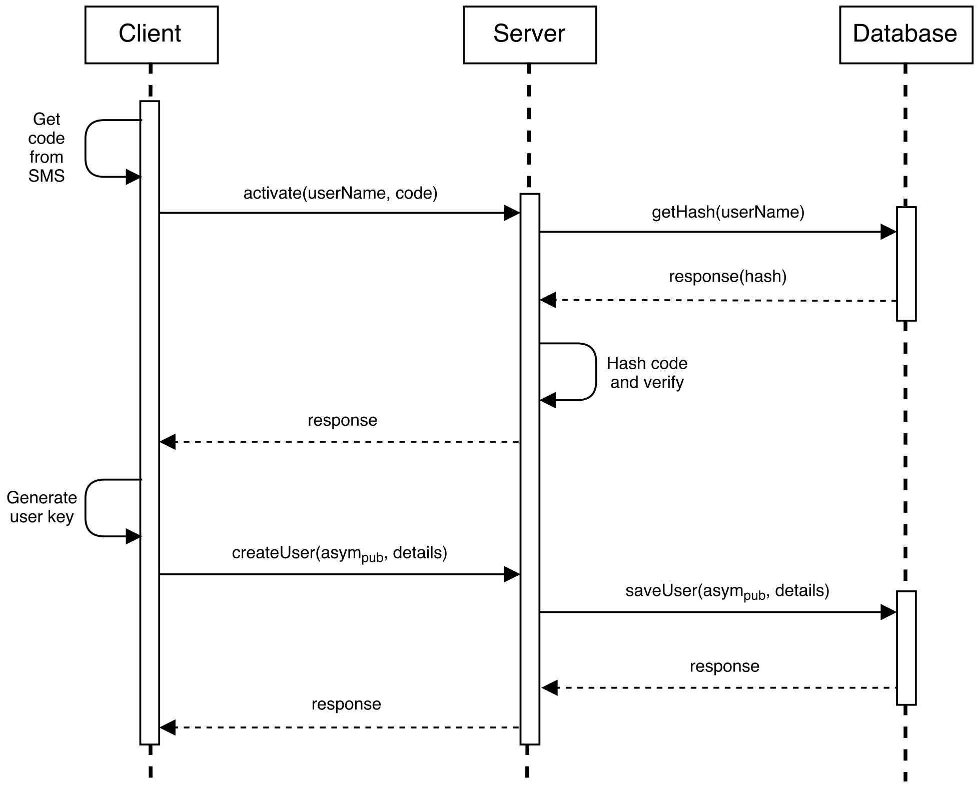


Figure 4.4: System flow for activating an account.

The new user can now activate the account in the system by using the received activation code. The user enters the activation code in the client, which passes it to the server. The server will then get the user's hashed activation code from the pending user table in the database. To verify that the user has provided the correct activation code, the provided code is hashed and compared with the stored hash. If the user did provide the correct activation code, the user can proceed to set a password for the account. The user will also provide details for the account, such as name and address. When the password is entered the client will generate the user's private and public key. The public key is sent to the server along with the entered account details. Finally the server saves the public key and account details in the database. The activation part of the process is now complete and the new account is ready to be used. This part of the flow can be seen in figure 4.4.

Authentication

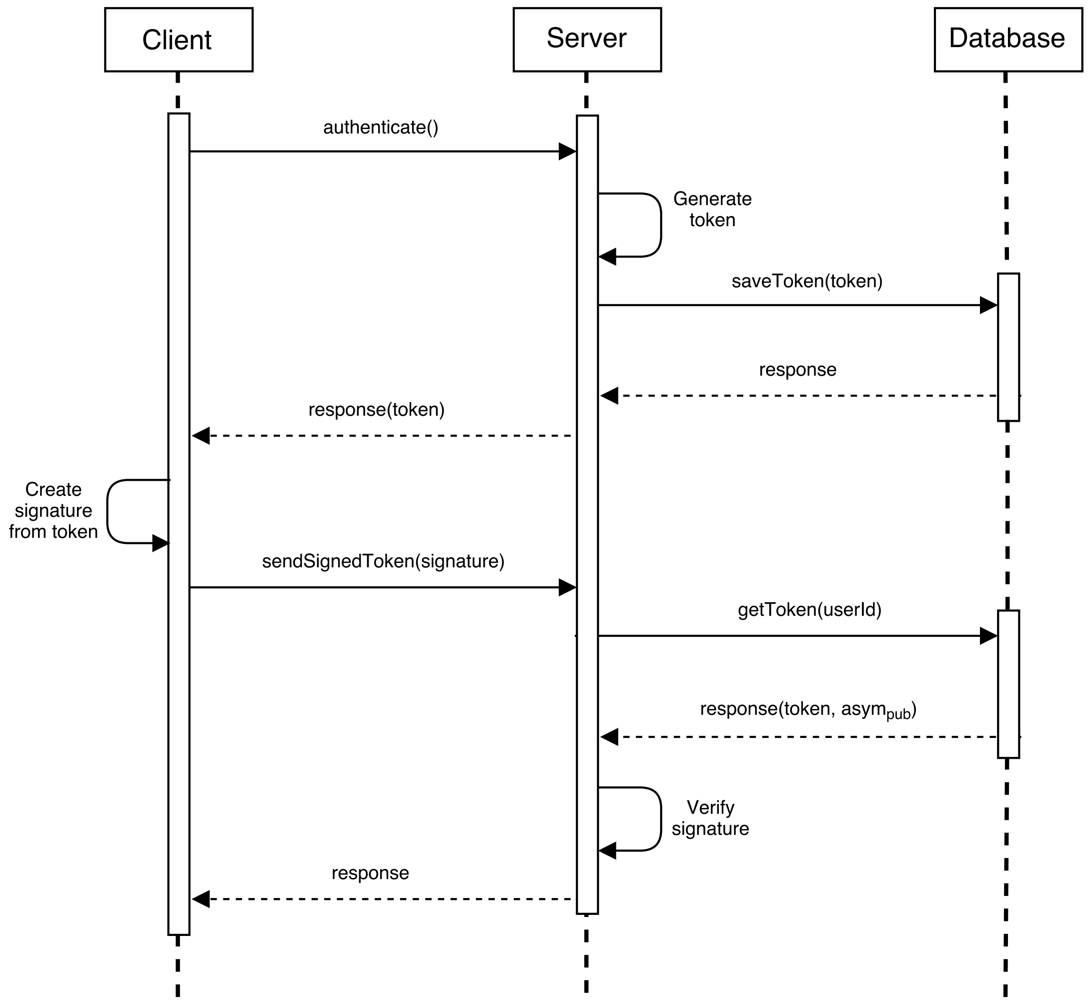


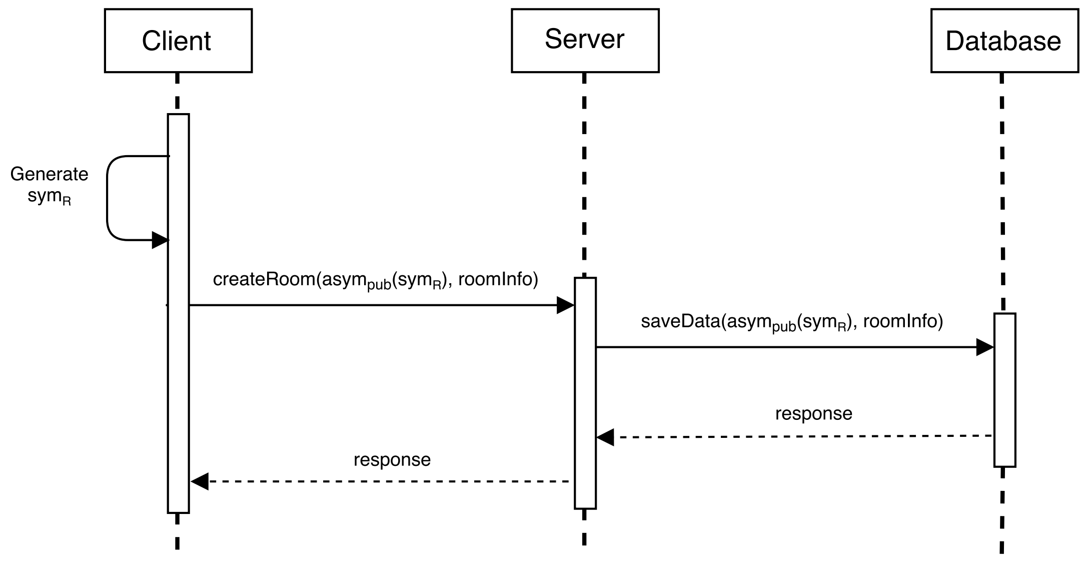
Figure 4.5: System flow for authentication.

The server authenticates the user by a simple challenge-response before an action is performed. First the server generates a random token and saves it in the database. The token is marked with a timestamp and is valid for a specified amount of time. This means that the server only needs to generate a new token if the previous token is out dated.

The server sends the token to the client, which create a digital signature from the token. The signature is sent to the server, which retrieves the user's token and the user's public key from the database. The user's public key is used to verify that the signature is in fact made by the requesting user and that it is made from the correct token. If the server can successfully verify the signature, the authentication is complete and the action can be carried out. The flow can be seen in figure 4.5.

In the following flows the authentication process will not be shown. It is assumed that the authentication process have been done before the start of the flows and that it was successful.

Create Room



Client
Server
Database
Generate sym_R
createRoom($asym_{pub}(sym_R)$, roomInfo)
saveData($asym_{pub}(sym_R)$, roomInfo)
response
response

Figure 4.6: System flow for creating a room.

When a user creates a room, the client starts off by generating the needed encryption keys. The user's key pair is generated as well as a random room key. When the keys are generated, the room key is encrypted with the user's public key. Details about the room such as a name and a description is also provided by the user. The user's encrypted room key and the details about the room is then sent to the server. The server sends a request to the database to create a new record in the room table containing the details associated with the new room. The server also updates the access table and stores the encrypted room key together with the privilege level of the user. If the system is setup to run in Enterprise Mode, the client also encrypts the room key with the keywarden's public key. The access table is then updated to also give the keywarden access to the room. When a new room is created, the user who created it will get owner privilege of the room. A response containing either "success" or an error message is then returned to the client. The flow can be seen in figure 4.6.

Invite a User to a Room

The owner invites a user to the room by the user's username. The username together with the id of the room, will then be sent to the server. The server will then verify that the user that requested the invite can in fact invite users to the room by checking the privilege level. If the user is an owner, the invited users public key and the owners encrypted room key will be sent back to the client. If the user does not have the required privilege level an error message will be returned to the client.

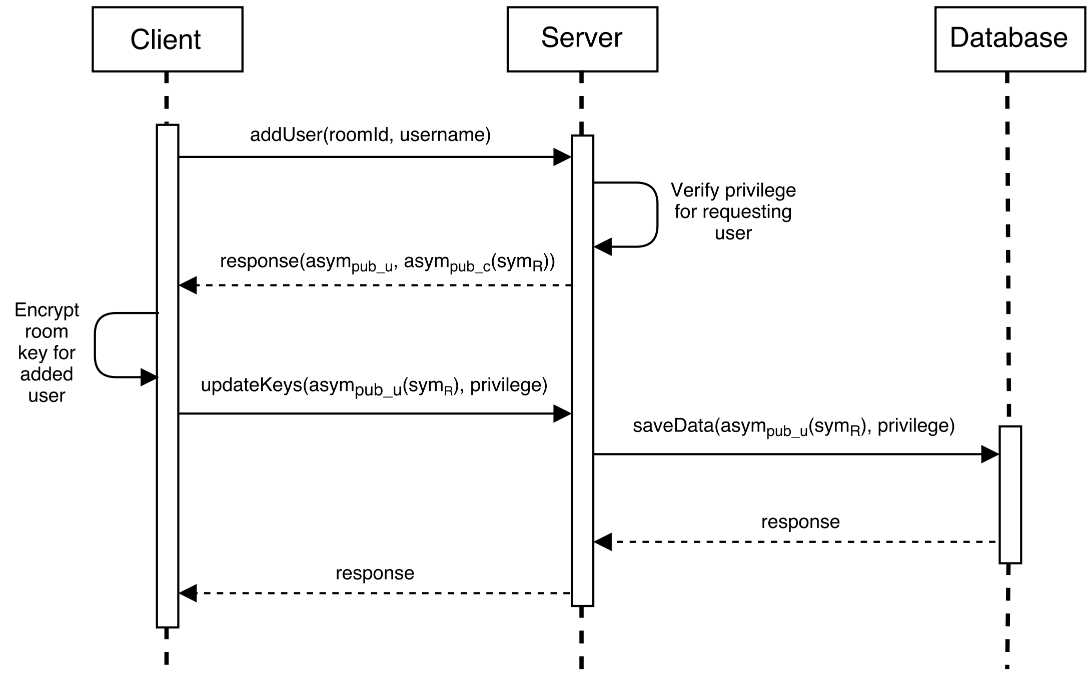

Client
Server
Database
addUser(roomId, username)
Verify privilege for requesting user
response($asym_{pub\_u}$, $asym_{pub\_c}$(sym_R))
Encrypt room key for added user
updateKeys($asym_{pub\_u}$(sym_R), privilege)
saveData($asym_{pub\_u}$(sym_R), privilege)
response
response

Figure 4.7: System flow for inviting a user to a room.

When the client receives the keys, the encrypted room key is decrypted using the owner's private key. The room key is then encrypted with the invited user's public key. After the room key is encrypted it is sent to the server together with the selected privilege for the invited user. The server saves the encrypted room key associated with the invited user with the privilege level in the access table in the database. The flow can be seen in figure 4.7.

Download File

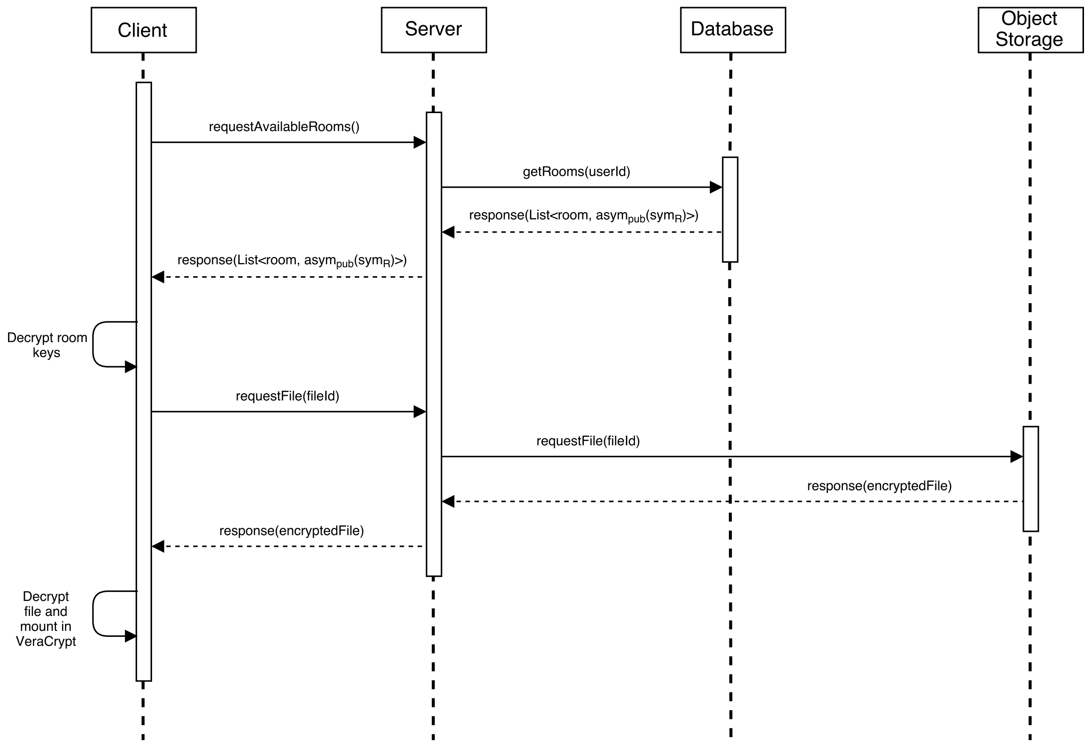


Figure 4.8: System flow for downloading a file.

When a client needs to download a file it will send a request for available rooms to the server. The server creates a list of rooms and encrypted room keys from the database using the user id corresponding to the issuer of the request. The list is then sent to the client. The client will use the user's private key to decrypt the room keys contained in the list in order to decrypt the room information. When the user selects a file for download the client will send a request to the server containing the corresponding file id. The server uses the file id to retrieve the encrypted file from the object storage and forwards it to the client. The client uses the room key to decrypt the file. The local encryption module then proceeds to encrypt the file with the local encryption key and mount it in the file system. The flow can be seen in figure 4.8.

Write File

When a user wants to upload a modified file to the object storage the client will first send a request to the server containing the room id. The server will verify that the user have the privilege to do so by checking the access table. If the privilege allows the user to write files, the server will retrieve the user's encrypted room key using the user id and the room id. The encrypted room key is then sent to the client. The client will retrieve the file to be uploaded from the local encryption module and decrypt the room key using the user's private key. The file is then encrypted using the room key. The encrypted file and the file id is sent to the server, which forwards them to the object storage. The object storage will replace the file corresponding to the file id and responds to the server reporting the success of the upload, which the server forwards to the client. The flow can be seen in figure 4.9.

When a new file is uploaded the procedure is similar. However, the file id is omitted when sending the encrypted file to the server. This tells the server it is new file. The server will send

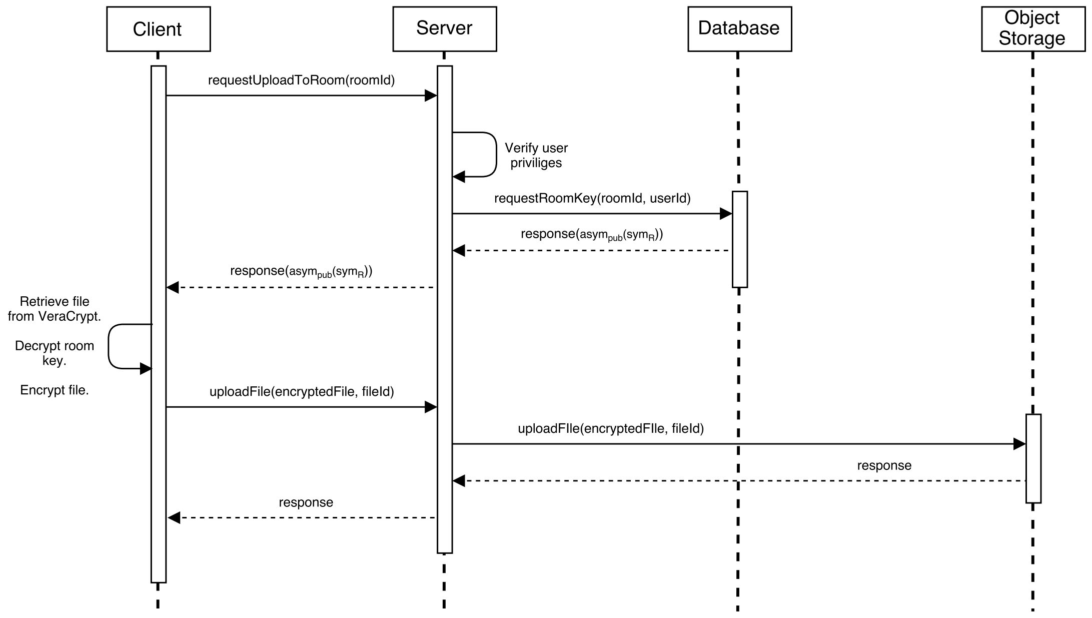

Client
Server
Database
Object Storage
requestUploadToRoom(roomId)
Verify user priviliges
requestRoomKey(roomId, userId)
response(asym_pub(sym_R))
response(asym_pub(sym_R))
Retrieve file from VeraCrypt.
Decrypt room key.
Encrypt file.
uploadFile(encryptedFile, fileId)
uploadFIle(encryptedFIle, fileId)
response
response

Figure 4.9: System flow for writing a file to a room.

the encrypted file to the object storage which will assign it a file id. The file id is returned to the server which adds a file entry in the database.

Change Encryption Key of Room

The process of changing the room key is divided into three parts: replication, removal and invitations.

The replication part of the process begins at the client. The issuer starts by generating a salt and a seed that will be used by the replication server. The seed is generated using a hash function by applying it to the password of the issuer and the salt. A room key change request is then sent to the server together with the seed, salt and room id for the room in question. The server verifies that the user is authorized to request the room key change. If so, the server adds the salt and room id to the replication server table in the database. The server proceeds to forward the room key change to the replication server, providing seed and room id. The replication server then generates an asymmetric key pair that will be unique for this replication. The RS responds to the servers request with the newly generated public key. The server adds the RS's public key to the replication server table in the database and sends a response to the client informing it that the RS is ready. The client invites the replication server to the room to be replicated using the public key belonging to the RS. The process of inviting the replication server is the same as inviting a user to a room described earlier. After the invitation is complete the client requests the server to start the replication. When the server receives the request it will send a request for a new room key to the RS. The RS generates a new room key that will be used for the replication and encrypts it using it's public key. The RS proceeds to send the encrypted room key to the server. The server adds the encrypted room key to the replication server table and sets the replication flag in the room table. The replication flag makes the files in the room read only while the server is replicating. The server then tells the RS to start the replication. The RS responds to the server telling it that it will begin the replication of the room. During this process an identical room will be built, the only difference is the room key. The server forwards the response to the client, ending the first part of the replication. The flow can be seen in figure 4.10

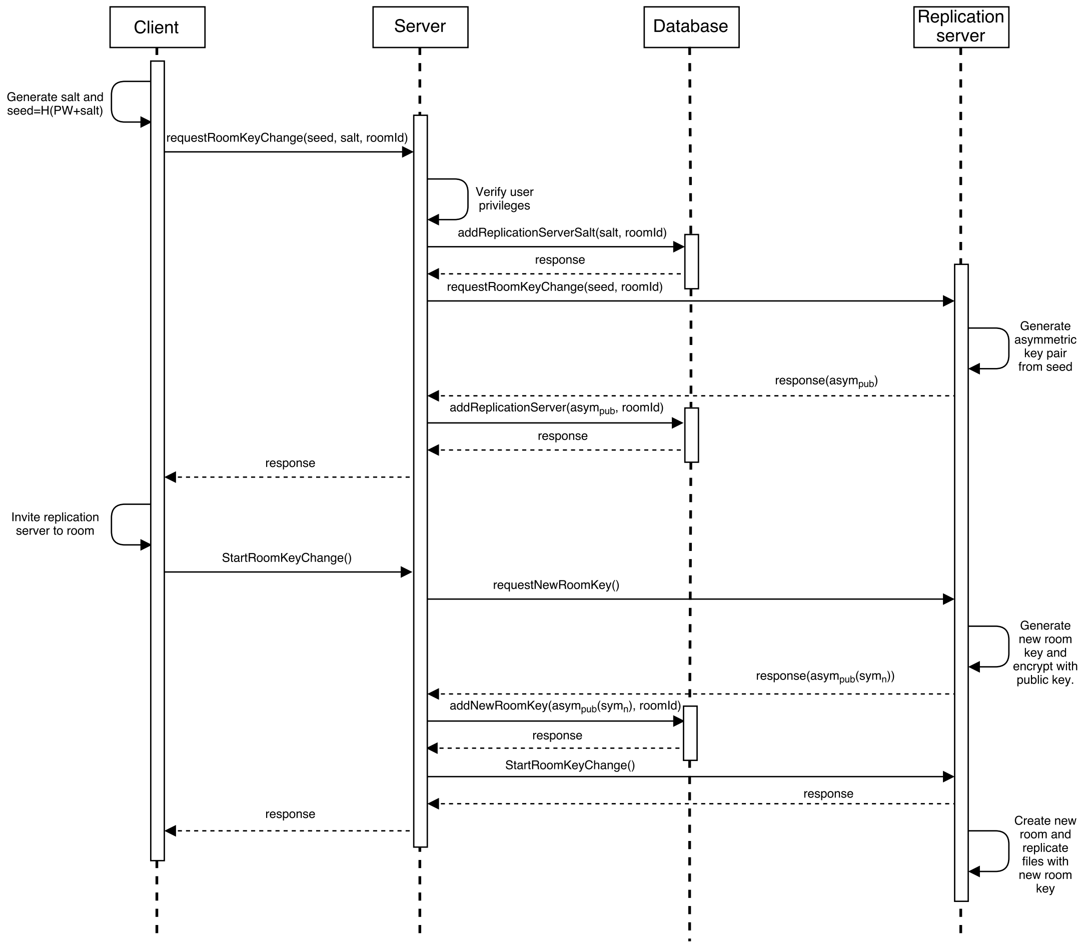

Client
Server
Database
Replication server
Generate salt and seed=H(PW+salt)
requestRoomKeyChange(seed, salt, roomId)
Verify user privileges
addReplicationServerSalt(salt, roomId)
response
requestRoomKeyChange(seed, roomId)
Generate asymmetric key pair from seed
response(asym$_{pub}$)
addReplicationServer(asym$_{pub}$, roomId)
response
response
Invite replication server to room
StartRoomKeyChange()
requestNewRoomKey()
Generate new room key and encrypt with public key.
response(asym$_{pub}$(sym$_n$))
addNewRoomKey(asym$_{pub}$(sym$_n$), roomId)
response
StartRoomKeyChange()
response
response
Create new room and replicate files with new room key

Figure 4.10: System flow requesting a room key change.

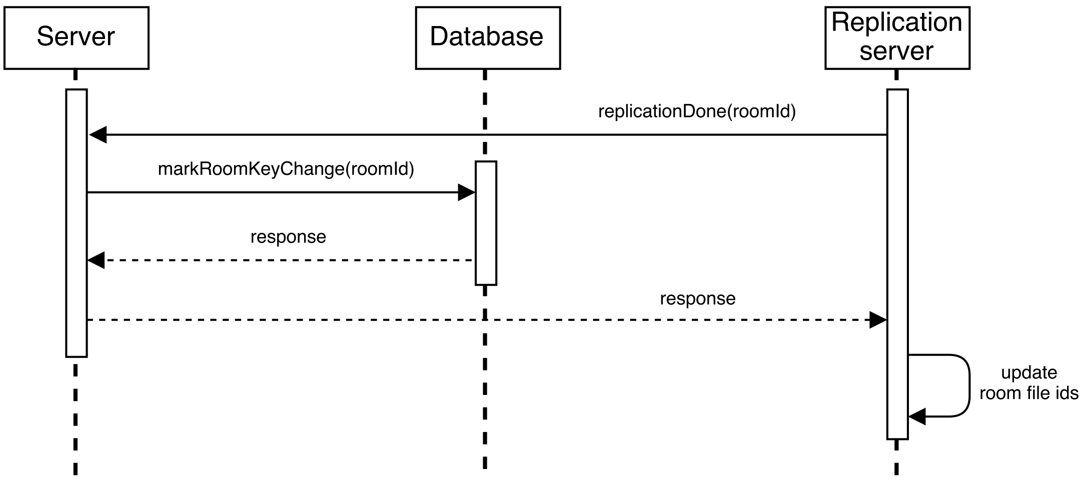


Figure 4.11: System flow for replication complete.

When the replication is complete the next step of the process begins, the removal step. The RS sends a message to the server telling it that the replication is completed. The server will mark all users in the access table belonging to the room with a verification flag. This flag will be used in the invitation part of the process. The server also removes the replication flag in the room table, removing the read only limitation. The server sends a response to the RS telling it the flagging is complete. The RS will then replace the file ids belonging to the old room with the replicated files' ids. The flow can be seen in figure 4.11. Note that the user's encrypted room keys are untouched during this step.

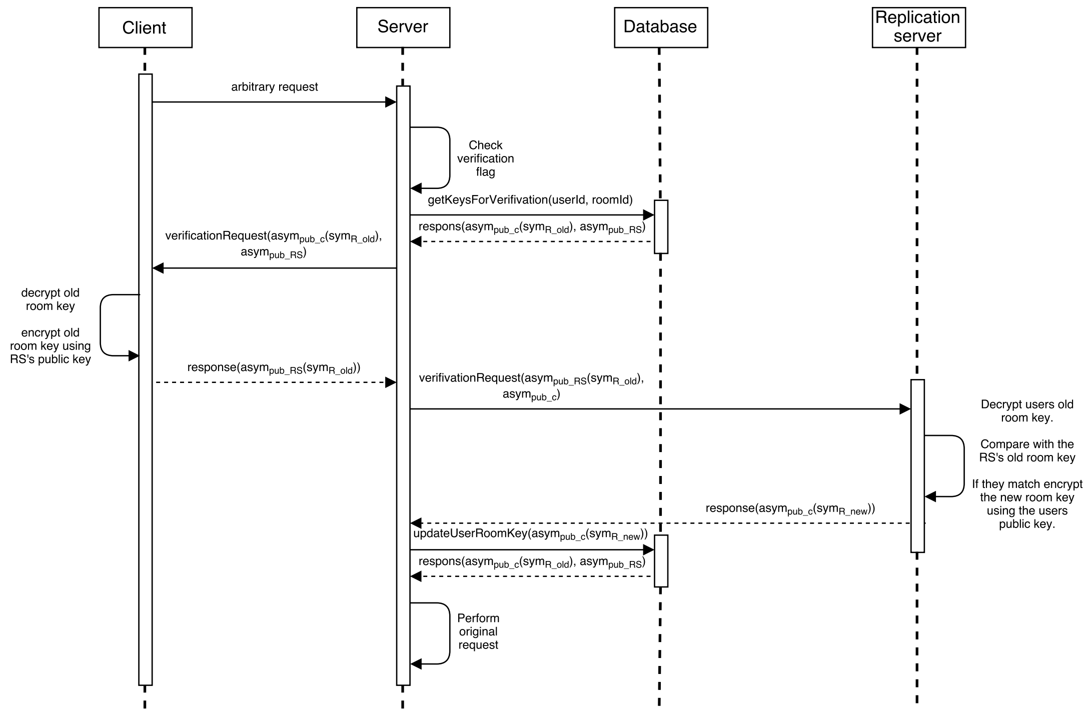

Client
Server
Database
Replication server
arbitrary request
Check verification flag
getKeysForVerifivation(userId, roomId)
respons($asym_{pub\_c}(sym_{R\_old})$, $asym_{pub\_RS}$)
verificationRequest($asym_{pub\_c}(sym_{R\_old})$, $asym_{pub\_RS}$)
decrypt old room key
encrypt old room key using RS's public key
response($asym_{pub\_RS}(sym_{R\_old})$)
verifivationRequest($asym_{pub\_RS}(sym_{R\_old})$, $asym_{pub\_c}$)
Decrypt users old room key.
Compare with the RS's old room key
If they match encrypt the new room key using the users public key.
response($asym_{pub\_c}(sym_{R\_new})$)
updateUserRoomKey($asym_{pub\_c}(sym_{R\_new})$)
respons($asym_{pub\_c}(sym_{R\_old})$, $asym_{pub\_RS}$)
Perform original request

Figure 4.12: System flow for verifying that a user belongs to a room.

When a user belonging to the old room requests an arbitrary operation concerning the old room the last step will be preformed. Instead of executing the requested operation the server will send a request to verify the old room key to the client. Along with this request the server sends the RS's public key and the user's encrypted room key, which is now the old room key. The client decrypts the old room key using the user's private key. The client proceeds to encrypt the old room key with the RS's public key and sends it to the server. The server forwards the message together with the users public key to the RS. The RS verifies the old room key sent by the client using its old room key. If the old room keys match the RS will encrypt the new room key using the users public key and send it to the server. The server will update the users encrypted room key in the access table with the new encrypted room key. The flow can be seen in figure 4.12.

4.7 Discussion

Design Decisions

In the following subsections the major decisions made during the iterative design process will be presented.

Object storage

The initial storage solution in the system was a conventional file storage. However, as the design took shape limitations and problems with a file storage was unveiled. Since the system shall be able to store large quantities of files, the file and folder hierarchy in the file storage needs to be maintained in order to prevent broken file paths etc. In a system where this hierarchy could get complex with the creation of rooms and folders in the rooms this was something we wanted to avoid. The object storage solved this problem by allowing files to be stored in a flat structure which eliminates the maintenance of a hierarchy. Instead the files where accessed by an id assigned to the file so no broken paths are possible. Furthermore the object storage allows for easier replication since that is a feature built in to the object storage unlike a file system where this process also has to be created.

Replication server

In the first revisions of the design the replication was performed by a client on request by an owner. This approach had a few drawbacks that we were not satisfied with. First of all it could potentially use a lot of the resources on the workstation where the client is running. Since a workstation usually have limited resources, this result in a poor user experience. An even greater problem is that the user must be online through the entire process. Since a room could contain files with a combined greater size than that available on the client, the client may not be able to keep the replicated room locally. Instead it would have to replicate a file and send it to the storage before starting with the next file. With the replication server we opted to use these problems were eliminated. Furthermore this also made the process seemingly transparent to the user since the replication is performed in the background without it affecting the user or clients. However, the use of a replication server required additional flows to be created in order to share room keys and avoid storing the new key unencrypted. We deemed that the extra complexity brought on by the replication server outweighs the poor user experience that would be caused by a client doing the replication

Digital signatures

For the authentication process we decided on using digital signatures instead of a classic user name and password solution. One of the reasons for choosing this approach was that we did not need to store any passwords in the database. Instead we could generate a key pair used for the digital signatures to be used during a session. The public key could be stored in the database for the duration of the session without the security of the system being compromised. Even though it is not trivial to crack hashed password stored in a database, this design mitigates those threats.

Hybrid encryption

When we started out designing the system, we had to decide on a way to handle the encryption of data. During the first iterations of the design we used purely symmetric encryption and a key exchange between the client and the server was needed. Using this approach became a problem when users needed be invited to a room since both of the users had to be online in order for the key exchange to happen without any keys being stored in the server. For the next iterations we changed the design to use a hybrid encryption scheme in order the server to store the keys encrypted with a users public key.

Related Work

There exists several other systems for secure storage. SpiderOak have designed a system which offers high security and that handles all the encryption on the clients [24]. The design of the system offered by SpiderOak is similar to the design of the system proposed in this thesis. It is also based on a hybrid encryption system and stores the user's encrypted encryption key on the server. As in the proposed design, the user's password is used to generate the keys needed to decrypt the keys received from the server. Up until recently, there was also a system called Wuala that offered an encrypted storage [11]. When they started out, a peer to peer approach was used to store the encrypted files. A centralized server was also used to store the user's encrypted keys and the meta data for the files in the system. The design was later changed to a more strict client-server approach. Both SpiderOak and Wuala influenced our design in different ways. The client-server architecture was chosen due to SpiderOaks architecture and Wualas transition to a similar architecture. The peer to peer approach was discarded since it would require peers to constantly be online to be able to access data. SpiderOaks hybrid encryption system inspired us to design a similar system which also used hybrid encryption since it enabled us to fulfill many of our requirements. Although the system is influenced by SpiderOak, and to a lesser extent Wuala, the application of the hybrid encryption system is by our design.

Conclusion

The designed system resembles a hybrid system described in section 2.2. The room key, which is used for symmetric encryption, is stored encrypted with an asymmetric encryption scheme, where the key used is a user's public key. This approach has two main advantages in this system. Firstly, it protects the stored room keys in the database, since they are encrypted. Secondly, it makes collaboration in the system simple. The room keys in the database are already protected and can safely be sent to the client. The public key of a user is also safe to store and send since it only can be used for encryption or verification of digital signatures. Therefore it is simple and safe to send an encrypted room key to a client together with a public key of a user to invite them to a room.

Regarding the functional requirements of the system, they are all possible in the proposed system design as can be seen in the system flows. The matter of fulfilling these requirements is therefore a question of implementations. This is also the case for non-functional requirement 1: *No encryption keys shall be stored on the clients.* Non-functional requirement 3: *The system shall be possible to run as a distributed system.* is not part of the design. This requirement can be fulfilled by introducing multiple servers, databases and object storages and using a load balancing server to route the traffic. Also database and object storage replication has to be introduced to keep the system consistent.

For the design to remain secure it is assumed that the registration and handling of the keywarden account is done correctly. It is up to the adaptors of the system to create secure policies for these tasks. The keywarden's credentials needs to be kept secure to avoid manual modifications. However, this aspect is outside the scope of the design and is a question of how this particular entry is managed. Whether it is stored in a protected directory, on a dedicated key server or in a hardware security module attached to the server is a question for the adaptors. Potential security risks could be registration of a fake account, impersonating someone. If a owner is tricked to invite a fake account to the room, the confidentiality of the room's content is compromised.

5 Threats and mitigations

5.1 Introduction

The following chapter will describe the observed threats and how the design mitigates these threats. The process that have been used to analyse the threats will be presented to get an understanding of how the threat analysis was conducted. The chapter will also present a system-wide analysis of the requirements concerning security and whether the design fulfill these. Finally, the chapter will contain a discussion about the threats and their mitigations.

5.2 Process

The process of threat identification and mitigation was preformed during the design phase of the thesis. The design phase was an iterative process were multiple drafts were produced. The changes made to the design was mainly caused by unveiled threats, and mitigations to these. When a threat was identified, the risk was analysed by an informal discussion. During these discussions, the probabilities and the affects of the threats was assessed. In cases where the risk of a threat was deemed unacceptable, the threat were mitigated which resulted in a new design. In some cases, where the risk was deemed acceptable, detection of these threats were incorporated into the design. The proposed design is the result of this iterative process including the threat analysis and mitigations.

5.3 Threats

The following subsections describes the threats and potential consequence that have been taken into consideration when designing the system.

Database manipulation

One of the requirements of the system is that it shall not be possible to modify the database to give an unauthorized user access to data. If an administrator or outside attacker can give access to files just by modifying the database, the impact would be huge. The privacy of the users and the confidentiality of the files stored within a room would be greatly compromised.

File leakage

Files that leak from the system can cause significant damage to an organization, especially if the files are confidential. Files can leak in numerous ways. The storage media can be stolen, i.e. physical theft. An administrator managing the storage can access the files and distribute them, e.g publish them on the Internet. In more extreme cases, an outsider can abuse a security flaw to gain access to the files and then distribute the files. Regardless of the cause to the file leak, this is a threat which can have a high risk.

Cryptographic key leakage

The whole system revolves around the use of cryptographic keys. The foundation of the security in the system relies on the keys being kept secret. If a cryptographic key used for file encryption where to be leaked the security of the file can be compromised, given that an attacker has access to the encrypted files. Furthermore, a user's asymmetric key pair could be leaked. If such a key pair where to leak, an attacker could act as an impostor of the user, abusing his account in the system. These keys could potentially be stolen from a database where the keys are kept or be extracted and leaked from the memory of a client.

Eavesdropping and MITM

Information that is transmitted between the client and the server may be listened to or altered. If communication is successfully eavesdropped, the attacker can steal files, encryption keys or tokens that are being transmitted. The keys and tokens that can be gathered from these attacks can be used to either decrypt information or authenticate as another user.

Gather sensitive information from the database

Data stored in the database can be of sensitive nature on its own. In addition, data stored can be used to draw conclusions about user patterns. For instance information about what a particular room contains, e.g. room name, room description or file names, in combination with members of the room can be used to gain information about what the room is used for. This information can be used to blackmail a user to give up information or to select a target room for an potential attack.

5.4 Design mitigations

In this section mitigations to the threats presented above will described.

Hybrid encryption

In order for the system to allow collaboration, encryption keys used on the files have to be shared among the users. This means that the keys have to be stored so the server can serve the keys to the requesting users. To avoid keys leaking from the database a hybrid encryption scheme was adopted. In this scheme the file encryption keys are encrypted with a user's public key. This entails that the keys can safely be stored in a database since only a user with the corresponding private key can decrypt the file encryption key. The hybrid encryption scheme also protects against eavesdropping and man in the middle attacks. Since the file encryption key will be sent encrypted to the user, and eavesdropper or man in the middle will not be able to get hold of the file encryption key. As long as the user's private key is kept secure, the keys stored in the database and sent to the client cannot be used by an attacker.

Encryption

To keep a user's information in the system private symmetric encryption is used. All the files is encrypted at the client and stored encrypted in the object storage. This means that the only ones who can decrypt and read the file are users who have the room key. Since the room key is encrypted, the only way to get access to it is to be invited into the room. Storing encrypted files also mitigates the threat that an attacker eavesdrop during transmission and reads the data, since the files are transmitted encrypted.

Encryption is also used to mitigate that sensitive information is gathered from the database. Information regarding file names, room names and descriptions have been encrypted so that only authorized users can read it. This means that even if an administrator have full access to the database it is not possible to draw conclusions regarding the content of a room based on the information in the database.

SSL/TLS

To protect the system from man in the middle attacks during activation code transmissions, authentication token transmissions or other request and response data, SSL/TLS will be used. As described in section 2.5 SSL/TLS uses asymmetric encryption and a certificate authority to distribute the public key. SSL/TLS effectively prevents a man in the middle attack since the attacker needs to have a valid certificate or the client will not accept to use the SSL/TLS connection. This means that tokens and activation codes can be sent safely to the server from the client without worrying about eavesdroppers or men in the middle.

Checksum verification

Some information regarding a user is not encrypted in order for the server to easily handle requests. An example of this is the user's privilege level in a room. There is a possibility that the privilege is modified in the database. Therefore, a checksum is used to verify that it has not been tempered with. The use of a checksum can detect database manipulation by malicious administrators or outside attackers. This checksum can of course be recalculated, eliminating the detection. However, there is not much gain in this attack since the user already have to be a member of the room. The privileges can only be elevated and such an attack can not be used to gain access to files the user would not already have access to. Therefore, the proposed design only offers a way to detect the manipulation so it can be restored.

5.5 System security analysis

The most important security aspect of the system is that no unauthorized user shall be able to gain access to confidential information. To evaluate if this aspect is fulfilled, an attack tree analysis was conducted. An attack tree represent attacks against the system in a tree structure. The goal of the attack is the root node and actions to achieving the goal is represented in the child nodes. Each node is marked with either a *P* or an *I*, representing possible or impossible respectively, depending on the deemed possibility of the action. The evaluation of the nodes begins in the leaf nodes and traverses upwards through the tree. If a child node is deemed possible the parent node is also deemed possible. Nodes connected with a *&* node means that the child nodes both need to be deemed possible to make the parent node possible. The resulting attack tree for this system is presented in figure 5.1.

The attack tree shows that unauthorized file access can be achieved by only one action. If an authorized user decides to spread the files or being forced to give access to the files, unauthorized users can gain access to the data. This is not something the design can protect against and it is somewhat out of the scope of the system.

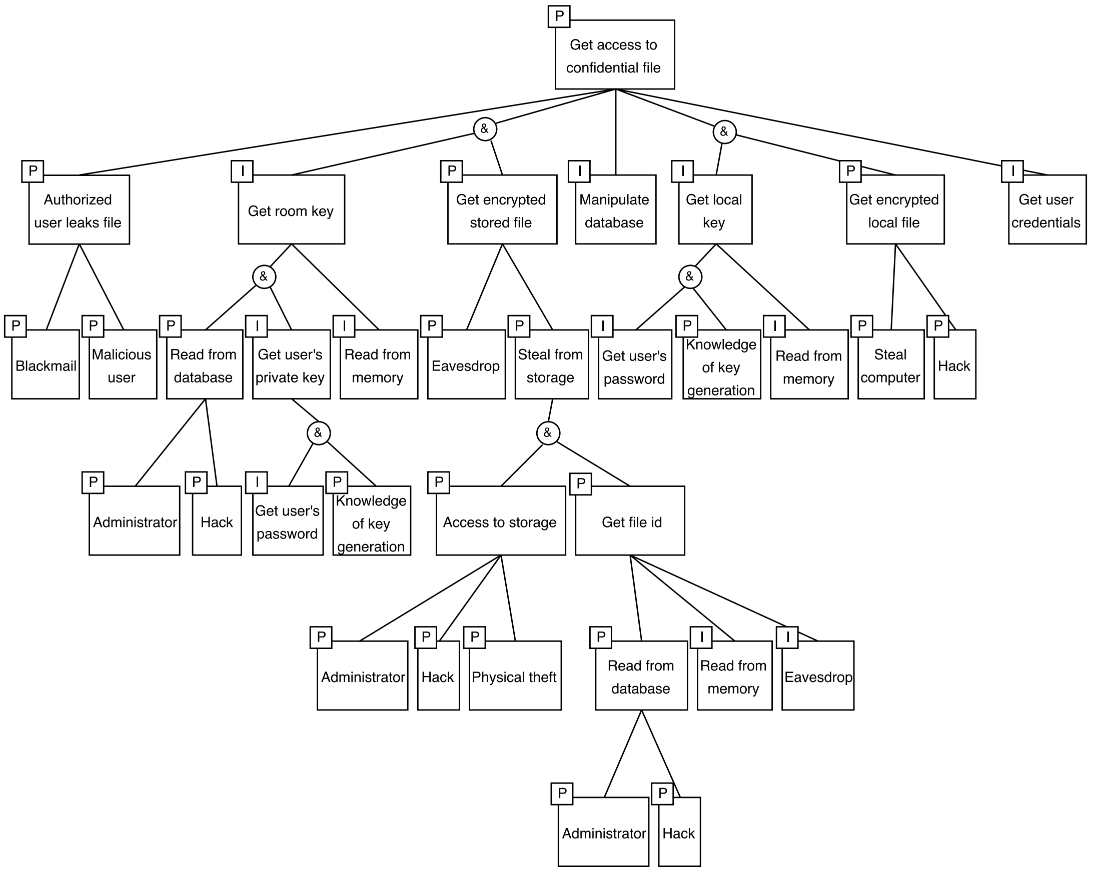


Figure 5.1: Attack tree for unauthorized file access.

The rest of the top level actions are deemed impossible to use in order to gain unauthorized access to a file. Since the whole design is based on that the user's password is kept secret and that it can not be derived from the system, the possibility to get hold of a user's credentials have been discarded. It is in theory still possible, but it comes down to how the users handle their credentials and the environment containing the client. An attack can for example plant malware, such as a key logger, on a user's computer to get the user's credentials. This is however not a flaw in the design of the proposed system. To protect against these kinds of attacks a suitable policy regarding the usage of the environment is needed. This is however outside the scope for the thesis.

Two of the actions are two part actions, where both parts need to be possible in order for the action to be successful. These actions are based on first gaining access to a encrypted file and then gain access to the key it is encrypted with. In both cases the attack tree shows that the encrypted file may be acquired but it is impossible to find the key to decrypt the file. The reason that it is impossible is because the keys are derived from the user's credentials, which where assumed impossible to get. There is no problem that the encrypted file can be acquired since it is encrypted. If modern encryption algorithms is used it is infeasible to extract the plaintext. It is also deemed impossible to extract the encryption keys from the memory of the machine running the client. This can be prevented by implementing the client so that the encryption key is generated when needed and then the memory used for the key is overwritten. By doing so, the key will only be in memory for a short period of time, making it far less likely to be extracted.

Furthermore, manipulating the database in order to get access to files is assumed impossible. The encryption keys for the files is stored in the database encrypted with a user's asymmetric key. This makes the action impossible since the attacker needs to first be able to decrypt the encryption key, which will be infeasible if a modern encryption algorithm has been used. The only way to gain access to a file is by being invited by a user who can decrypt the used encryption key.

5.6 Summary

From the threat analysis preformed during this thesis, the security of the system is deemed good. The proposed design mitigates the threats that have been identified and the attack tree reveled no design related weaknesses. There is still ways to get access to confidential files from an authorized user, either by blackmail or if the user decides to go malicious. However, this is beyond the scope of this thesis and will not be addressed. Furthermore, the design fulfills the requirement that no system administrator shall be able to access confidential information. The requirement stating that no encryption keys shall be stored on the clients has also been achieved. There is no need to store a key on the client in the proposed design, which makes it harder to extract keys from the memory. By using reverse engineering it can be possible to get knowledge of the key generation process. However, it is impossible to generate the used encryption keys with this knowledge since the attacker also needs to have the users credentials to generate the keys. As previously mentioned, user credentials are assumed to be kept secret. Although the threats identified have been mitigated, it is possible that other threats exist which can affect the security of the system. As of writing however, the system is deemed secure.

6 Implementation

6.1 Introduction

In the following chapter the implementation phase of the thesis will be described. This includes the development of the different software components, delimitations of the implementation phase, testing of the developed system and a discussion about the implementation.

6.2 Delimitations

Due to the limited time frame available for an implementation during the thesis, only a proof of concept (PoC) has been developed. This means that certain aspects of the design has intentionally been left out. The main focus has been on providing core functionality such as encryption and collaboration on files.

The replication server (RS) which shall be used to change the room key has been left out. The reason being it will operate independently of the core system and is a large component to implement. Therefore it was discarded and additional time was put into the development of the core functions. The same reasoning has been applied to the distributed system requirement. The overhead of setting up and develop replication features for a distributed system was simply to large to manage in the limited time available. Communication using TLS/SSL was also discarded due to the need of third parties and the configuration time it would require. Additionally, recovery mechanisms and error handling had low priority throughout the development to allow focus to lie on the feasibility to implement the design.

6.3 Motivations

During the implementation, certain decisions had to be made regarding what encryption algorithms to use throughout the system. The following subsections will explain our reasoning behind the choices of AES and RSA.

AES

When deciding on which algorithm to use in the developed system, two main aspects were considered. The first being the security of the encryption and the second being the efficiency and performance of the algorithm. With these two aspects in mind, AES was chosen as the symmetric encryption of the developed system. Since AES is named the current standard of symmetric encryption by National Institute of Standards and Technology (NIST) [26], the encryption algorithm is widely used and considered secure. The algorithm is also implemented in modern processors, which greatly increases the performance of the algorithm.

RSA

RSA was chosen as the asymmetric encryption algorithm for the system because of three reason. The first being that NIST recommends the usage of RSA, together with Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA), for digital signatures [6]. Since the asymmetric encryption algorithm used in the system will be used for both digital signatures and encryption, the other two alternatives were discarded. The choice therefore came down to RSA or ElGamal, which both have the possibility of creating digital signatures and to be used for encryption. When conducting the literature study it was concluded that RSA had seen a wider usage than ElGamal [1]. Due to this, RSA has been a target for analysis to a much larger extent than ElGamal. This increases the probability that a weakness will be found and mitigated, which can increase the lifetime of RSA.

6.4 VeraCrypt

In this thesis VeraCrypt was used as a local encryption module. VeraCrypt is an open source software for disk encryption [30]. VeraCrypt is used for creating and maintaining encrypted disk volumes. It supports encryption with three different algorithms: AES, Serpent and Twofish. The user can also choose to use a combination of the three in various ways. For generating the encryption key, a selection of hash functions is supported as a step of creating random data. VeraCrypt also have support for hardware accelerated AES encryption. VeraCrypt will automatically use hardware acceleration if it the processor used supports it. This will increase the encryption and decryption speed of AES by up to eight times compared to when preformed by a pure software implementation. This means that the speed of AES is significantly faster than the other available encryption algorithms.

By using VeraCrypt the user can create a so called *encrypted container*. This is basically an encrypted file that can be copied, moved or deleted just like a regular file. When the container is created, the user has to choose which of the available encryption algorithms that will be used for the encryption of the container. The user also has to choose a hash function that will be used in the process of generating the encryption key. Next the user have to provide a password for the container, as well as some random mouse movements. The password, mouse movements and the hash function will be used in order to generate an encryption key for the container. VeraCrypt can now be used to mount the encrypted container as a disk volume. When mounted, the encrypted container can be used as any disk volume; files can be added, copied and opened. The mounted container will always remain encrypted. VeraCrypt handles this by on-the-fly encryption and decryption. If the user access an encrypted file in the volume, VeraCrypt will automatically decrypt and place it in memory. The application used to open the file can then access the unencrypted data in the memory. In the same way, VeraCrypt will automatically encrypt a file that is moved to a mounted, encrypted disk volume.

6.5 Software components

The client was written in the Go programming language as a command line interface. A simple GUI was also developed using JavaScript that communicated with the command line interface. To perform encryption and decryption, the libraries for AES and RSA built in within the Go core was used. For encrypting the downloaded files locally, Veracrypt was used. Like the client, the server was written in the Go programming language. The server was divided into three different modules; a communication module that receives the messages and spawns a new process for each request, a database handler module and a module for handling the connections to the object storage. The developed client and server communicates by JSON-formatted messages over a TCP connection. PostgreSQL was used as the database in the system developed during the implementation phase.

6.6 Testing

Tests where performed in two ways, continuously during the development and a larger stress test towards the end. The tests preformed during the development was performed locally when new functionality had been added using two different techniques depending on their suitability. Unit tests was used for independent modules such a the encryption module and the object storage module. Functionality that preformed operations throughout the system was tested using exploratory testing, for instance after adding functionality to upload files to the object storage. These tests was performed and decided upon by the developer implementing the functionality. Due to the lack of recovery mechanisms and error handling these tests was preferred since the developer had insight about limitations of the implementation.

The stress tests performed targeted the server, database and object storage to ensure that the developed system could handle larger loads than individual users. These tests was performed on a deployed server and database as well as a live object storage with a dedicated server creating multiple threads running the client software. Three different kinds of stress tests where performed:

1. Upload 500 100kB files simultaneously.

2. Upload 500 1GB files simultaneously.

3. Preform 1000 simultaneous write request to the database.

6.7 Discussion

The purpose of the PoC was to verify if the proposed design could be realized in an implementation and what problem areas that might exist.

The main problem that needed to be solved during the development was how large data files could be encrypted/decrypted and how a checksum of the encrypted files could be calculated at the client. Large files could not be read into memory and encrypted as a big chunk. This was solved by creating a data stream of the local file, as can be seen in figure 6.1. Before sending the stream over the TCP connection, it streamed through the encryption module which used AES in stream mode to encrypt the data of the stream. The data stream was then forked, one stream was used to calculate the checksum of the encrypted file while the other was sent to the server. When the server received the stream, it forwarded the stream to a temporary container in the object storage. When the file was uploaded, the server received a checksum of the file from the object storage. The server compares the checksum from the client with the one from the object storage. If they are equal, the file is moved to a permanent container in the object storage. When downloading a file in the system, the same solution is used but in the opposite direction.

Aside from the encryption problem for large files, no problems regarding the implementation of the design was unveiled. All other functionality that was implemented in the PoC were possible to implement as described by the design. However, as mentioned in section 6.2 not all parts of the design were implemented. From the implementation performed however, it was concluded that the parts omitted won't pose a problem. Despite the operation

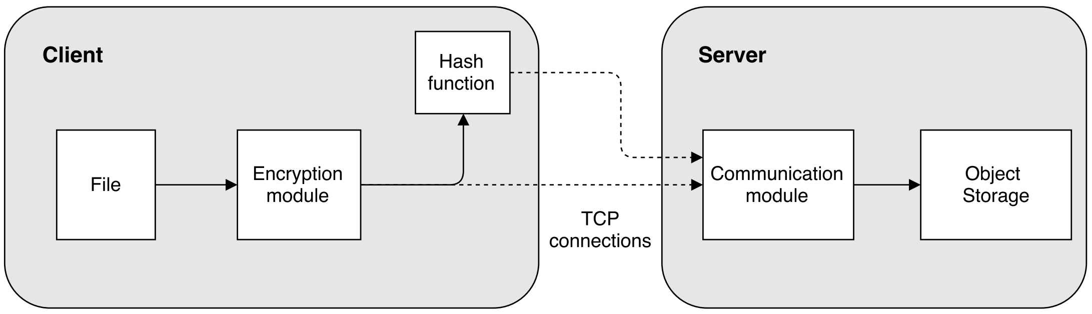


Figure 6.1: Data stream used for uploading a file in the developed system.

and initiation of the RS being one of the most complex flows in the design, many of the features required have been implemented in other parts of the system, which indicates that they will be possible to realize. The transition to TLS/SSL from the usage of plain TCP will not affect the design, the only aspect affected is the communication between the server and the client. This change will only increase the security of the system. Finally transitioning from a client-server system to a distributed system is deemed possible with the proposed design, assuming the distributed system is implemented properly. The flows described in chapter 4 can still be used without modification in a distributed system. This is due to the fact that a distributed system is transparent to the client, i.e. the client is only aware of the server it is communicating with.

7 Conclusion

In this chapter the results will be summarized and the conclusions presented.

7.1 Literature study

In order to design the desired system, a literature study was performed. The areas studied was mainly associated with cryptography but also available storage solutions. Common attacks were studied to get an idea about potential pitfalls that had to be mitigated in the design. Encryption types and algorithms were researched to find suitable alternatives for the system. The goal with this part of the study was to find algorithms to encrypt as well as sign data. In addition, cryptographic hash functions was researched due to their role in RSA-OAEP. Due to the system's need to generate and store cryptographic keys, different management strategies was analysed. This resulted in research about the random number generator */dev/random* and key derivation functions to generate cryptographic keys from passwords. TLS/SSL was studied in order to provide secure communication in the system. The final area of the literature study was concerned with storage solutions, but mainly on object storage solutions.

Apart from the literature study, similar solutions available where analysed to get an idea of how the system could be designed. Among these solutions were SpiderOak, Wuala and VeraCrypt, the last of which was used in the proof of concept as a local storage module. Both SpiderOak and Wuala gave inspiration to do research about hybrid encryption which was later used in the design.

7.2 Design and implementation

Before starting to design the system, the requirements of the system was analyzed. The analysis of the requirements helps to fully understand how the system shall operate and what functionality should be available within the system. This is of course of importance for designing the system in a good way. A design was then proposed with the requirements in mind. In the early proposed designs there was some minor flaws that did not reflect the requirements of the system and the design was reworked. During this iterative process, security risks with the proposed designs was discussed and analysed. After the analysis of the proposed system was completed the implementation phase began.

In the implementation phase the proposed design was implemented. This was done in order to see if there would be any problems to realize the design. The implementation intentionally left out parts of the functionality that was deemed too much work for the given time frame. During the implementation, there were no problems to implement the majority of the designed system. The main problem was encrypting and sending large data files since they required a data stream to not consume all the memory of the client.

7.3 Conclusion

When designing a collaborative storage that only stores encrypted files the main focus lies in handling the keys used for the encryption in a secure way. The keys must be stored in a database within the system in a secure fashion while still being available to the clients. It is of importance that the keys can not be extracted from the database by administrators maintaining the system. If the keys used for encryption can be obtained, the confidentiality of the files are greatly compromised. In order for the system to be useful in the described scenario, it requires the above criteria to be fulfilled.

To summarize the findings of this thesis, the research questions in section 1.2 will be used. In order to answer the main assignment of the thesis, the two sub assignments will first be addressed.

2. *How can such a system be designed to securely manage cryptographic keys?*
3. *How can such a system be designed to prevent administrators from accessing confidential information?*

Both of these questions is solved in the proposed design by hybrid encryption. The room key, which is used for symmetric encryption, is stored encrypted with an asymmetric encryption scheme, where the key used is a user's public key. This solves the issue of managing the cryptographic keys. It protects the stored room keys in the database since they are encrypted. It also makes collaboration in the system simple, encrypted room keys can safely be sent to a client together with the needed public key to give a user access to a room. The hybrid encryption approach also prevents administrators from accessing confidential information. Even though they have access to the storage where the encrypted files are stored, they have no possibility to decrypt them since the the room keys are encrypted. Furthermore, it is not possible for the administrators to manipulate the database to gain access to these keys, thereby decrypting the stored files. The additional encryption of fields in the database prevents the administrators from deriving information about files and rooms stored within the system.

The previous questions were answered in order to address the main assignment of the thesis:

How can a system for file storage be designed to achieve full encryption from storage to client?

To answer this question we refer back to chapter 4. The proposed design uses a hybrid encryption scheme, which solved the sub assignments. In the design, the files at rest are encrypted and are also sent to the clients encrypted. In order to decrypt the file the encrypted room keys are retrieved from the database. When a file enters the system, the client once again retrieves the encrypted room key from the database and uses the room key to encrypt the file. This file is then sent to the server and stored in the object storage. Therefore, the proposed design is an answer the the main assignment of this thesis.

7.4 Future work

This thesis only described the usage of TLS/SSL, a replication server and the possibility of running the solution as a distributed system. Therefore, the first step to future work would be to make the transition to TLS/SSL, a distributed system as well as to implement the replication server in order to verify this thesis' claims of their feasibility. Additionally, the management of the *keywarden* has been deemed out of scope for this thesis. In future work this aspect should be considered since the security of the system is reliant on this special user if it is run in *enterprise mode*. Mainly methods to store the *keywardens* credentials will have to be researched since they need to be accessible by the system but protected from manual modifications. Finally, adopters of the system should consider implementing their own AES and RSA algorithms and local encryption module. The reason being that maintenance of these components is made easier than with a third party provider. However, these modules will require extensive testing to verify their functionality and security to not compromise the system. If a third party where to be used, the adaptors would have to do research on the reliability of the third party.

Bibliography

[1] Krithika K Annapoorna Shetty Shravya Shetty K. *A Review on Asymmetric Cryptography – RSA and ElGamal Algorithm*.

[2] Tim Dierks. “The transport layer security (TLS) protocol version 1.2”. In: (2008).

[3] Taher ElGamal. “A public key cryptosystem and a signature scheme based on discrete logarithms”. In: *Advances in cryptology*. Springer. 1984, pp. 10–18.

[4] *ELITS*. 2016. URL: `https://elits.com/what-we-offer/` (visited on 02/15/2016).

[5] Eiichiro Fujisaki et al. “RSA-OAEP is secure under the RSA assumption”. In: *Journal of Cryptology* 17.2 (2004), pp. 81–104.

[6] P Gallagher and C Kerry. *Fips pub 186-4: Digital signature standard, dss (2013)*.

[7] Rosario Gennaro. “Randomness in cryptography”. In: *IEEE security & privacy* 2 (2006), pp. 64–67.

[8] Ravindra Kumar Gupta and Parvinder Singh. “A new way to design and implementation of hybrid crypto system for security of the information in public network”. In: *International Journal of Emerging Technology and Advanced Engineering* 3.8 (2013), pp. 108–115.

[9] Hakan Hacigümüş, Bala Iyer, and Sharad Mehrotra. “Data and Applications Security XVII: Status and Prospects”. In: ed. by Sabrina Capitani di Vimercati, Indrakshi Ray, and Indrajit Ray. Boston, MA: Springer US, 2004. Chap. Ensuring the Integrity of Encrypted Databases in the Database-as-a-Service Model, pp. 61–74. ISBN: 978-1-4020-8070-8. DOI: `10.1007/1-4020-8070-0_5`. URL: `http://dx.doi.org/10.1007/1-4020-8070-0_5`.

[10] Ross Anderson1 Eli Biham2 Lars Knudsen. “Serpent: A Proposal for the Advanced Encryption Standard”. In: *First Advanced Encryption Standard (AES) Conference, Ventura, CA*. 1998.

[11] T. Mager, E. Biersack, and P. Michiardi. “A measurement study of the Wuala on-line storage service”. In: *Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on*. 2012, pp. 237–248. DOI: `10.1109/P2P.2012.6335804`.

[12] Teddy Mantoro and Andri Zakariya. “Securing e-mail communication using hybrid cryptosystem on android-based mobile devices”. In: *TELKOMNIKA (Telecommunication Computing Electronics and Control)* 10.4 (2012), pp. 807–814.

[13] Mike Mesnier, Gregory R Ganger, and Erik Riedel. "Object-based storage". In: *Communications Magazine, IEEE* 41.8 (2003), pp. 84–90.

[14] James Nechvatal et al. *Report on the development of the Advanced Encryption Standard (AES)*. Tech. rep. DTIC Document, 2000.

[15] *OpenSSL Frequently Asked Questions*. 2016. URL: `https://www.openssl.org/docs/faq.html` (visited on 03/11/2016).

[16] *OpenUP*. 2016. URL: `http://epf.eclipse.org/wikis/openup/` (visited on 09/27/2016).

[17] Shari Lawrence Pfleeger. *Software Engineering: Theory and Practice*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1998. ISBN: 0-13-624842-X.

[18] *random(4) - Linux man page*. 2016. URL: `http://linux.die.net/man/4/random` (visited on 03/11/2016).

[19] Robert Richardson. "2010/2011 CSI Computer Crime and Security Survey". In: CSI Computer Crime and Security Survey (2011).

[20] Ronald L Rivest, Adi Shamir, and Len Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[21] Bruce Schneier. "Attack trees". In: *Dr. Dobb's journal* 24.12 (1999), pp. 21–29.

[22] Bruce Schneier et al. *The Twofish Encryption Algorithm: A 128-bit Block Cipher*. New York, NY, USA: John Wiley & Sons, Inc., 1999. ISBN: 0-471-35381-7.

[23] Claus Peter Schnorr and Markus Jakobsson. "Security of signed ElGamal encryption". In: *Advances in Cryptology—ASIACRYPT 2000*. Springer, 2000, pp. 73–89.

[24] SpiderOak. *SpiderOak Official Website*. 2016. URL: `https://spideroak.com/` (visited on 04/08/2016).

[25] William Stallings. *Cryptography and network security: principles and practices*. Pearson Education India, 2006.

[26] NIST-FIPS Standard. "Announcing the advanced encryption standard (AES)". In: *Federal Information Processing Standards Publication* 197 (2001), pp. 1–51.

[27] Wade Trappe et al. "Introduction to cryptography with coding theory". In: *The Mathematical Intelligencer* 29.3 (2007), pp. 66–69.

[28] Meltem Sönmez Turan et al. "Recommendation for password-based key derivation". In: *NIST special publication* 800 (2010), p. 132.

[29] *Veracrypt*. 2016. URL: `https://veracrypt.codeplex.com/` (visited on 03/08/2016).

[30] *Veracrypt Documentation*. 2016. URL: `https://veracrypt.codeplex.com/documentation` (visited on 03/08/2016).

[31] *What is Scrum? An Agile Framework for Completing Complex Projects - Scrum Alliance*.