Client Side Encryption for Soonr Cloud Service and Secure Key Distribution

Andrius Andrijauskas

A dissertation submitted to the Technical University of Denmark in partial fulfilment of the requirements for the degree of Master of Science in Computer Science and Engineering

July 14, 2014

Declaration

I declare that the work described in this dissertation is my work, except locations where stated otherwise. This dissertation has not been submitted as an exercise for a degree at this or any other university.

Signed: _____

Andrius Andrijauskas

Date: July 14, 2014

Abstract

Data privacy depends on the honesty of the source that has to be trusted during storage and/or exchange of said data. Users choose ISP, servers for storage and solutions to store data. The choices are based on a combination of functionality and credibility. An alternative is to decrease the need to trust an outside source for data management and give the user more control over it.

This thesis presents an analysis of currently existing state of the art methods and tools for securing data, provides considerations for the choices made, details the implementation and analyses the results. Current problems are analyzed in addition to the tools and appropriate considerations for solutions are given.

The project is performed in collaboration with “Soonr” - an external company, which is specializes in dealing with confidential data around the world. The company was founded in 2005 and deals with data from more than 150,000 companies worldwide. The project includes changes to part of their code to support additional features. The changes are analyzed to display how well they follow the outlined requirements.

The information regarding the company’s data is disclosed to the extent needed to carry out the analysis of the implementation.

A key management solution is designed, implemented and analyzed to fulfil its’ part of the requirements.

Table of contents

1 Introduction

Storage and sharing of stored data is becoming more important as electronic devices become cheaper and more accessible to people. Every electronic device generates data and as the world is now, this data is stored and shared in some way. This includes storing it on local drives and using network technology, which is now covering a large part of the world. This is part of what is known as the internet of things.

It does not necessarily require human interaction to generate data, so the speed at which data is created is far higher than what people generate manually. This includes computers, cameras, phones and other everyday appliances everyone seems to have and less common devices, such as, washing machines and refrigerators with integrated internet connection. [1]The data generated relates to the status of the device, updates to the system or even command lists waiting to be executed. As mentioned, every piece of data has to be stored in some location, so the choices come down to local storage or sending the data to online locations. In this situation the local storage becomes a temporary medium, which eventually leads to sending data to online locations.

With all the tools connected to the network, several problems appear: size of the data being generated and the size of the devices, complexity of the network needed to be able to use the devices, security of the data moving in every direction. While incorrectly chosen size of the device, the amount of data needed to be transferred in a short amount of time and complexity of the network can result in non-functional equipment, which can be troubleshooted after unsuccessful installation, failure in security can result in deliberate damage to the equipment or the user at any point in time, when unauthorized people gain access to the devices. This includes tampering with the functionality and stealing data from the system.

The security is considered enough to provide practical security, while maintaining overall functionality of the network intact. This means that while maintaining an acceptable level of security, which is different for individual scenarios, reliability of the network is another important issue. Various solutions have been developed for data sharing over the network, which allow the data to remain reasonably secure while easily reachable to authorized parties. The reachability is covered by faster network connections with wider coverage, data compression and cloud storage. The data is considered to only be reasonably

secure. In order to achieve true security every piece of technology would have to be user developed - even the tools. As soon as a third party has a part in developing anything related to the technology, there is a possibility that they included a deliberate flaw, which allows access to extra data or cripples a feature to later benefit from that. So, data can only be reasonably secure.

Data transfer speeds over global networks are getting higher with each passing year thanks to new technologies, such as, fiber optics by google and a lot of the world has not even caught up with the current technologies[2], so it should not reach a dead end anytime soon. In addition, data compression decreases the amount of data actually transferred. This leads to having more freedom when considering security model for data being transferred.

Cloud storage in combination with data compression is considered as single element, since cloud storage solution has to take care of the data transfer either way – that is data compression and security.

The Cloud is a popular choice for many users. It provides storage solutions for any device connecting to the web and can also provide computational power to the user. In case of computing, the user is given the possibility to unload processor or graphics card intensive workload to remote sources. The advantages of such an approach are increased computational power, which can be reached from any device or devices at once. By using these solutions, the user shows that he/she trusts the location/provider, where the work is done. Cloud computing also uses cloud storage as part of the entire package. This does not mean that the user is forced to use everything or nothing. The cloud storage can be used separately just as easily.

When using the cloud storage, the users give data to a trusted party, who makes the data available to the users wherever they are and often also allows users to share data with other authorized users. These users are chosen by the original uploader. Such sharing of data happens through a web interface or locally installed applications. The next step gives the data to cloud storage provider, who then pushes it to their own storage locations. This is illustrated in Figure 1:

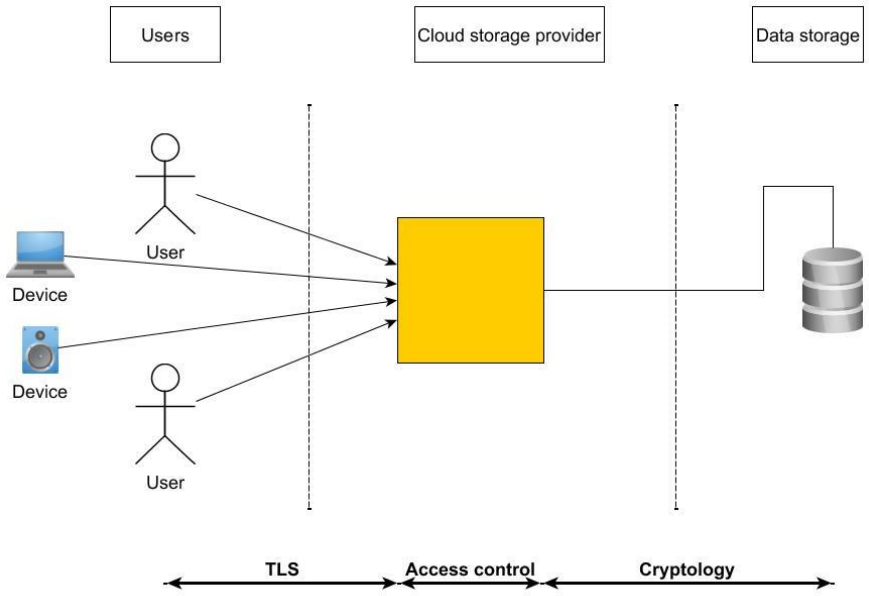

Figure 1. Cloud storage layout

The users or their devices share data with the cloud storage provider, using security technologies, such as, TLS for data stream encryption. Then access control provides a way to manage the data, which at some later point can be encrypted with various tools and transferred to remote storage locations . The retrieval of data works in a similar fashion, only in reverse. The tools used in all the steps are set and known to the cloud storage provider. This includes the keys and the encryption/transfer methods.

The data in the cloud is managed by various parties, which make sure the data does not get destroyed, is always available to the users and provides various other utilities to manage the data. This is acceptable if the cloud storage provider is trusted. When the data reaches them, the sender assumes that the data will be handled as agreed. Cloud storage providers are generally storing the data at facilities, which are not bound by country or even continent. These facilities can be part of the company or the data storage might be outsourced to unrelated companies. At each step more people are involved with the data and cases, where someone uses the data for personal gain, prove that the user cannot completely trust others with their data - it can be stolen by employees [3] or outsiders, who gain access to employees accounts [4] [5].

The location, where the data is stored, the country, which the provider is based at have to follow the laws of the country. The relevant rules include the need to accept requests for data from government

officials [6], where the company has to provide data even from overseas locations [7]. The user has no control over the actions and it has no difference, where the data is stored physically.

Cloud storage providers also have to trust their own sources. It has been revealed to the public that ISP's, data storage solutions with their servers do not have enough control over the processes they handle to prevent governmental organizations, such as NSA, to can gain access to servers [8], install backdoors in routers [9] and computers [10] and cripple cryptographic algorithms [11].

A concern of entrusting the data to cloud is that actual protection methods connecting the user to the data are mainly software based. This provides various different properties, such as, permissions to view, edit, the need to protect information in case of a failure in the system. The user can store this data offline (flash drives, hard drives, etc.) and share it with others when the need appears by using cloud solution. The information is travelling over a public environment and is being stored in locations, which are managed by third parties. Cloud services provide methods to protect data going from the user to the servers and then the data is fully accessible to employees unrelated to the user when it reaches the servers. This is because the data is protected with the tools and keys, which are known to the company when stored. So, the information distribution can be controlled by settings for sharing, storage location and general privacy, but this is as effective as the solution decides to allow it.

After the overview this emphasizes the following issues/problems:

1. The data is accessible by several third parties without the permission or knowledge of the user.
2. The user cannot actually approve who can gain access, only request storage providers to follow user decisions.
3. The safety of the data is dependent on the cloud storage service the user chooses.
4. The user does not have any verification of how well the security solutions function.

One of these cloud service providers is a company by the name of "Soonr". This company provides cloud storage service to various customers and the data is stored around the world, which is accessible by the employees around the world. The need for moving the trusted link closer to the user and increasing the power users have over their own data during the entire storage resulted in the Secure Deal Room project. It stands for keeping unauthorized parties out of the communication between users, while still providing data transfer services as before.

Secure Deal Room appears because of the need to:

1. Limit the access to data by third parties. Provide the users with data access control, where the user is the only one in control, so no access to server data with request of third party organizations, such as governments with warrants or unauthorized access by the employees, is gained.
2. Limit the responsibilities of the company, when handling the interactions between the users. The user personally decides and invites other participants. This includes the transfer methods and sender/recipient verification.

In addition to the direct requirements for Secure Deal Room, additional requirements are:

1. Provide data integrity control, so the users' access control cannot be physically tampered with.
2. Provide simple controls for the user.
3. The tools offered to the user must be trusted, so they have to be transparent for everyone to be able to verify the functionality.

The sub-problems not considered are:

1. Data storage and transfer methods. The company, collaborating with the project provides data storage and transfer methods.
2. Data and access control protection from direct monitoring via key logging or looking over the shoulder.
3. The stability of the tools used, such as crypto algorithms and their implementations, programming languages.

The following parts of the report will look into currently existing state of the art technologies, analyze their advantages and shortcomings. In addition to the analysis, Soonr offering will be analyzed in separately and in more detail. After the analysis, the secure deal room problems and sub-problems will be identified. Following that, the design of the solution will be defined and the implementation will be described. Finnaly, the results are evaluated and the conclusion is given.

2 State of the art

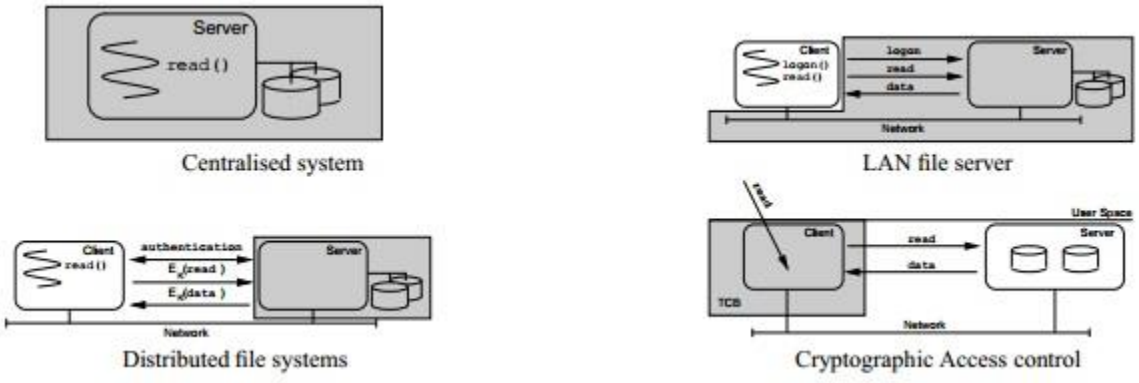

Figure 2. Trusted Computing Base in different types of file systems [12]

Data flow in various systems is divided into four categories as displayed in Figure 2:

- Centralized system has all the components in a trusted computing base, which is on the local operating system. There is no transferring of data involved and the user can reach the physical storage location of the data.
- LAN file server has both the file server and the intermediate network and the user must login. This covers web based interactions, where the user does not store any data on the local machine.
- Distributed file system has an insecure network, but the server is considered secure. This covers the local client based interactions, where the user stores data on the local device and the server side takes care of the rest.
- Cryptographic access control model only trusts the client. Covers the data being locked by the user and the server waits for the user to process the data before it is sent to the server.

During the project, state of the art solutions are a base point against which the need for improvement is analyzed. The analysis is carried out on cloud storage, encryption and access control solutions.

2.1 Cloud storage

The cloud storage solutions are relatively similar among each other and the features will be analyzed further. They fall into the distributed file system category, since the service provider manages transfer and protection.

DropBox has cloud-based services for data storage and management, clients for data access and storage on desktop and mobile devices and web interfaces for data and service management. Desktop client supports synchronization, sharing, personal storage and revisions of files, so deleted files may be recovered from any linked computer. Dropbox uses Amazons' storage system to store the files, though it may switch to a different storage provider at any time. SSL is used for synchronization and data transfer using AES-256 encryption with encryption keys, which belong to Dropbox, and not the user. This means that the user does not have any control over the encryption mechanisms. The entire data transfer generally follows the flow from Figure 1, where user data is transferred to cloud storage after being encrypted with various tools to protect it during transfer. Then the data is decrypted and moved to Amazons' storage locations. In 2011, it has been reported, that Dropbox system checks if a file has been uploaded before by any other user, and links to the existing copy. In addition, a single AES-256 key for every file on the system is used, so Dropbox can look at private files stored on the system, with the threat that any intruder, who gets the key, could decrypt any file if they had access to Dropbox's backend storage infrastructure.[13]In later events DropBox has been hacked, where private data has been leaked to the web.[14] The most popular solutions are usually targeted by a larger number of attackers, so a possibility of a successful attack is always exists. DropBox gives priority to functionality and convenience in data management, which means that security is covered to the extent, that attackers would not gain access to the "user-dropbox" link.

Competing solutions to dropbox employ similar solutions. AES encryption is used throughout and the main difference exists in the amount of storage offered to the user. These solutions are box, google drive, one drive, icloud.

The online solutions, like "SendThisFile" or "DropSend", provide the "entire package": encrypting data with offered tools, sending it over the network to the other user. The desktop client solutions, similar to web interface solutions, offer encryption, transfer over the network and sharing with other side. These solutions store data on remote servers, where this data is protected with locks, which can be unlocked by the company and not necessarily by the original owner of the data. These solutions offer a trust based security. The user trusts their honesty, that the data will not be misused or passed to another party. The information is protected with officially approved methods(encryption, transfer protocols), so this is currently accepted. The methods offered to the users are fully controlled by the service providers – the source code is hidden from the public, the external audits proving that the solution is correct also depend on external forces. Even if the solution is open source, not every user can confirm the

correctness of the code, so they have to believe that storing, encryption is done behind the scenes the way it should be. The user does not take care of the tools needed for the entire process. Usually, a password is used as first and only line of defense when using the tools.

2.2 Encrypted data storage

Encrypted data storage allows the user to store information in the same location as the original source and the information is inaccessible to others without the users' consent, so it belongs to the centralized system category. The encryption can be carried out on separate files or entire disks. Further choices define whether data on the current file system is encrypted or is an encrypted virtual drive used. The choices for encrypted data storage are:

- TrueCrypt encryption suite, which is well known. It offers the possibility to lock the data from others and the user is given the choice to lock small parts of data or entire disks. It can create a virtual encrypted disk within a file or encrypt a partition or the entire storage device. This solution is open to auditing, which has been verified to have integrity [15]. TrueCrypt supports parallelized encryption for multi-core systems to reduce the performance hit of encryption and decryption. The system is not used for data sharing between various users automatically. It is possible encrypt data and manually put it on external media, but this is not part of the TrueCrypt system. The TrueCrypt solution supports key generation for encryption methods. The keys are stored in the RAM, when the methods are used to ensure that no unnecessary data remains after the computer is turned off. Still, the data remains in memory for several seconds, so successful attacks on this feature exists [16]. This is available only when the attacker has physical access to the machine. As long as the user does not provide physical access to the machine or installs malware on the system, TrueCrypt guarantees its security measures. Unfortunately, TrueCrypt has been discontinued [17]and it is recommended to change to an alternative solution. Alternatives include DiskCryptor, which offers similar features and is still supported.
- Another alternative to this is Windows Bitlocker, which offers a similar solution to TrueCrypt. Differences between Truecrypt and Bitlocker are that Microsoft solution is proprietary and has to be trusted by the user to function correctly. This solution is supported by the company and is integrated in Windows operating system, so the user is already equipped with it. It offers the possibility to encrypt full disks only, rather than giving the possibility to encrypt individual files.

The relevant difference between open source vs. proprietary is who needs to be trusted. In first case the online community determines the quality of software and in second case the users have to trust the word of the company that created the tool. After encrypting portable media, it is then physically transferred to another location. Transfer of data and key is a manual process – the participants must meet to exchange data and keys or use the network to pass data. If keys are passed over the network as is, it defeats the purpose of encryption in the first place, since encryption security depends on the attackers not knowing the key.

2.3 Cryptographic access control

Observing the popular and easily accessible management choices, the clear concerns are related to access control. The user does not control what happens to data, once it is sent to the servers or even while being sent. The data may be theoretically viewed by anyone, who is approved by the management solution provider. Thus the concerns are the ability for the user to limit the access to personal data. Secondary concern is the ability to let users decide, who can read the data without having to trust an outside party to do the work.

Another concern is that at the same time access control can be completely removed from the users' side, where the service provider blocks data access to users' approved participants, the user included. In addition to that, the user lacks reassurance that the chosen participants are really who they claim to be. The service provider can always tamper with this information in compliance with government request [18]. The alternative choices in this scenario are locally generated and maintained methods to preapprove participants. The concern is how reliable the preapproving actually is.

Since access control data should be closer to the user, i.e., the user machine, the machine itself becomes more prone to being attacked. Any program on the computer can potentially look at the access control keys without user consent, so instead of just storing access data in plaintext, access control data has to have a layer of protection. This includes physical access to data and remote access without user consent. The data has to be locked from prying eyes when the user is not present and other applications on the users' device should not have access to access control data without user giving permission to do so. This comes down to a locking mechanism with a key, which is kept of the device when not needed and only revealed to chosen mechanisms when needed.

2.4 Symmetric encryption choice

AES encryption has several modes, each with different advantages and disadvantages to meet different needs by the users [19].

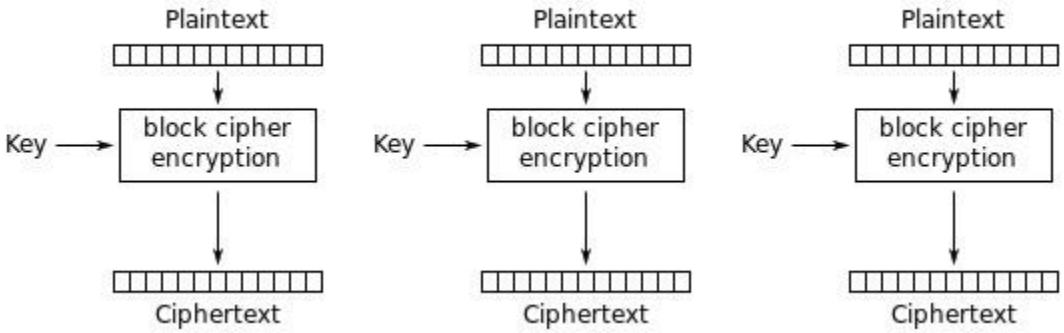

Table 1. Electronic codebook mode - ECB

Table 1 displays a message, which is divided into blocks, and each block is encrypted separately. Same blocks are encrypted to the same cyphertext, so this disadvantage makes this mode unusable in real life situations.

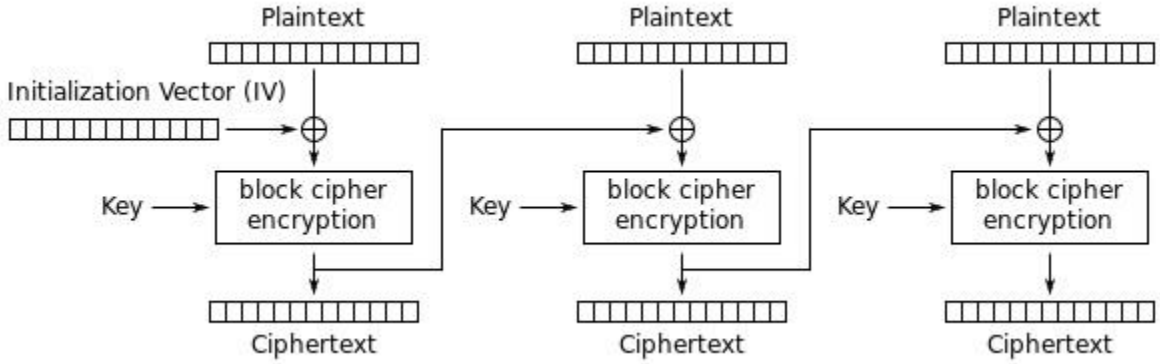

Table 2. Cipher block chaining mode - CBC

Table 2 shows each block of plaintext XOR'ed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block. The resulting encrypted message appears random even if each palintext data block is the same.

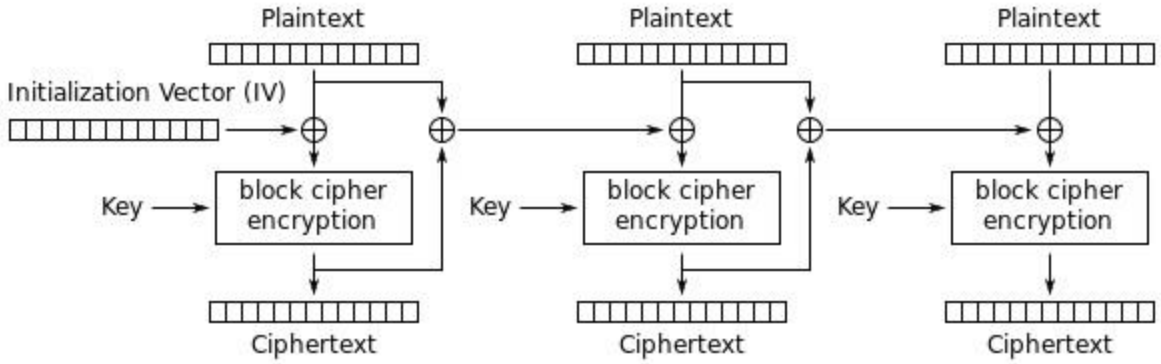

Table 3. Propagating cipher block chaining - PCBC

The propagating cipher-block chaining or plaintext cipher-block chaining mode displayed in Table 3 causes small changes in the ciphertext to propagate indefinitely when encrypting or decrypting. As with the previous mode, the resulting encrypted message appears random.

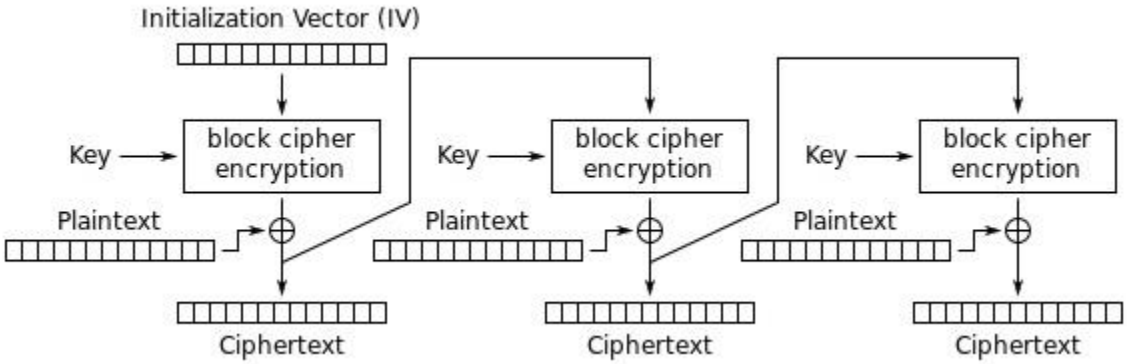

Table 4. Cipher feedback mode encryption - CFB

The cipher feedback (CFB) mode makes a block cipher into a stream cipher. CFB decryption is almost identical to CBC encryption performed in reverse.

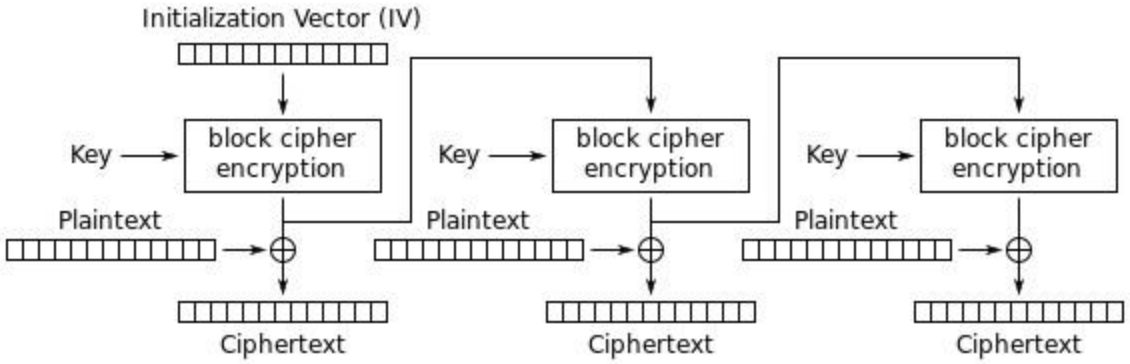

Table 5. Output Feedback mode encryption - OFB

The output feedback (OFB) mode makes a block cipher into a synchronous stream cipher. It generates keystream blocks, which are then XORed with the plaintext blocks to get the ciphertext.

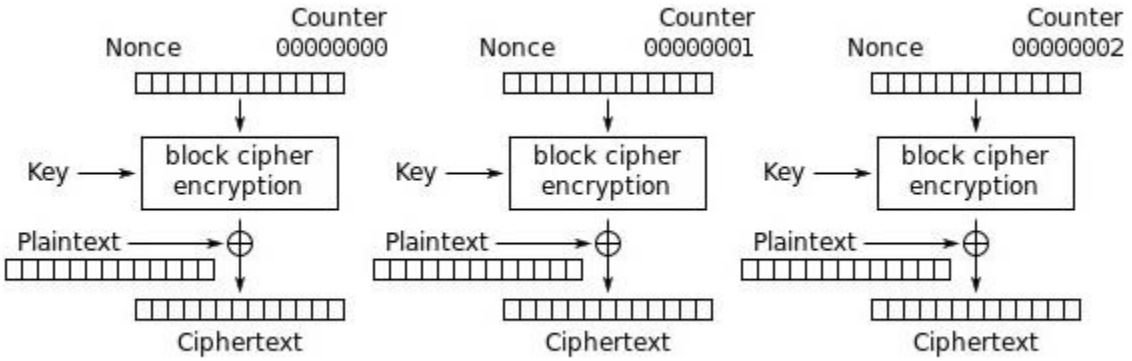

Table 6. Counter mode encryption - CTR

Like OFB, counter mode turns a block cipher into a stream cipher. It generates the next keystream block by encrypting successive values of a "counter".

The ECB mode is the only choice from the AES group that does not provide consistent secrecy when encrypting.

Since the encryption modes are divided among block cyphers and stream cyphers, the stream cyphers are removed from the selection. This is decided, because stream cyphers are generally used, when a continuous flow of data is appearing, such as, chatting with another user. This narrows the choices between two modes. CBC mode is more common than PCBC and provides an acceptable level of security, while PCBC increases complexity of the implementation and the load on the device. So, CBC mode is considered to be the choice in AES encryption.

2.5 Asymmetric key encryption considerations

A public key will be known to a large and, in practice, unknown set of users. All events requiring revocation or replacement of a public key can take a long time to take full effect with all who must be informed (i.e. all those users who possess that key). For this reason, systems that must react to events in real time (e.g., safety-critical systems or national security systems) should not use public-key encryption without taking great care. There are four issues of interest:

- Recovery from a leaked key: A user who the key belongs to decides to revoke that certain key. This is carried out because the private key gets compromised. Such a compromise has two implications. First, messages, which are already encrypted with the matching public key or will be encrypted later, can no longer be assumed to be secret. Second, signatures made with the no longer trusted to be actually private key after being compromised can no longer be assumed to be authentic without additional information (i.e. who, where, when, etc.) regarding the events leading up to the digital signature. They will not always be available, so these digital signatures will no longer be trusted.
- Spreading the revocation: Anyone having the public key in question must be notified about the compromise as soon as possible. There are two ways to spread this notification: it is either pushed by the central location (server) or collected by the users (checking with the original owner before every use). Pushing the information is convenient, because the message is sent to all participants. However, there is no confirmation that all public key holders will actually be informed about the revocation of the key. This can happen because of "denial of service" attack on the central service. The alternative is pulling the current state of the key. If the user cannot verify the current state of the key, the key will not be used until further notice.
- Distribution of a new key: After a key has been revoked, a new key must be distributed to the users. In case the key is pushed from a central location, the user obtains the key, when the service is online. Otherwise, the user has to locate the key owner and request a status update.
- Privilege of key revocation: Key revocation lies with the service, which stores the public key or with the user, in case no service is used. In the first case, the service is issuing a revocation command, so it is possible to call it without the user giving specific order. This is possible if the service is hacked. In the second scenario, only the original user has the right to revoke the key, since the participants would connect to the user for key status update.

2.6 Soonr

A separate case of the solutions, which offer desktop clients and web interfaces, is Soonr. Since the project is carried out in collaboration with the company developing Soonr cloud storage tool, a clearer definition of the methods used is available. The solution is also part of the state of the art, so the description is relevant.

The soonr data backup system is divided among the client and the server segments. The server solution is not part of the project, so it will only be covered as much as it is related to the client side. The client is developed in c++ language for Microsoft Windows platform. The existing code on the client side will have minimal adjustments, while new features will be added as separate files/classes. This will include GUI changes and security related elements from the project itself.

2.6.1 Description

The server side of the system contains the database for files and the user login information. The server is responsible for every interaction with the client side.

The client side on the user device contains a syncing mechanism and an environment for storing data to be backed up to the servers. The user is authenticated by sending necessary data encrypted with a predetermined key.

The Soonr clients' syncing mechanism consists of transfer module and sync core. The transfer module prepares the data, which needs to be sent to the server. This means moving the data to the cache location, splitting it into 64 kilobyte blocks necessary for encryption, encrypting data and hashing each block with an MD5 function for integrity. After this, each block is sent to the server. The transmission can be interrupted, multiple files can be transferred in each direction and the data is compressed along the way. The data is received in 64 kilobyte blocks as well. The blocks are combined and managed in the cache folder, where the data received is decrypted and assigned a random name. After the file is fully downloaded and verified, the data is transferred to the project folder with the original name.

The sync core is constantly monitoring the data between the client and the server. When a change is detected, the sync core launches the transfer module to receive or transfer a file. When a file has been received, the **sync core** receives a hash of the file. This hash is compared to the result from hashing the

received file and if the results match, the file is given the original name and instruction to transfer outside of the cache location.

The sync core has been designed to only accept commands from the server, meaning that as long as the client is online, it can be manipulated by commands from the server. The client cannot launch the sync core if there is a need to interact, it can only be the recipient of the action.

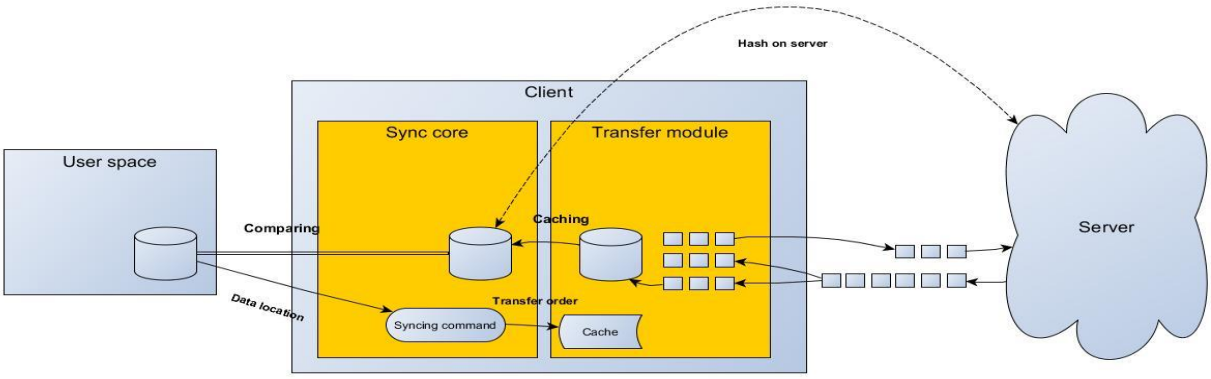

Figure 3. Soonr solution architecture

The graph displays the sync core comparing the data between the user space and data known from the server. The sync core gives the command to sync and points to the data needed to sync. The file is placed in the cache and after being processed, is sent block by block to the server.

The server side obtains files in 64 kilobyte blocks, decompresses them and decrypts. The total data received and sent must have the same size. If not, the hashing verification is not even carried out, assuming that the file must be transferred incorrectly.

The client side obtains multiple files during a single transmission, indicating multiplexing. After the file has been reassembled, it is moved from the cache to the final destination.

3 Secure Deal Room

The problems related to Soonr are defined from the general problems in the introduction:

- The data is accessible by many third parties without the permission of the original user. Soonr system has multiple employees who deal with user data directly. This includes those who take care of data storing, customer complaints, network upgrades. These employees have signed a non-disclosure contract and do not have a criminal record, which would otherwise cause a concern for data secrecy. The people, who can access data, extend beyond the company and this data can be requested by government officials. In order for the company not to lose customers and get sued for data mismanagement, it follows the rules. The data going from the user travelling to the Soonr servers is encrypted, so unauthorized parties, which may be eavesdropping are blocked. Soonr is generally a secure location to store data as long as the user trusts the company and does not have issues with the any governments. This displays an issue for Soonr, who cannot provide a secure storage location regardless of the situation of the user.
- The user is not informed about who is accessing the data. Data stored on the servers is moved to different locations in order to provide high speeds and stable connections to the server. During these actions, data is analyzed to determine its size, type, frequency of access and similar information. The data itself can also be accessed to prepare it for streaming mode. During all the actions, the user is not bothered with status updates, since no information is leaked to unauthorized parties. This changes, when the government tries to gain access to user data. The data is moved to any part of the world if it is considered to be beneficial and the law applies from any of those countries. The governments can demand that the user would not be informed about the request or the user is notified after the request has been completed. In this situation, Soonr does not provide tools to constantly keep the user informed about the state of his/her data and move to different locations accordingly.
- The safety of the data is dependent on the cloud storage service the user chooses. Soonr implements technologies necessary to ensure that user data does not leak at any step, where Soonr is involved – transfer and storage. Since no know technology provides all the possible features, the company makes a decision which technologies to use. The company keeps the tools up to date, but the tools might have a bug, which might pose a security risk. The company

cannot shift to a completely different set of tools because of that and cannot disable the system until the issue is taken care of.

- The user does not have any verification of how well the security system functions. Soonrs' tools are proprietary, which are audited by external companies regularly. This shows that the code meets the requirements needed for correct operation. At the same time the user has to trust the auditing company, rather than personally chosen sources.

After analyzing general problems from the companies' perspective, the following specific problems become visible:

1. The level of data security the company can offer the user depends on the situation the user is in.
2. The user does not impact the storage location and does not have the information on the status of personal data.
3. Soonr does not have a layer of protection, which would not be affected by the remaining protection mechanisms.
4. The user has to trust security verification methods Soonr chooses with no alternative.

3.1 Solutions

The possible ways to address the problems identified come down to having an addition to data management. The choices are:

1. Introducing an extra tool to the Soonr system, which addresses the problems.
2. Changing existing Soonr tools to address the problems.

In both scenarios the same results must be achieved. In the case a new tool is introduced and integrated with existing tools, the user gains missing features with no changes to existing solutions. This is beneficial to Soonr, since the existing implementations are not rewritten. In case of the second choice, Soonr has to change a part of the code, which would consume extra time as compared to the first choice. This leads to accepting the first choice.

The extra tool should use encryption/decryption and keys for carrying out cryptographic methods. The location of the keys has to be where Soonr cannot reach them. Otherwise, there is no difference compared to already existing methods. This points to the user storing the keys in a remote location. The encryption should be carried out in the remote location as well, since otherwise the keys would become known to a third party. The encryption method can be implemented by Soonr or another party, but the

user still has to trust one of those sources and by using a separate tool from Soonr system, the user has to manually take care of encryption and decryption. The Soonr system, on the other hand, provides the encryption implicitly, which does not force the user to do extra work. Soonr is a trusted source, so the implicit encryption method is preferred.

After the data is encrypted, it is considered that it cannot be accessed by the third parties, so Soonr transfer mechanisms can be used.

The keys used for encryption must remain for as long as the files need to be decrypted. The user has several methods for storing the keys:

1. Using a removable media to disconnect the keys from the network completely.
2. Using the same device where encryption is carried out. The keys are locked by using encryption on them.

The first choice requires the user to import the keys on each use and after everything is finished, the key should be removed. The second choice does not move the keys around. The need for protecting the keys comes from cases, where a third party gains access to the device itself. The first choice has the risk that the user would keep the keys just on the device without any protection, while the second choice poses no difference. Therefore the second choice is preferred.

The key used for encrypting data is given to other users, since the data is intended to be shared. Since the key should not be connected to Soonr system, the sharing mechanism should not depend on it. This leads to using a transfer mechanism, which is not limited to what method is used to send.

The initial solution leads to:

1. Keeping the keys and encryption methods on user device.
2. Protecting the keys from unauthorized access.
3. Using the Soonr system to transfer files.
4. Transferring the keys without limiting the transfer method.

3.2 Key management

The keys used in cryptography are generated by taking a random number for symmetric keys or performing calculations to determine the key combination for the asymmetric keys. It is generally accepted that asymmetric encryption is several orders of magnitude slower than symmetric, so

symmetric encryption should be generally used for data encryption and asymmetric encryption should be used for key transport and digital signatures.

3.2.1 Key distribution

The choices for what type keys have to be shared are linked to the cryptographic methods, which are symmetric and asymmetric. Absolute choice for key being sent is in favor of the symmetric key, since it uses “special cryptographic methods” versus asymmetric key transportation method, where the security is based on difficulty to calculate a large prime number, which does not have a guarantee that this will remain this way in the future. The “special cryptographic methods” are a combination of symmetric encryption of small pieces of data with the key, shuffling several pieces of data before or after encrypting them, encrypting the result even further and using an extra piece of information for initial shuffle of the data. Asymmetric key, on the other hand encrypts the data by using the message as a single variable in a mathematical function. Real life situation has to take into account the storage of the key issue, where the symmetric key does not have any hiding advantage/factor. The asymmetric key, on the other hand has a high hiding factor, since it is made from two separate key parts. This hides the functional structure of the key from the other participant as well as from hijacking of one participating side. The communication is voided, but the damage is only to the hijacked side, not both parties as in the symmetric key case.

The symmetric key itself has the issue of safe distribution. This usually would be done over a trusted third party, which is accepted by both participants. Since the project is handled in a way that the third party should not gain easy access to the conversation just by using the information at hand, the symmetric key transfer by using a trusted party is immediately ruled out for the key distribution segment. The asymmetric key can be easily stored on the trusted server, which is only trusted to the extent of forwarding data to the correct destination. The trusted party is only responsible of giving the correct keys to the participants. This key distribution can be limited to only one time and afterwards the participants will be able to avoid giving any key data to the trusted server.

The keys distributed are symmetric since the symmetric key offers a higher level of cryptographic complexity then the asymmetric type. The key is a one-time transfer only element, so the participants are protected from the adversary analyzing the key data flow and getting the key. The asymmetric key is slower than the symmetric counterpart in encrypting and decrypting data, so it is at a disadvantage when it comes to transferring large amounts of data. This becomes more troublesome when bigger keys are used. Although the case, where the longest asymmetric and symmetric keys are used, does not have

this problem. The other element is the sender authentication. It is partially covered by the asymmetric key, but the initial cases are completely overlooked. The users need to exchange the initial asymmetric keys in order to perform authentication of each other. The standard situation points to the user being forced to trust external party for authentication matters. At this point the whole disappearance from the trusted party becomes partially invalidated, unless we consider that the first usage is acceptable by everyone. The external party becomes any solution that transfers data, not just the one, which uses the one, which will be used to encrypt the data. In this case, choices, such as, gmail, dropbox, facebook or Soonr, become equally acceptable, since they must be trusted by the user. The exception to the situation is exchanging the keys, when meeting face to face.

The last part to consider is the work environment itself.

- The user can have a virtual localized segment in the "trusted" server, where only the key distribution is done outside of that environment. This way the server cannot monitor the exchange of the keys.
- All processes are completed in the trusted servers' environment. This way the server monitors every step the user takes, which includes the exchange of the key. Communication data is gathered in hopes that the session key gets unlocked when the asymmetric key gets compromised or some other flaw becomes visible.
- Everything is done outside of the servers' environment. This choice does not rely on the server at all, so any data transfer is a responsibility left for the user.

3.2.2 The abstract solution

The choice for user authentication will be outside companies' authentication mechanism. This includes exchanging physical media, transferring over email and other similar methods. After this the user will have to choose between trying to get a new key for another communication or use the already acquired key. The public key in itself will be the authentication verification. The user still has the option to go through the server mechanism, but this is on the same level of trustworthiness as using an email.

The symmetric keys will be encrypted with asymmetric keys and sent to other users. The differences in various asymmetric encryption protocols are considered. The protocol should be trusted by having a mathematical proof of security or by remaining unbroken since its introduction(that being decades). In addition to that, speed, key space and widespread usage are considered. The exact asymmetric key must

be hidden throughout the execution and the copy used in the encryption must be securely destroyed after use.

The data itself will be fully visible to the users, while the server either has no data at all or stores all the data fully encrypted. Generally the server will act as a middle point, where the data is stored, since constant connection between the users is unreliable. The difference between regular secure connections, as defined in state of the art chapter, is that the user will have to decrypt data, which is also encrypted with server keys. The need to use server keys is justified by the fact that the system already uses them and trying to avoid them would increase the amount of changes done to the system without giving extra protection. Therefore, the new "sub-system" will keep using the same solutions.

The big dilemma is where the data should be stored. This will impact how the data will have to be destroyed and treated before destruction. If the users are only responsible for the key, the server will be stuck with wiping the encrypted material(trust issue). If it is stored in some volatile memory, the issue of storage limit appears. Another problem is how to decide when data is no longer used and must be destroyed(What is the entire metadata of this). What are the issues during the notification phase – possibility to delay the deletion of data or even keeping it alive permanently, since the kill command was lost. It could be considered to be checked for keep alive state by regular connection to sever, but this is considered as monitoring the participants' activities, the same as having activity log. But this is only a moral dilemma, since the users do not have a legal protection from having a log on them. Only the data itself must be protected from being discovered. So it all comes down to having a provable method of key distribution for data encryption and afterwards the data itself can be entrusted to the most convenient source. If this is proved to be a breach in security that means the encryption method used and the key generation itself is not optimal, since current generation choices already provide satisfactory results for the situation of untrusted environment.

The data storage type will be spilt into data itself and the metadata. This way the user has the ability to see what is available without being forced to download the entire data beforehand. Otherwise this would rule out the use of server storage and introduce a partial risk for full data destruction. The use of communication between users will come down to messaging(simple structure) and other media found in the solution, even though some media would be considered to be less trusted for secret talks, even though it is user preconception.

3.2.3 Initial solutions to problems

The problem of having third party managing access to users' data is related to who has the tools to control the access. In non-private case, where everything is sent to server by using their protection(algorithms and keys), the access control is given to trusted third party. In this case, even if the user decides to remove data, there is no guarantee that third party will comply.[20] This leads to the situation where the user has to have the controls to personal data without a third party interfering: transferring the orders, monitoring the process or even adjusting the commands depending on the situation. The only method that hides the data from others is encryption while still maintaining access to the network, so the solution is for the user to have the encryption tools locally. The data is given to the server already in hidden format and the tools used are kept locally – this covers keys and the algorithm itself. In that case the keys are hidden and the algorithm is not modified in a harmful manner. The algorithm itself is not hidden, because if it needs to be hidden, this is a possible security flaw[21] - the algorithm might be unverified, eventually revealed and at the same time does not provide its functionality verification to the user. The keys can be locked or removed from the system. If they are locked, and extra locking mechanism is necessary for the keys themselves. This would be an extra key, which is removed from the system. This would not force the user to carry all the keys, when they are not in use. The most common "key" for this purpose is a password – provides an easy way to remove it from the system, when not needed.

The solution of choosing and verifying participants is a multistep process, where the user has to create a packet and pass it on to the other user and the other participant must be capable of verifying the source of the packet. This way both parties have mutual verification. To combat the issue of sending data over unsecure channels needs the previously mentioned methods. In this situation third party solutions for transferring access data are not trusted, since it would mean that data, which is meant to be protected, would become visible to anyone who transfers the access control data. The sent packet is encrypted beforehand to remain hidden from third party, but at the same time must be available for decryption to the recipient. This comes to using asymmetric encryption, since it gives the possibility to share data without needing to meet for the initial information exchange.

The solution to securing access control data on the user device is a choice between having a key, which has to be carried around or a password, which provides enough protection as long as the password creation requirements are followed. This information would be the only way used to lock access control

data. A symmetric encryption algorithm is a commonly accepted encryption choice, where a password conveniently serves the purpose of a key.

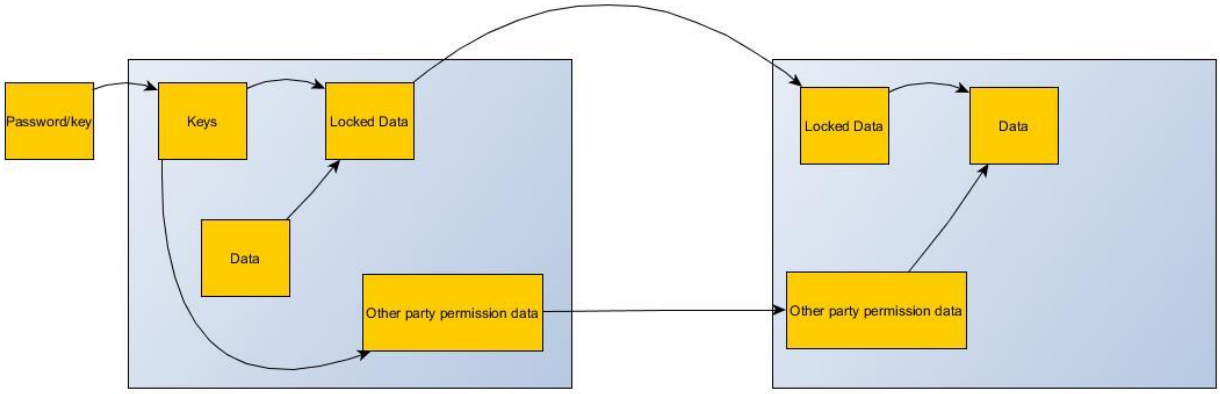

Figure 4. Data management flow

The table above shows the initial layout of the solution. The password/key is positioned outside the user device, while the keys are kept on the device. The data that needs to be protected is locked by using the stored keys. The keys are prepared to be sent to the other party over unsecured link and the data is sent in a similar fashion. Once everything reaches destination, the recipient uses the received key to unlock the data.

The chosen design is simple and cheap to implement. This comes from trying to add new elements to existing structures, rather than editing them. In the shown table, the transfer mechanisms will not be changed.

4 Design

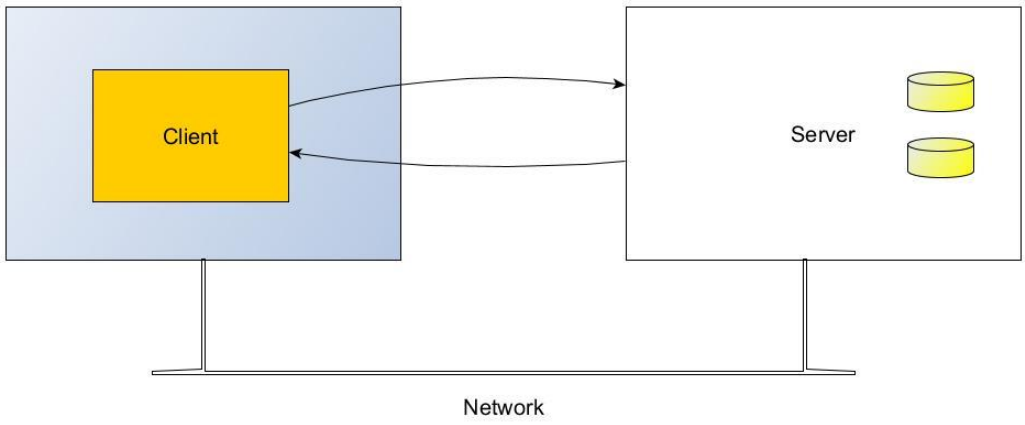

Figure 5. Cryptographic access control

The project is divided among the access control management tool and the data transfer solution. The first is developed throughout the project. The second part is adjusted to work with access control solution. The table above shows the trusted computing base in a system with cryptographic access control, where only the client is trusted. Data can be sent from the server without prior authentication and the client must decrypt the data before it can perform any operations on them. The client then re-encrypts the data before sending them back to the server, which stores the encrypted data back on disk. This design is following an example from “Cryptographic Access Control in a Distributed File System”.[22]

4.1 Design of access control solution

The access control is composed of keys for the data locking, locking mechanism, key sharing mechanism. The data locking mechanism is a symmetric encryption algorithm. The choice is of block cipher. Block ciphers may be evaluated according to multiple criteria in practice. Common factors include:

- Key parameters, such as its key size and block size, both which provide an upper bound on the security of the cipher.
- The estimated security level, which is based on the confidence gained in block cipher after it has not been broken despite major efforts in cryptanalysis for extended periods of time, the design's mathematical proof, and the existence of attacks.

- The cipher's complexity and its suitability for implementation in hardware or software. Hardware implementations may measure the complexity in terms of energy consumption, which are important parameters for resource-constrained devices.
- The cipher's performance in terms of processing throughput on various platforms, including its memory requirements.
- The cost of the cipher, which refers to licensing requirements that may apply due to intellectual property rights.
- The flexibility of the cipher, which includes its ability to support multiple key sizes and block lengths.

The exact algorithm is AES with a 256 bit long key. Currently the key space for AES-256 is $2^{254.4}$, which is slightly smaller than the theoretical maximum of 2^{256}, but is still larger than what current technology can look through. This algorithm is chosen, because it is considered to be the standard for symmetric encryption. It won “National Institute of Standards and Technology” competition, which provided enough openness to be approved by the cryptographic community.[23] The highest key length for the system is chosen because of user preference and their limited understanding of technology. The level of security does not suffer and there is less convincing the user about the choices.

The exact AES-256 implementation is chosen from the Microsoft cryptography solutions. This is based on the compatibility with the methods used by the company dealing with data transfer. Microsoft encryption implementation is considered to be trustworthy, so this is an acceptable choice. The alternatives to this are open source implementations, such as, OpenSSL or Crypto++. They are also trusted by the community, but lack documentation for used methods. OpenSSL lacks documentation for the methods they provide and recent security flaws related to one of OpenSSLs’ methods revealed that the entire solution is maintained by few people and could lose support soon in favor of remaking the implementations. So the more supported choice is preferred, even if it remains partially hidden. High speed and low RAM requirements were criteria of the AES selection process. Thus AES performs well on a wide variety of hardware, from 8-bit smart cards to high-performance computers. The encryption algorithm is used for encrypting large amounts of data in a reasonable amount of time.

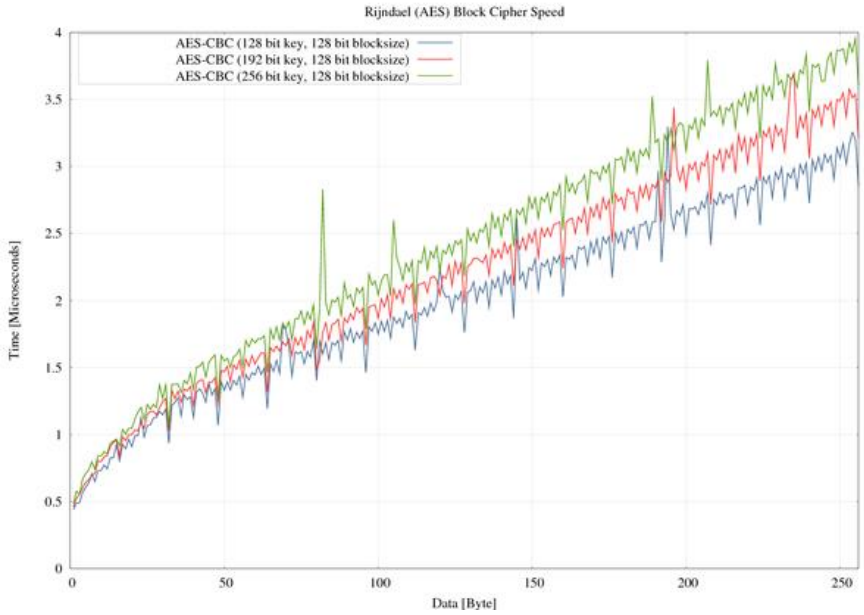

Figure 6. AES key/encryption speed diagram[24]

The table shows that the speed between different keys sizes is negligible and therefore there is no problem to be the use the longest key speed wise.

Key sharing mechanism is composed of asymmetric encryption algorithm. The chosen algorithm is RSA with the longest used key – 4096 bits. Currently it is assumed that this length is enough to prevent the encrypted data to be decrypted. And it should remain so for the foreseeable future. Asymmetric encryption is relatively slow and not used for encrypting large amounts of data directly. In addition to that the data must be shorter than the key, so the encryption process becomes even slower. This encryption is used to deal with small amounts of data, so the speed is not a concern with modern computers. As an asymmetric encryption algorithm, it has two separate keys for both parties, it is more convenient to use for key sharing.

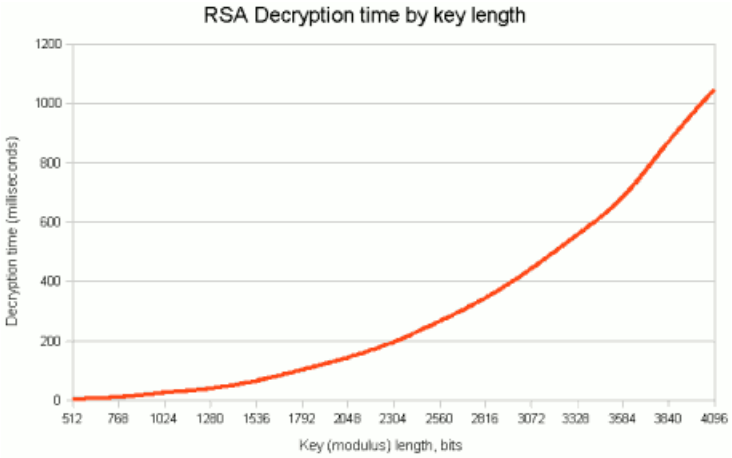

Figure 7. RSA key length/decryption speed diagram[25]

The decryption time of RSA increases around six times, when the length of the key doubles. As mentioned, small data sizes make the encryption time tolerable.

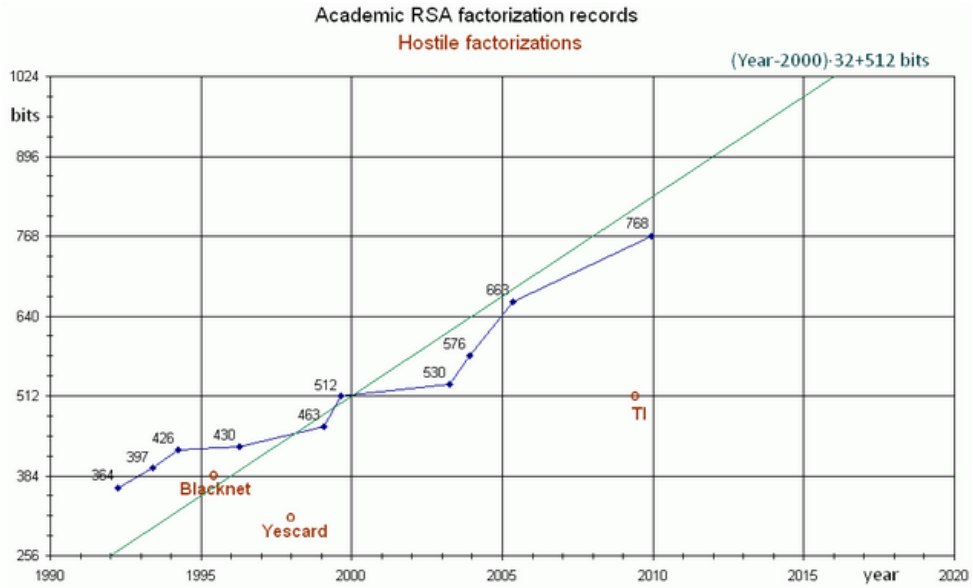

Figure 8. RSA factorization extent diagram[26]

The length of the key has to be considered for the future as well. The table above displays how far the academic has come in factorizing the RSA keys. The conclusion is that the key must be above 1024 bits and seeing that data is small, so it takes a reasonably short time. At this point the largest size is used to convince the users of safety of data and provide the longest lifetime for a single encryption implementation.

The keys used for locking the mechanism on user device are also AES keys, so the length requirements must be fulfilled. These keys are passwords of variable length, so the user is given the freedom to go from a much smaller key to the required size or even above the required size. This is commonly solved by putting the password through a hashing algorithm to have a constant size for the AES key. The hashing algorithm choices are between MD5 and SHA variations. The MD5 is 128 bit in output and the SHA can go from 128 to 512 bits. The key for locking the control mechanism is in line with other purposes for the encryption, so the size must be 256 bit, therefore the SHA-256 algorithm will be used. The hashing algorithm is common in many systems, so its' implementation is widely available. The hashing is only meant to provide a conversion from a password to an AES type of key. It does not have to take care of having the widest possible range of results, since this data is kept on the device in the form of data encryption keys.

The data being transferred over insecure network must be presentable in the format readable to the user. This is a choice between the base64 and hex format. The first one converts the text by increasing the size by 50%. The second choice doubles the size of output text. In both scenarios the data that will have to be encrypted, are small, so the increase in size is negligible. The data is sent over the networks, where both encoding types are supported. Base64 is a standard in sending data related to asymmetric encryption, so it will be the one used in this scenario as well. This includes the keys themselves and the encrypted key data.

The key creation for the symmetric algorithm is left to a random number generator, since the key is just random bits. The random number generator itself has to have a satisfactory randomness, since no currently available RNG is capable of true randomness. The cryptographically random pseudorandom number generator uses good block cipher as its method. The idea is to run the cipher in counter mode, then encrypt successive values using an incremented counter. On its own, this would produce statistically identifiable deviations from randomness, so the method has to have seeds, which are periodically changed. Currently it is changed on each request or when no more than 1 megabyte of data is generated. Computer cryptography uses integers for keys. Keys are randomly generated using a pseudorandom number generator (PRNG). In other situations, the key is created using a passphrase and a key generation algorithm, which involves a cryptographic hash function such as a variation of SHA as already mentioned. For a seed to be used in a pseudorandom number generator, it does not need to be random. Because of the nature of number generating algorithms, so long as the original seed is ignored, the rest of the values that the algorithm generates will follow probability distribution in a

pseudorandom manner. Microsoft Windows editions newer than Windows 95 use CryptoAPI (CAPI) to gather entropy in a similar fashion to Linux kernel's /dev/random. The Linux kernel generates entropy from keyboard timings, mouse movements, and IDE timings and makes the random character data available to other operating system processes through the special files/dev/random

Windows's CryptoAPI uses the
binary registry key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\RNG\Seed to store a seeded value from all of its entropy sources.

Early versions of Netscape's Secure Socket Layer (SSL) encryption protocol used pseudo-random quantities derived from a PRNG seeded with three variable values: the time of day, the process ID, and the parent process ID. These quantities are often relatively predictable, and so have little entropy and are less than random, and so that version of SSL was found to be insecure as a result. The problem in the running code was discovered in 1995 by Ian Goldberg and David Wagner, who had to reverse engineer the object code because Netscape refused to reveal the details of its random number generator. This is known as security through obscurity – a product is less likely to be attacked against random or automated attacks if the system has discovered vulnerabilities. A variant of the basic approach is to rely on the properties (including whatever vulnerabilities might be present) of a product which is not widely adopted, thus lowering the prominence of those vulnerabilities (should they become known) against random or even automated attacks.

In August 2013, it was revealed that bugs in the Java class SecureRandom could generate collisions in the k nonce values used for ECDSA in implementations of Bitcoin on Android. When this occurred, the private key could be recovered, in turn allowing stealing Bitcoins from the containing wallet.[27] Looking at various possible attacks, it is divided into

- **Direct cryptanalytic attack -** when an attacker obtained part of the stream of random bits and can use this to distinguish the RNG output from a truly random stream.
- **Input-based attacks -** modify the input to the RNG to attack it, for example by "flushing" existing entropy out of the system and put it into a known state.
- **State compromise extension attacks -** when the internal secret state of the RNG is known at some time, use this to predict future output or to recover previous outputs. This can happen when a generator starts up and has little or no entropy (especially if the computer has just been

booted and followed a very standard sequence of operations), so an attacker may be able to obtain an initial guess at the state.

The choice is between Microsoft or OpenSSL random number generators, since these packages are used for other purposes as well. Microsoft uses an unpublished algorithm to generate random values for its Windows operating system. In November 2007, Leo Dorrendorf published a paper titled Cryptanalysis of the Random Number Generator of the Windows Operating System. The paper presented serious weaknesses in the Microsoft approach. The paper's conclusions were based on disassembly of the code in Windows 2000, but according to Microsoft apply to Windows XP as well. Since the project solution should be available on Windows Xp as well, the pseudorandom number generator is taken from OpenSSL package. The OpenSSL implementation currently does not contain any publicly known attack as long as the newest API version is used.

The extra elements used by the company will be analyzed when it is reached.

4.1.1 Specification of password and hashing

The key used in AES encryption is expanded to accommodate various size data encryption and keep the data securely hidden. Detailed analysis of each step is skipped, since it is irrelevant to the project. The following are the steps used to derive the key to a further state during encryption[28]:

1. KeyExpansion—round keys are derived from the cipher key using Rijndael's key schedule.
2. Initial round:
    1. AddRoundKey—each byte of the state is combined with a block of the round key using bitwise xor.
3. Following rounds without the last one:
    1. SubBytes—a non-linear substitution step where each byte is replaced with another according to a lookup table. It is used to avoid linear patterns in the cipher.
    2. ShiftRows—a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
    3. MixColumns—a mixing operation which operates on the columns of the state, combining the four bytes in each column.
    4. AddRoundKey
4. Final Round:
    1. SubBytes
    2. ShiftRows
    3. AddRoundKey.

These steps display that the key goes through multiple states of modification to reach the end result. The key modification is carried out for 14 rounds. [29]

R0 (Key = 00000000000000000000000000000000) = 00112233445566778899aabbccddeeff
R1 (Key = 62**63636362636363626363636**2636363) = 011a85ba96**049815**cf**655f97**b0**88e9c0**
R2 (Key = 9b9898c9f9fbfbaa9b9898c9f9fbfbaa) = 1beb461fff271da10d7a08ad1f01b177
R3 (Key = 90973450696ccffaf2f457330b0fac99) = 5f1e33b73856d349ab071e13816472be
R4 (Key = ee06da7b876a1581759e42b27e91ee2b) = 7f88a9b53495b7448c1fb3e2ebaedbd1
R5 (Key = 7f2e2b88f8443e098dda7cbbf34b9290) = ed214bdeffc2ebe9ba186c165c4945f6
R6 (Key = ec614b851425758c99ff09376ab49ba7) = 3bcc5d8ba7dd354f8aad99844c782ce3
R7 (Key = 217517873550620bacaf6b3cc61bf09b) = 5936c0b765930e7147e2a65d805a4483
R8 (Key = 0ef903333ba9613897060a04511dfa9f) = 3411734ea04b62250cb785a63870abe5
R9 (Key = b1d4d8e28a7db9da1d7bb3de4c664941) = 01ca3647fc5d4574b992586751bfef28
R10 (Key = b4ef5bcb3e92e21123e951cf6f8f188e) = c8a331ff8edd3db175e1545dbefb760b

Figure 9. 128 bit AES key expansion with NULL key

R0 (Key = 61000000000000000000000000000000) = 61112233445566778899aabbccddeeff
R1 (Key = 03**63636303636363036363630**3636363) = 63960935f7**049815**ae655f97d1**88e9c0**
R2 (Key = fa989818f9fbfb7bfa989818f9fbfb7b) = 6f6cc15c76bd43a726b55158e0bc5ae4
R3 (Key = f197b981086c42faf2f4dae20b0f2199) = 8cca691be876164d8c1d5c45520a1a3f
R4 (Key = 8f6a57aa8706155075f2cfb27efdee2b) = 30d5009e509c9fa50eab1b5dcbb2a6c4
R5 (Key = cb42a6594c44b30939b67cbb474b9290) = 09171d1c63b4d7ac48d02c05eb68f7f1
R6 (Key = 580dc6f9144975f02dff094b6ab49bdb) = 063f505f9d7691dd7341e565cddf23d3
R7 (Key = 95197ffb81500a0bacaf0340c61b989b) = bc102b674ab6d1c98233ec9ef4999e29
R8 (Key = ba5f6b4f3b0f614497a0620451bbfa9f) = c94a33315cdcf6fcb4a11a8f44c2eea7
R9 (Key = 4b72b09e707dd1dae7ddb3deb6664941) = 8519507e5decab8bfaf44dcfd07f951e
R10 (Key = 4e4933d03e34e20ad9e951d46f8f1895) = d987d0a2728bc8f9f43b02e91f5b7a1f

Figure 10. 128 bit AES key expansion with single symbol key

Figure 16 and Figure 17 displays that parts from modified key(left side) and data being encrypted(right side) match, when the key has small changes. The matching parts are marked in bold. AES 128 bit key is used instead of the 256 bit key to more easily display the relevant parts. The 256 bit key follows similar pattern.

The hashing function in this scenario is used to increase the randomness of the key as seen by the encryption algorithm. The similarity of the keys is hidden by using the hashing function. The function changes the key bits to a great extent. One changed bit results in at least 50% changed resulting bits. In case the user changes a single symbol of the key, the attacker does not gain any additional information by comparing the hashes from the older key and the new one. This is confirmed by no know successful preimage attack on the sha-256 hashing function. A change in a single change in the key in addition to the hashing function, successfully shuffles the key when it is passed to the key expansion function.

By using a hashing function, the user can have a varying length password and slight changes to it will not be as easily noticeable. The most popular passwords remain more or less the same for multiple years, so hashing will make it harder to determine the password.

4.2 The access control management system design

The access control system design is divided into segments for creating a new symmetric key, asymmetric key, key import windows, key export windows and key locking/unlocking mechanisms. The various components make use of symmetric and asymmetric encryption.

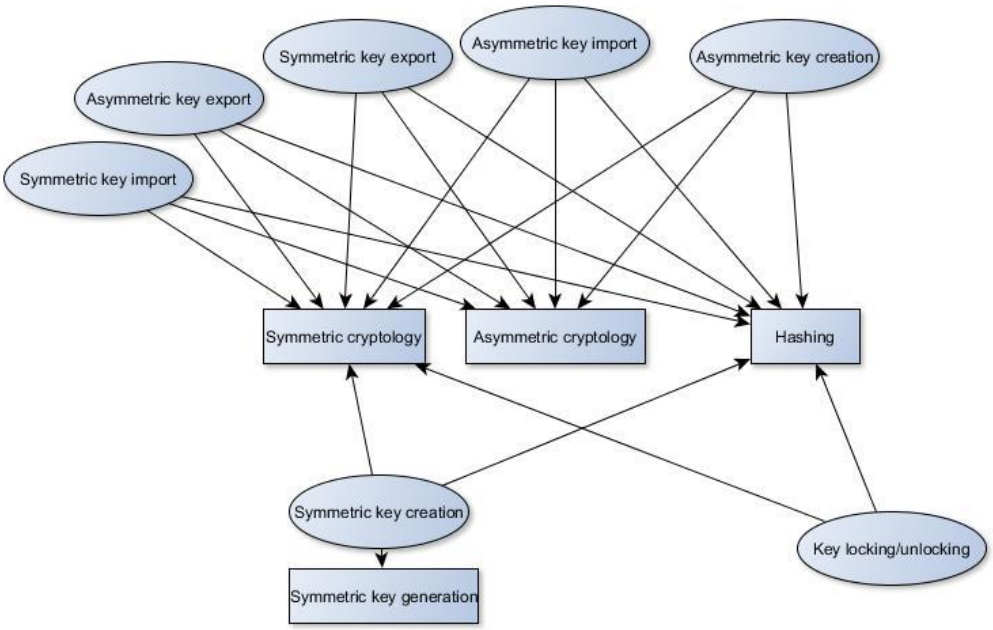

Table 7. Access control management component diagram

Table 7 displays the controls in the bubbles and the underlying technologies in the boxes. Symmetric key creation and locking mechanism does not use asymmetric encryption, while the remaining controls make use all three mechanisms. Symmetric key creation uses a separate method(random number generator), so it has the mechanism listed, while the asymmetric key is created using methods from asymmetric cryptology. Every control has a link to the hashing mechanism, since it is used in password preparation for use in the encryption and decryption of keys. Symmetric cryptology is used in locking down all the keys from unauthorized access and the asymmetric cryptology is used for key export and import actions.

Next the individual controls are described.

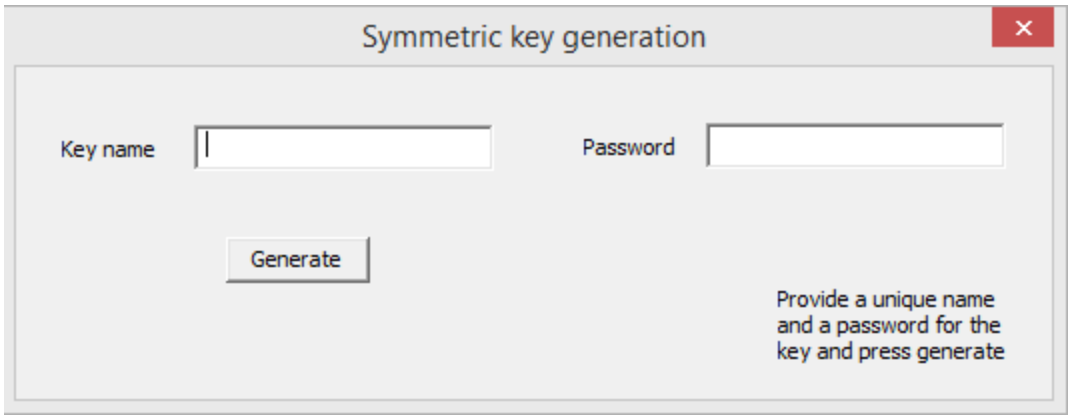

Figure 11. Symmetric key generation box

The symmetric key generation mechanism is used for creating keys for hiding personal data from outsiders. The key is created specifically for AES 256bit edition, so the user is not given the functionality to change the size of it. Since the key is not designed to be directly compatible with another tool, the naming and structure of stored key is not analyzed in this sense. The naming convention has to be understandable for the user and allowed on the Windows file system. The key file itself will have no specific extension to identify it. Since the created key is only available in memory in raw format, it can be stored only in locked format. The key is locked using another AES 256 bit key, which is made from the password. The key creation method must have the password specified as well.

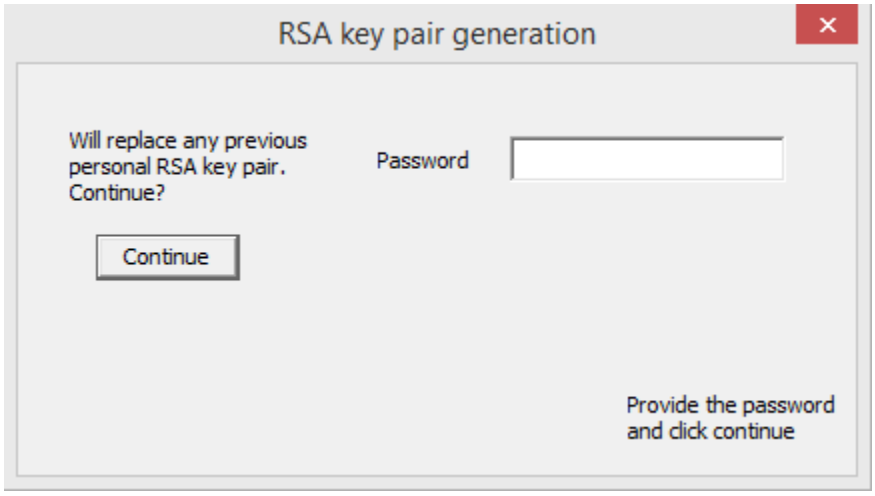

Figure 12. RSA key generation box

The asymmetric key creation mechanism is used for creating asymmetric keys for RSA 4096 bit edition, so the user is not given the freedom to change the size. The naming is convention is the same as with the symmetric keys. Since the keys for symmetric and asymmetric methods are stored in separate

locations, the names for the keys can be the same as long as they are not the same in the same folder. The asymmetric keys are created in combinations of two(private and public), so the name during creation is appended with private and public words. The key pairs user creates have a separate folder from the keys, which are imported from other locations. The names have no need for a specific extension, since compatibility with other software is not taken into consideration and the access control system is looking into the folder location only.

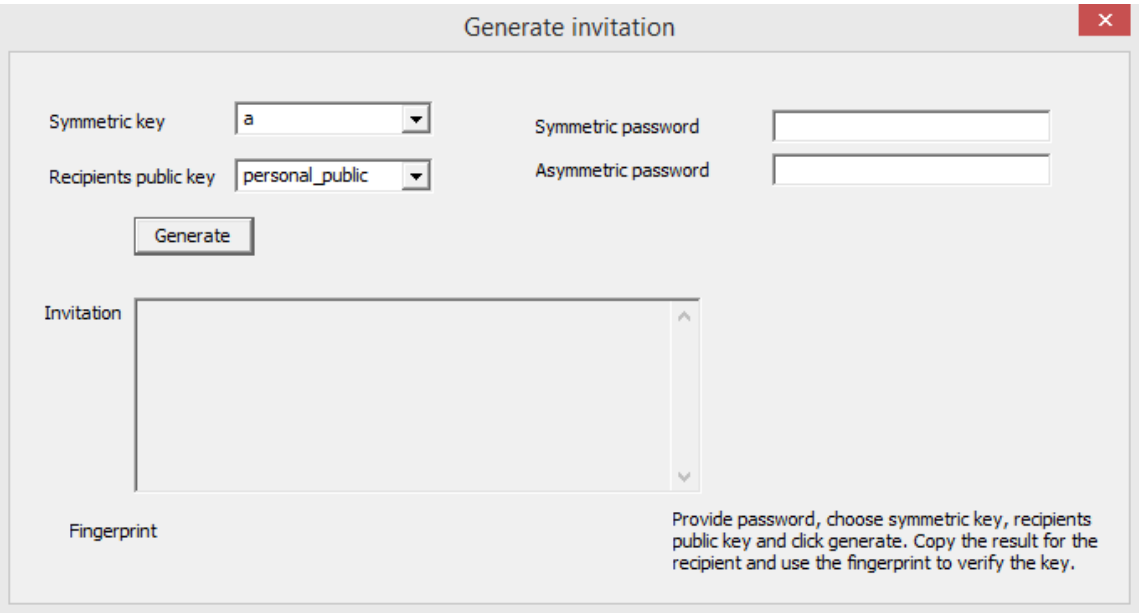

Figure 13. Invitation creation box

The symmetric key export mechanism uses a single symmetric key and the public key obtained from another user. That user is the recipient of the exported key. The symmetric key is encrypted with the asymmetric key and the result is sent over the publicly available network interface. The keys must be unlocked with their passwords, since otherwise there would be a potential possibility of an outsider triggering the key export method and claim the result. Since the key must be transferable over public networks, it is converted from raw data format into base64 design. The user has a range of methods to transfer the key over the network, since the output is fully visible in text format. The data has to be verified by the user in on the other end, so the key is given a fingerprint. The fingerprint is a hash function result with a truncated output. The purpose is only to show both parties that the key has not been corrupted and that the intended key has been sent. The hash is truncated to 16 symbols from the 32 symbols in sha256 hash algorithm. This hash algorithm remains mainly because the same hashing

algorithm is used for other work. The speed differences are negligible in this scenario, so they are not regarded.

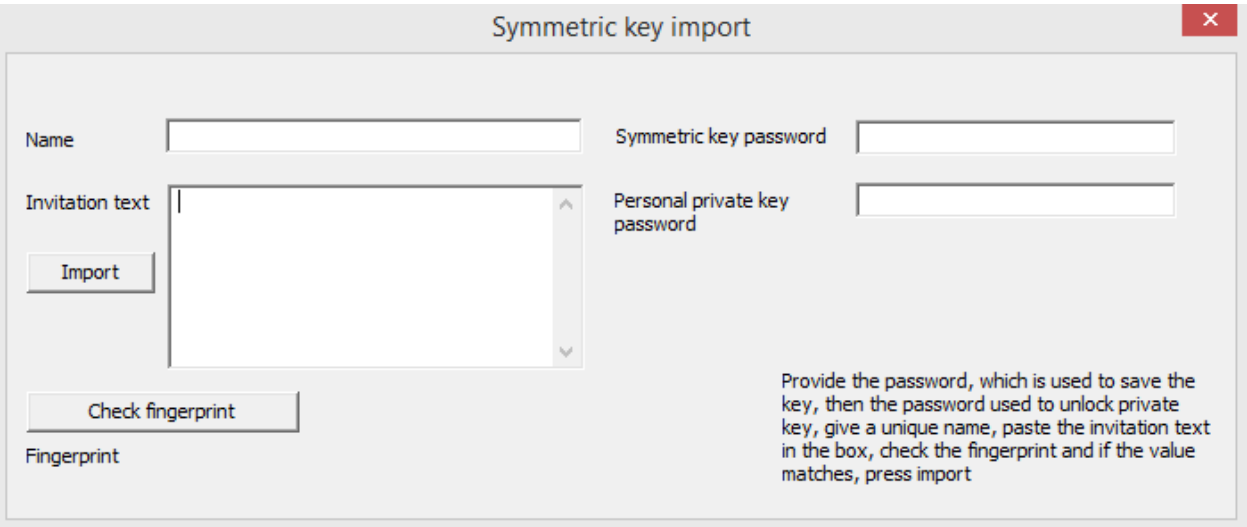

Figure 14.Symmetric key import box

Symmetric key import mechanism uses a single encrypted symmetric key to get a data unlocking key. The encrypted key is in base64 format, so it is pasted in and given name and password. The name serves the purpose of giving the path where to store the key and the password is for locking the key from unauthorized access. Before the key is saved, the user has to verify that the fingerprint matches the one provided by other party. The key is extracted by decrypting an asymmetric encrypted message. The users' private key is used for key extraction, since the other party encrypted the symmetric key with the public edition of the users' key. The decrypted key is saved in the same fashion as the symmetric key, which is created from scratch. The symmetric keys are not different depending on which users' device it was created.

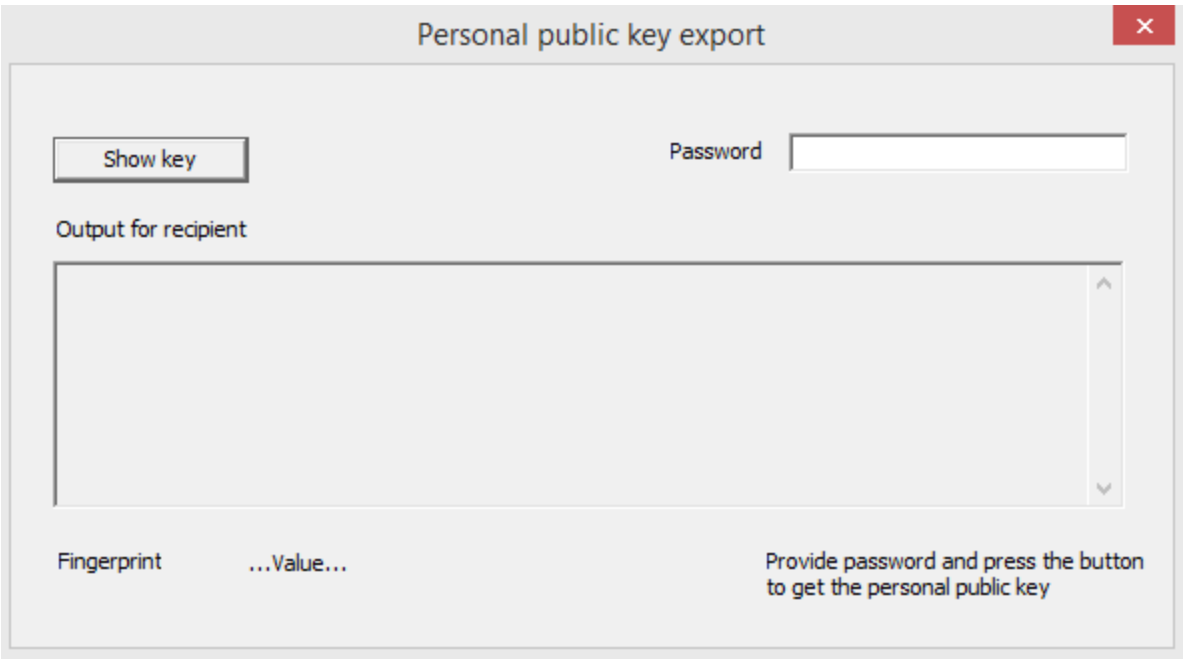

Figure 15.RSA key export box

Asymmetric key export mechanism uses a single public key as the exported element. The user has to provide the password to protect from unauthorized usage. The key is given the base64 format for convenient output. The user is shown the fingerprint of the key as a confirmation of correct key transfer, meaning that the other user is going to compare both fingerprints.

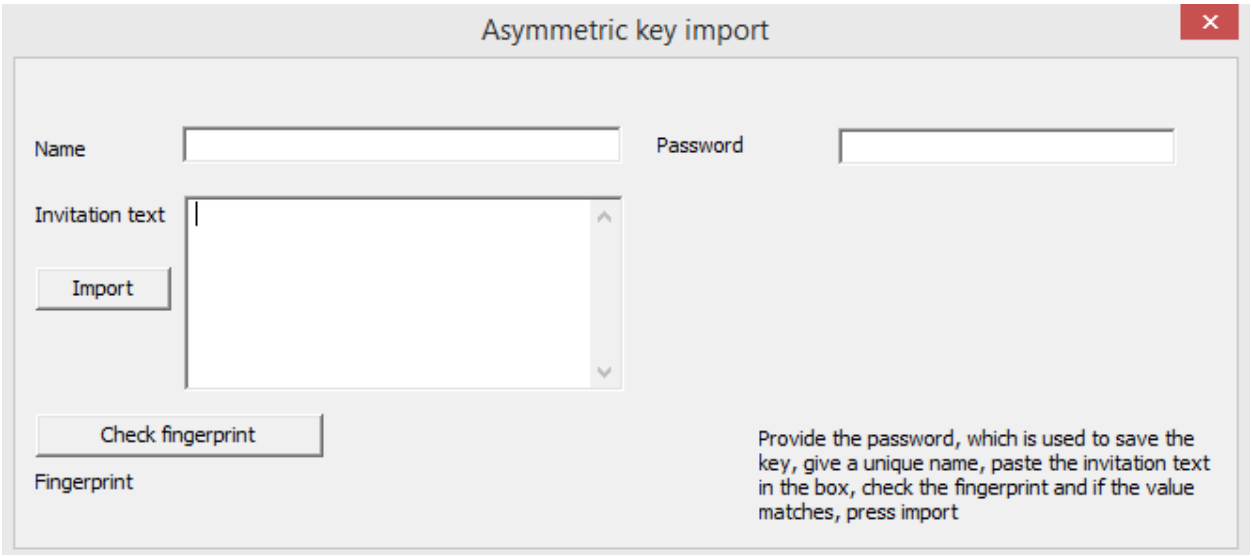

Figure 16.RSA key import box

Asymmetric key import mechanism uses the public key from the other participant received through a publicly available network. The key has a fingerprint value for convenient key verification. The key has the password for locking the received public key with a local mechanism. The name for the key must be unique only from other public keys.

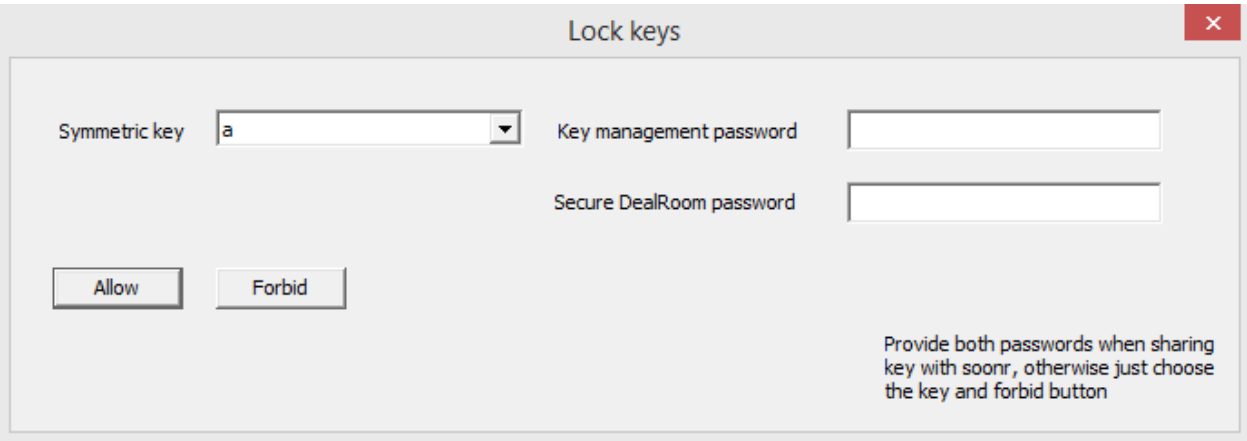

Figure 17. Key locking box

The symmetric key permission mechanism uses symmetric keys in the folder and replaces the password they are locked with. This allows the keys to become available to the data transfer system without giving the personal password to it. The chosen key is re-locked with another password, which is known to the other system. The key itself is stored in another location, which is given to all the “unlocked” keys. The name remains the same, so that the system and the user can maintain a link between which key is which. Since the key is just pushed to another folder and does not have any trail in other system locations, the user just deletes the key, which has been shared with the system. In the same fashion, the user can replace the password used for data transfer key.

4.3 The flow of the access management tool

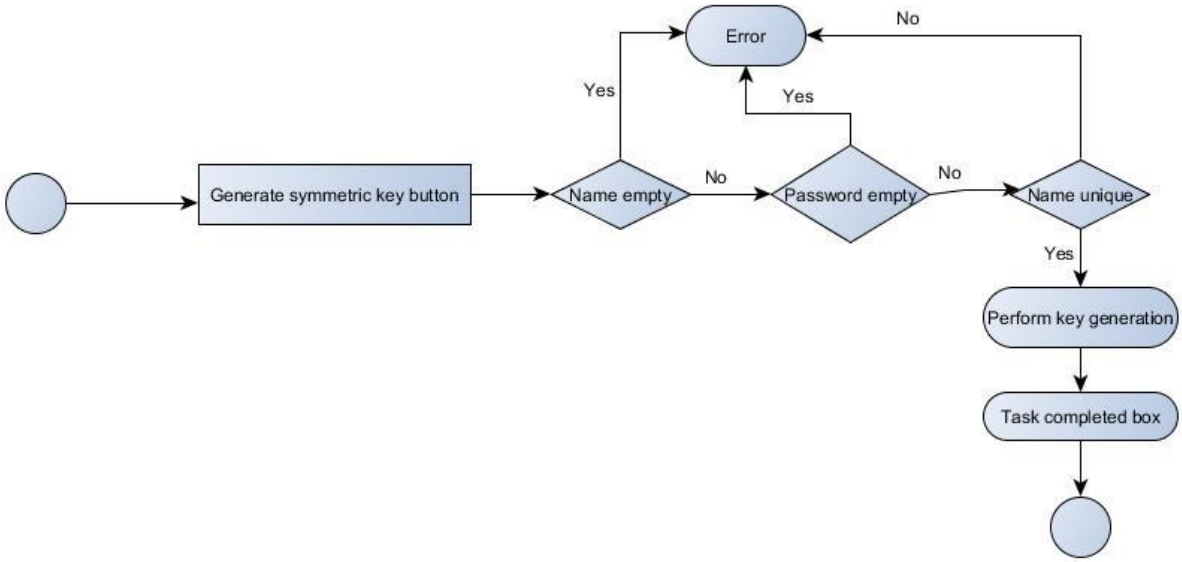

Figure 18. Symmetric key generation flow

The generation of symmetric key is linked to the checking of the input into the password and name fields. If one of them is skipped, then the entire process is canceled. The user cannot change the inputs during execution, so the result of the check cannot change. The uniqueness of the name is necessary, so the already existing keys would not be lost. When the choices are made and approved, the internal processes finish the work.

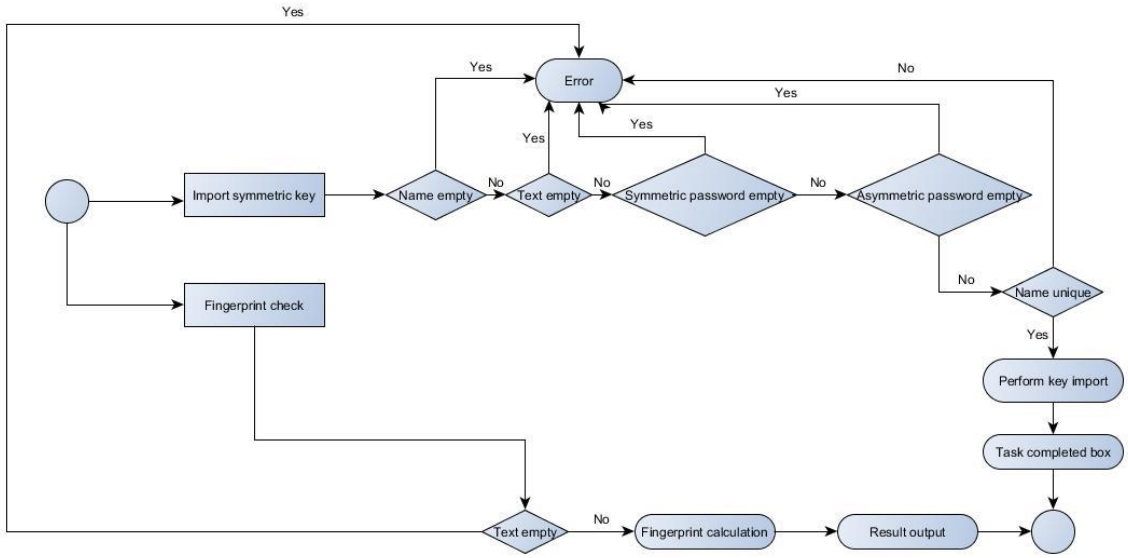

Figure 19. Symmetric key import flow

Symmetric key import is a two-step process. Both of the processes work individually, so there is no interference when it comes to data sharing. The fingerprint is executed as long as there is an input in the box. The result is displayed in a field, which cannot be edited, so no collision happens in the field.

The import of symmetric key is executed when the name, text and password fields are filled. The checking is done for empty fields and the uniqueness of the name. After this data is provided, the user cannot interrupt further execution. The internal methods stability is not dependent on the user input. The end result is provided in a file, which is not the same name as the other files. In this situation no accidental damage can be done. If after executing the import, another file is written under the same name, it will be overwritten, since the tool does not use multiple threads for the same process.

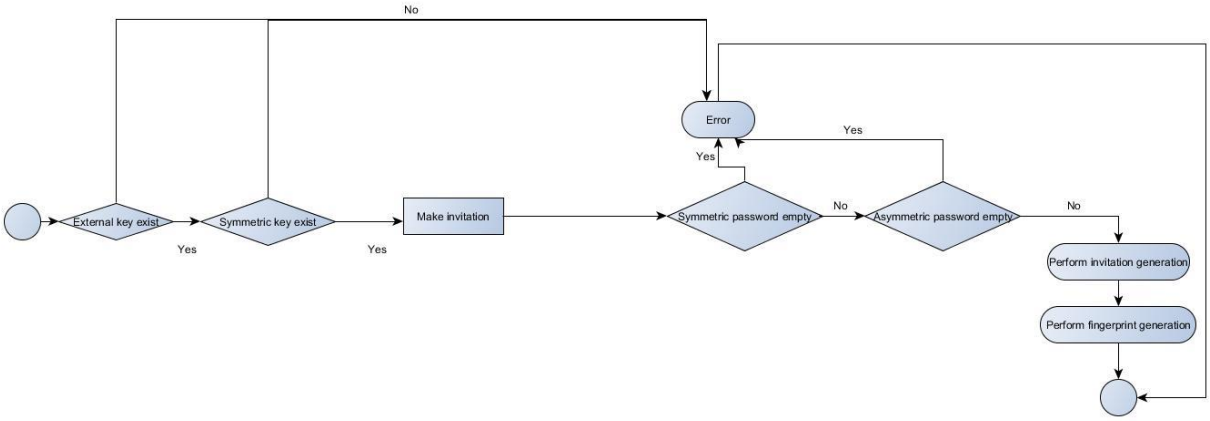

Figure 20. Invitation creation flow

The invitation creation is done by doing a check on the keys, which will have to be used for the next processes. If one of the keys does not exist, the process will be aborted. After choosing the keys to use and the passwords are not empty, the invitation is generated. The user cannot change the values for the passwords, so the execution is not interrupted. If the passwords provided turn out to be incorrect, the process is aborted and the user begins the set up again. So incorrect passwords do not return incorrect results. The output is in a separate field and the user causes no impact on it.

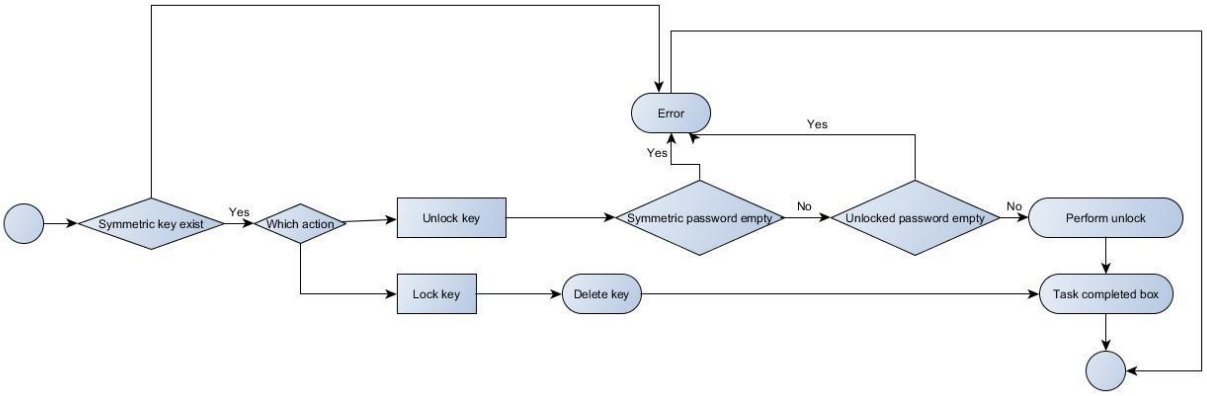

Figure 21. Key locking flow

The key unlocking mechanism is one of the mainly used functions by the user. The passwords to use are listed in a dropdown list and the user cannot pick a non-existing key. The work is executed when the user gives the passwords to the tool. The process executes, if the provided password is correct, otherwise it will abort the process. No check can be carried out on the second password, since it is not used to unlock anything. If the file in the password location is not a password, it will be treated as a password during execution and the check will fail.

The locking of the key is does not require any passwords, so the choice is only from the key dropdown list, where the keys are from the locked key folder. If the key no longer exists in the unlocked folder, the deletion will be executed either way.

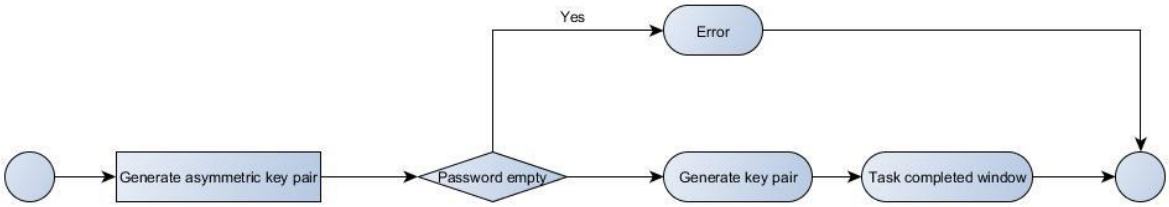

Figure 22. Asymmetric key generation flow

Asymmetric key generation requires only a new password for the process to execute. The users' choice does not cause the process to stop. The result overwrites any information with the same name, so nothing is aborted at this point as well.

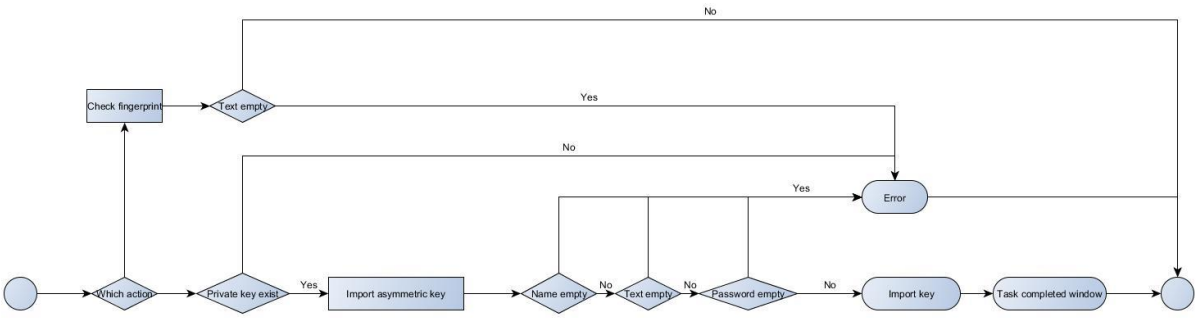

Figure 23. Asymmetric key import flow

Asymmetric key import is a two-step process, so the user does a fingerprint check with the text provided. As long as the field is not empty, the fingerprint will be displayed. The user does not have more options in this sense.

The symmetric key import requires the text, so the user can provide incorrect data. In that case, the process is cancelled midway, when the data cannot be converted to a key. The execution starts, when the name and the password are provided and the name is the only field, which needs to be unique/correct. The password is set up for the first time, so the choice cannot be compared to any previous input.

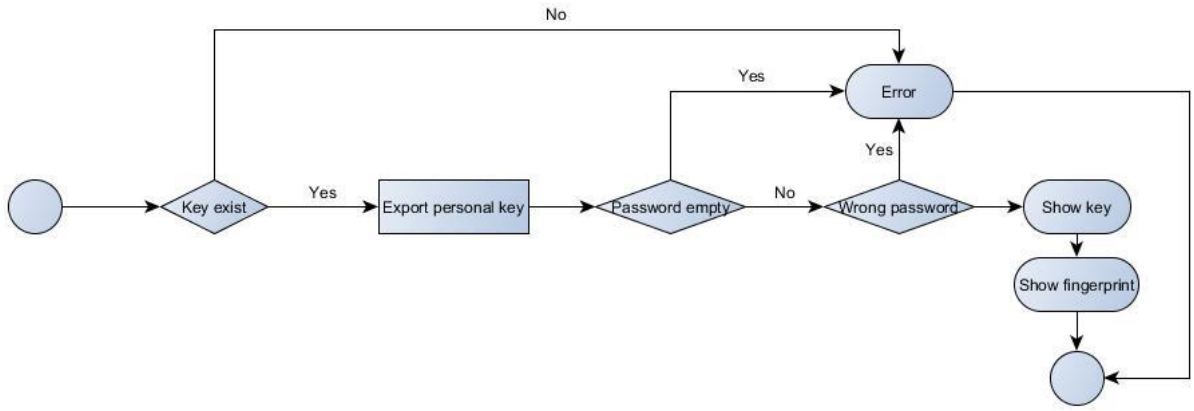

Figure 24. Personal public key export flow

The success of personal key export relies on the value of the password provided and whether the password is provided at all. In case the key does not exist, the method is not launched at all. If everything is provided completely, the result is given in a non-editable box.

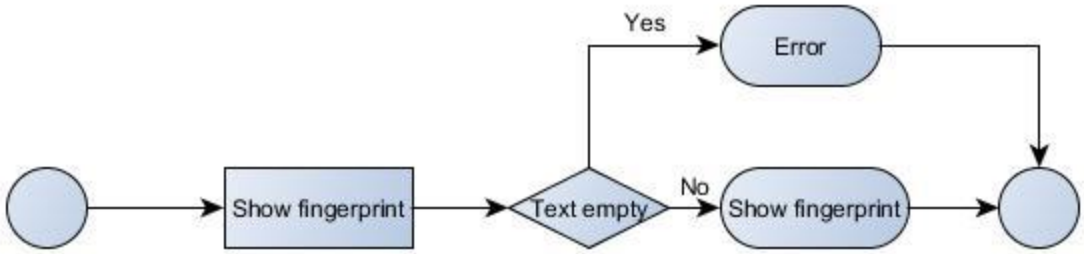

Figure 25. Fingerprint display flow

The fingerprint needs the text of any kind to work with and the result is then provided to a non-editable box.

4.4 The flow of Soonr interactions

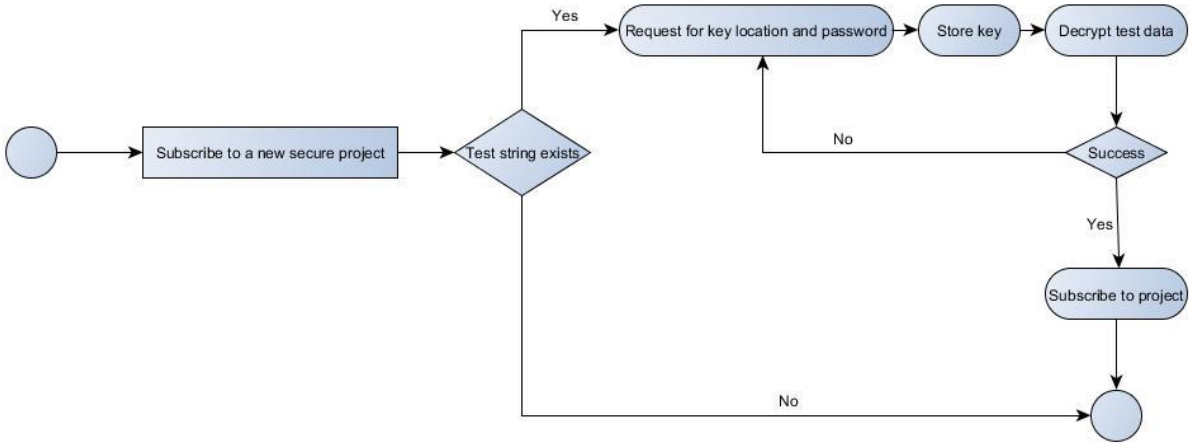

Figure 26. Secure project subscription flow

The secure project is subscribed to, when the existing projects on the server side report a test string used to determine the correct key. The user is given a window to type in a password used for the key and the location is given in the form of a name. The name is shown as a dropdown list and the user chooses, rather than types the name in. The key is stored by overwriting any file in a predetermined location with the same name. This is used for the project; the final design will have a database to store the keys. The test data is decrypted with the key and if the key is correct, the project will be subscribed to. The wrong choices the user makes will stop the subscription, so incorrect does not pass as correct.

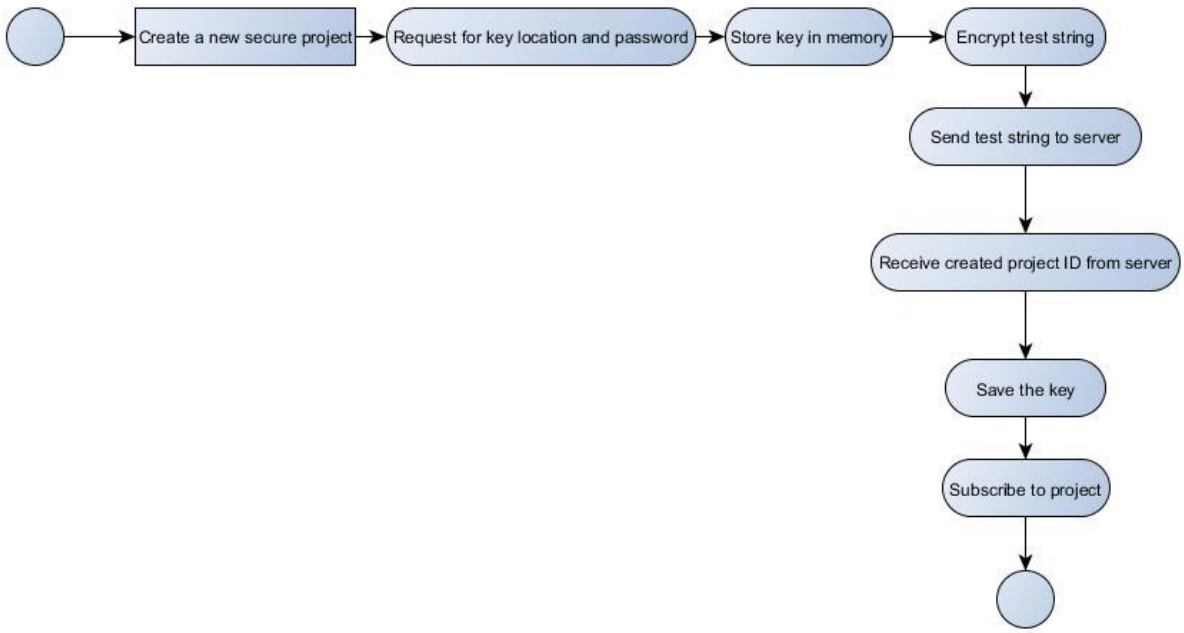

Figure 27. Secure project creation flow

The secure project is created, when the user chooses this command. The choices for the key are checked with the password and if the key fails to unlock, the creation is aborted. After this the entire process is done automatically. The saving of the key pointer is done in a way that any previous data will be overwritten. The test data is sent to the server, so storage is not a concern.

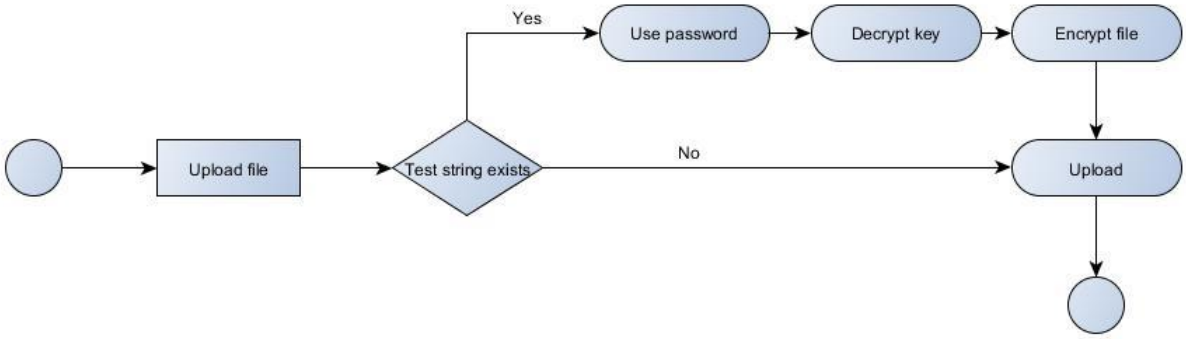

Figure 28. File upload flow

The file is uploaded after there is a secure project, so the key is already in a known location and the test data has already verified the key. The user provides the password, which only needs to be successfully decrypted. After that the system takes over – encrypts and uploads the file.

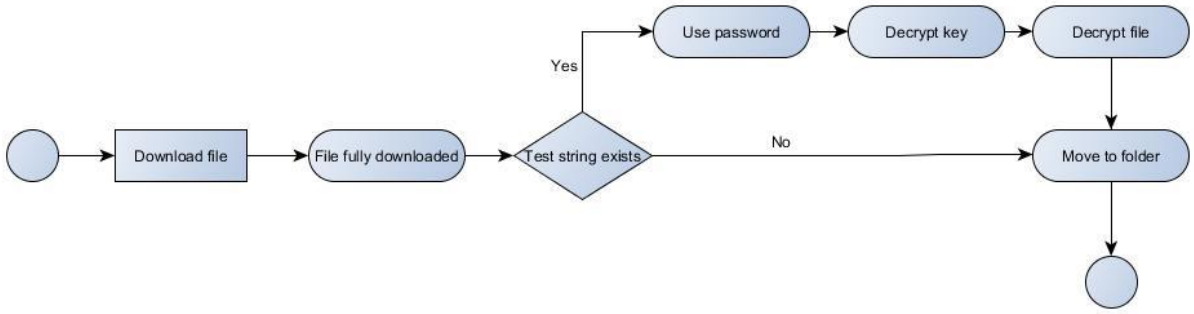

Figure 29. File download flow

The file download is done in a similar fashion to the upload part. The user is only given the password as the element with errors. So, the decrypted key is checked and if it matches the test, it is considered that the remaining elements are correct.

5 Access management software implementation

The first part of the implementation is the symmetric key generation, since it is the part, which is used for locking data and is also used in sending invitations to other parties. This method uses AES-256 for encryption and decryption. The encryption is done using Windows Crypto solution. The method is used to save generated keys, which as analyzed are generated using OpenSSL implementation. The key generation itself is left to the OpenSSL mechanisms entirely. The output from the generation is a sequence of bytes. The acquired data is sent to the AES encryption method. In addition to the key data, an initialization vector is also acquired in a similar fashion, when the OpenSSL random method generates the necessary bits. In this case, the length of the data is 16 bytes. This data is also returned to the AES encryption method. The third element taken for the encryption is the password for securing the key. The password is pushed as a character array to the hashing function. The function is taken from the OpenSSL library. The hashed password is returned to the encryption algorithm with a 32 byte length. The data is copied in the encryption algorithm to a cryptographic blob format. The format is a structure containing data in the form of bytes and the length of the code in DWORD. In addition to the data, an extra piece of data is added. It is a string known in advance and which is a static variable throughout every encryption. This is done to have a way to verify whether the decryption is carried out successfully. The other way would be to decrypt a key, do the decryption or encryption of the data and then manually look at the result hoping that it is something reasonable. The encryption method itself is set up by getting the CryptAcquireContext with PROV_RSA_AES setting, which is the base for Windows AES encryption. Then the key data is copied into compatible structures, which are the blob type and then the key is imported into the encryption design. The initialization vector is copied from the byte format to CryptSetKeyParam. Finally, the data to encrypt is copied to the blob format. The encryption is carried out in one call, rather than keep pushing data continuously. In case the encryption fails, everything is cancelled and data is removed from memory. This includes the key and the information that needed to be encrypted. When the encryption is complete the data is saved to a text file in a predetermined location. This is done by appending the initialization vector to the beginning of the data and then data is saved using FILE method.

The next step is the providing the key for external programs to use. The user provides a password for the locked key, chooses the key to use and then proceeds to decrypt the key. The password provided is hashed as in the case with the encryption key. The key name is used to find the exact file to decrypt.

The path to the key is partly given in a predetermined location + the key name itself. The encrypted key is read from the file into a buffer, where the key data is split from the initialization vector. Since the length of each part of the data is known because of the way the code is written, the length of the key data depends on the fact that this is a decryption part. The encryption part would have another size given in advance as well. After this data is set up, the CryptAcquireContext is called with a PROV_RSA_AES variable to use the AES decryption. The key and the initialization vector are set by calling CryptImportKey and CryptImportKeyParam. The data to decrypt is copied to a buffer and passed to CryptDecrypt function in one call. If the decryption succeeds, the resulting data must contain the verification string, which is extracted from result and compared in the next step. If it matches, then it is assumed that the decryption is done correctly and the data is passed to further methods. This is done by copying it to the variables, which will be returned through pointers and then destroying the data in the temporary variables. The encryption of the key is done by passing the password used to prepare the key for external programs to the AES encryption method, then passing the decrypted key that needs to be encrypted. In addition to the password, an initialization vector is created by calling an OpenSSL method to generate random bits and passed to the encryption method. The key has the verification string added to it and everything becomes ready for encryption. The encryption method is set up by calling a CryptAcquireContext, CryptImportKey and CryptImportKeyParam. The data is encrypted by calling CryptEncrypt once with all the data in one go. The result is appended with the initialization vector and then the data is saved to a new file, which is then handled with further functions. The location, where the file is saved is partly hardcoded, with the name of the file and the root folder being variables.

The creation of personal RSA key is done by calling an RSA key creation method having provided a password for storing the results during the same call. The OpenSSL methods are called to do the key generation. The structure is set up by calling BN_new, BN_set_word, RSA_generate_key_ex methods. The result is split into public and private key by calling BIO_new, PEM_write_bio_RSA_PUBKEY, PEM_write_bio_RSAPrivateKey and after it both results are passed to the AES encryption method. Each part of the RSA key pair is given the encryption method with the password for locking them provided as well. The keys go through the set up steps as with the previous encryption cases: appending a verification string, copying data into buffers, giving new initialization vectors, encrypting, appending the vector to the result and saving the data to a predetermined location with a predetermined name. In this case only one pair of personal keys can be created. If a method is called again, then the old keys are overwritten and the new pair is saved under the same name.

The export of the RSA key to another destination needs the password, which was used to lock the keys in the first place. The key data is taken from a predetermined location and passed to the AES decryption method. There this data is read to the buffer, splitting the initialization vector from the data itself and then the hashed password is used to do the decryption – setting up the necessary structures, doing the verification of the decryption. The resulting data is given as a plaintext and then this data is passed back to the output console. In addition, the same data is passed to the hashing function, where the result is outputted to the console next to the public key itself.

The invitation creation, which is the call to the symmetric key export, is done by hashing the password used to lock the key in the first place and then the acquiring encrypted data with a file name and a predetermined location for the file. The key is acquired by using the AES decryption method: getting the data, the vector, decrypting the data, verifying it. After the symmetric key is acquired, the external public key is extracted in a similar fashion. The password used to lock this key is used as the key to do the decryption. It is hashed before passing it to the decryption. The external public key location is determined by the name given when calling the invitation creation function. When both keys are acquired – symmetric and public, the OpenSSL method for RSA encryption is called. This is the createRSA method. The method sets the acquired public key in the necessary structure to be able to use it in the RSA encryption. The structure is set up by calling BIO_new_mem_buf, BIO_set_flags, PEM_read_bio_RSA_PUBKEY. The acquired RSA key is passed to RSA_public_encrypt method. At the same time the symmetric key is passed for encryption and the setting RSA_PKCS1_PADDING is provided to make the RSA encryption use padding for the remaining part of data. The key, which is encrypted, is shorter than the maximum length allowed by RSA, so there will always be a part of data filled by the padding. The result from encryption is the length of the encrypted data. The data itself is returned as a pointer. The size of the pointer is set to 4096, which is a little bit more than the encrypted text can occupy. After the encryption, the encrypted data is passed to the base64 encoding method, which is also a part of OpenSSL library. The encodeb64mem method calls BIO_new, BIO_push, BIO_set_flags and BIO_write. After that the BIO_get_mem_data is called to store the length of the result and the result itself is stored in the output pointer. The result is returned to the output console. At the same time, the hashing of the result is carried out and the result of that is also given to the output console. The result of the hash is truncated to be more convenient for the user to read. The length of the hash is left at sixteen symbols.

The import of external symmetric keys is done by providing the password and the encrypted text to the decryption methods. At first, the encrypted data is run through the hashing method and the user is presented with a hash to verify that the received data is the correct one. The encrypted text is run through the base64 decoding to get a format, which can be used for decryption. The decoding is used by calling BIO_new_mem_buf, BIO_new, BIO_push, BIO_set_flags and BIO_read. After this, the private RSA key is extracted from the predetermined location. For the extraction, the password, which was used to encrypt, is provided, run through the hashing function and the result of this is pushed to the AES decryption method. The location for the data to be decrypted is already known, since the private key can be in only one location. After the private key is acquired, the private key structure is created to be used with OpenSSL methods. This is done by calling BIO_new_mem_buf, BIO_set_flags, PEM_read_bio_RSAPrivateKey. Finally, the decryption if the RSA is done by calling RSA_private_decrypt. The decrypted data is pushed to the AES encryption method, where a password to encrypt new key and the name of this key is provided. The key is saved following the encryption steps as with other situations. The result is saved in the symmetric key location.

The external public key import is done by providing this key in a console. The data is hashed and the user can compare, whether the values match with the intended result. After this, the acquired key data is passed to AES encryption along with a hash of a password used to lock this key. The key is given a unique name, when storing it and the initialization vector is generated along the way. The resulting encrypted data is stored in the predetermined folder.

The fingerprint check is done by providing the related data to the console. This data is run through the hashing algorithm and the result is given as a sixteen symbol output in the window.

The random bits generation for the keys and initialization vectors is done by calling OpenSSL method RAND_bytes and providing the length of the output, which is 32 or 16 characters.

The inputs from consoles are received in LPTSTR format, which is long string pointer. This data is converted to character array by using wcstombs method to copy data from one type to another. This applies to encrypted data, passwords, names for the data and choices from the list of existing values.

The checking is done for the empty values, where the method execution is halted. The check is carried out by comparing the first symbol with the empty value. At the same time, a check for duplicate names is carried out. In case of a match, the method is also cancelled.

The check for incorrect password is carried out during decryption stage. If the password is wrong, further methods are not carried out and every piece of data stored in the memory is removed. The user is given an error box, explaining the reason for cancellation of the methods.

5.1 Key management method layout

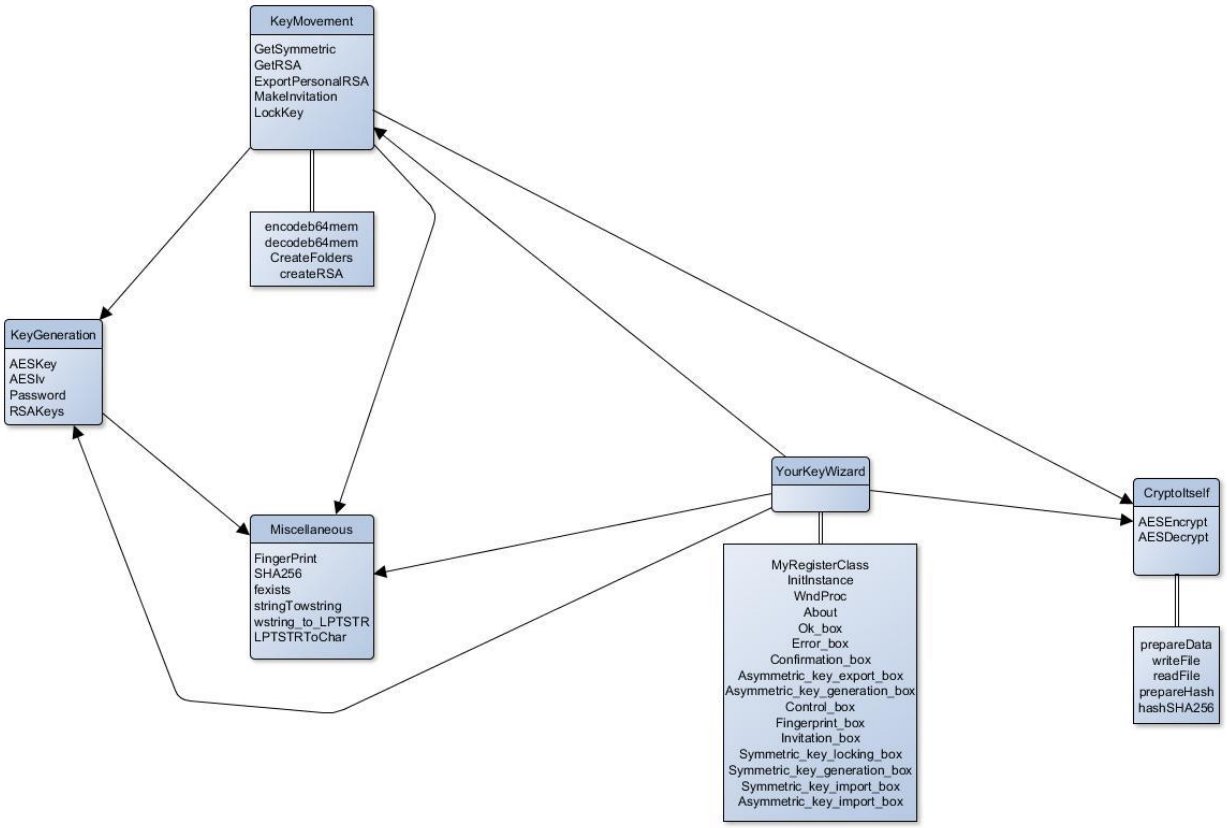

Figure 30. Class interaction diagram

The layout of classes is displayed in the table above. The dependencies are displayed in a way, where the arrow is shown to go from the class that uses another class to another class. There are in total five classes, which contain all the used methods. The classes are divided among public methods and private ones. They are displayed in top or bottom table respectively.

The main class is YourKeyWizard class, which contains all methods linked to the user interface windows. MyRegisterClass and InitInstance are used to set up the communication with the windows the user is interacting through. WndProc is used to start the the control window, where the user will be calling all the methods from. The WndProc itself is linked to menu editor, which also has the exit command. The main controls are linked to the Control_box.

Ok_box controls are linked to the task completion window. When the command is called, it takes no extra parameters and just displays the successfully completed task notification.

Error_box takes an extra parameter, which is the error text to display and the displays the error message to the user.

Confirmation_box takes no extra parameter, when called, but is an extra link before continuing with the tasks, such as creating asymmetric key pair.

Asymmetric_key_export_box is called with no extra parameters. When the show button is clicked, the method proceeds to take the password value from the input box, convert it into a character array and compare whether it is empty. Then methods for personal key export are set up and called. In case of empty password, the error box is called.

Asymmetric_key_generation_box is called with no extra parameters. When the generation button is clicked, the password is converted and compared to an empty value. If it's not empty, the key generation methods are set up and the key is generated. When the task is completed, the Ok_box is called.

Control_box is called, when the application is started. The user chooses a method to call and the controls are redirected from control_box. The symmetric key generation, invitation creation, symmetric key import, symmetric key unlocking, asymmetric key generation, asymmetric key import, asymmetric key export and fingerprint windows are called from control_box.

Fingerprint_box is called with no extra values. When the fingerprint button is clicked, the method checks whether the text is empty. If not, the methods needed for fingerprint creation are called and the result is outputted to an output box in the window.

Invitation_box is called with no extra values. When it is called, the method checks the state of the folder, where the keys are placed. If the check determines that there are no values in the location, the method is cancelled with an error message. This is carried out using FindFirstFile call for symmetric an asymmetric keys and the result is compared with an INVALID_HANDLE_VALUE. This is called twice. It checks the existence of the folder itself and then looks at the files in the folder. If keys are found, they are listed in a combobox for symmetric and asymmetric keys. When the generate button is clicked, the method compares the values from the password inputs and if they are empty, an error box is displayed. If the passwords are not empty, they are converted to characters, the necessary methods are set up and

called. These methods are an invitation method and a fingerprint generation method. If during the method execution, the passwords are incorrect, the user is given an error_box and the execution is cancelled.

Symmetric_key_locking_box is called with no extra values. When it is called, the existence of symmetric keys in the predetermined folder is assessed. If no keys are found, an error box is called and the method is cancelled. Otherwise, the method fills the values into a combo box. When the user clicks the allow button, the method checks whether the passwords are not empty. Then it sets up the methods for key locking and after everything is complete, the user is given an ok_box to confirm the success. If the password is incorrect during the method execution, the user is given an error_box and the method is cancelled. If the user clicks on the lock button, the method calls the key locking mehods, which remove the key from the chosen location.

Symmetric_key_generation_box is called with no extra values. When the generate button is clicked, the method checks, whether the password and name values are not empty. If they both have some values, the name is checked, whether it is not unique with any other file in the folder. Otherwise, an error box is displayed. Then the methods for key creation are set up and called and the user is given task completion box if everything goes successfully.

Symmetric_key_import_box is called with no extra values. When the method is called, the existence of private key is checked. If no key exists, the method is cancelled with an error box. When the user clicks on the import button, the method checks, whether the passwords, name and text box have any values. If not, an error box is displayed. Then the uniqueness of the name is checked. If there already exists' another file with the same name, an error box is displayed and the method is cancelled. Then the method sets up and calls the key import methods. If the password for asymmetric key is not unique, the user is given an error box and the execution of the method is cancelled. If everything goes successfully, the user is given the task complete box. On check fingerprint button click, the method checks', whether the text box is not empty and calls the method for fingerprint output.

Asymmetric_key_import_box is called with no extra values. When the user clicks on the import button, the method checks, whether the password, name and text box have any values. If not, an error box is displayed. Then the uniqueness of the name is checked. If there already exists' another file with the same name, an error box is displayed and the method is cancelled. Then the method sets up and calls the key import methods. If everything goes successfully, the user is given the task complete box. On

check fingerprint button click, the method checks', whether the text box is not empty and calls the method for fingerprint output.

The CryptoItself class contains the methods for the symmetric encryption and the related methods for the preparation. AESEncrypt and AESDecrypt are linked to other classes and the other methods are just internal.

AESEncrypt takes the key for encryption, initialization vector, data to encrypt and output location. The method itself performs symmetric encryption.

AESDecrypt takes the key for decryption, data for input or output, the input file location and the output file location. The method itself performs the symmetric decryption.

The prepareData method is used to get the data from the files or the buffer used for encryption and put it into the correctly sized buffers. If the data is read from the buffer, the size of the buffer is determined by providing the size value before calling the method. The calculations are done by checking the size of a string and then passing the value itself. If the method is called for encryption, the data that is encrypted, is appended with a test string before returning to the encryption method. The buffer size is increased by 11. If the method is called for decryption, the buffer size is as provided and the method calls the method for reading data from file – readFile. The return of the method is void.

writeFile method is used for writing the result of the symmetric encryption. The data passed to it is the output location, the data to write into the file and the initialization vector. The vector is written before the data itself. The method is used only by the encryption algorithm.

readFile method is used for reading data from a file and forming the buffer for data, which will be decrypted. It cannot be called for encryption. The method can also be called file size checking, which later will be used to set the buffer size. The file size is decreased by 16 to accommodate the size of a vector. The vector and the variables are pushed into different variables and pushed back to the prepareData method.

Miscellaneous class contains the methods for hashing, file existence checking and various data type conversions. Every class in the method is public.

Fingerprint method takes the text to hash and truncate and passes the data along to sha256 method. The return of the method is a string.

SHA256 method takes the text to hash, calls all the structures from OpenSSL library necessary for hashing and then proceeds with hashing. The method returns a string.

Fexists method takes the location of the file and checks, whether it exists at the given location. The return is a Boolean.

stringToWstring method takes a string value and converts it to wstring, which is the return value.

Wstring_to_LPTSTR method takes a wstring and converts it to LPTSTR, which is the return value.

LPTSTRTochar method takes LPTSTR data and converts It to char array, which is the return value.

KeyGeneration class contains the methods needed for symmetric and asymmetric key generation. It also has a method for preparing a password.

AESKey method takes the length of the key and uses OpenSSL random number generator to create the key, which is the return value.

AESIv method takes no values and uses OpenSSL random number generator to create a 16 symbol long value, which is the return value.

Password method takes the password coming from the yourkeywizard class and passes it through a SHA256 method to receive a 32 symbol long value, which is the return value.

RSAKeys method uses OpenSSL library to generate the private and public key. After the values have been created, they are passed back to the return variables.

KeyMovement class contains methods for the transferring the key from one state to another, which is for encryption, import and export.

GetSymmetric is a public method that takes the value of the symmetric password, asymmetric password, location of symmetric and asymmetric keys and the encoded text. In case it is used for symmetric key creation, the method calls symmetric encryption with the key, vector generation and the location, where to save. If the key is imported from a console, the base64 decode is called to get raw encrypted key by using OpenSSL library method. Then symmetric decryption method is called to get the stored asymmetric private key. The result is converted into OpenSSL private key structure. After that RSA decryption method is called with the text. Finally the decrypted data is saved with symmetric encryption method.

GetRSA method takes the password for storing the data. If the keys need to be created, then the KeyGeneration classes' method is called. The resulting keys are saved under the AESEncrypt methods, where each key gets a separate saving step.

ExportPersonalRSA method takes the password for decryption of the public key. The method sets up the structure for symmetric decryption and returns the data to the console box.

MakeInvitation method takes the locations of the symmetric and asymmetric key and their passwords. The structure for the keys is set up and AES decryption is run on both keys to get them in raw format. Then OpenSSL structures for RSA encryption are set up and the data is encrypted. The result is encoded in base64 for convenient output and this result is returned.

LockKey method takes the location of the key to lock and deletes this key from that location.

Encodeb64mem method is private and takes the text, which needs to be encoded. OpenSSL library is set up for the encode methods. The result is returned.

Decodeb64mem method is private and takes the text, which needs to be decoded. OpenSSLs' base64 decode method is set up for this. The result of this decoding is returned.

createRSA method is private and takes the key from RSA type. This key is converted to OpenSSL RSA key type and this result is returned to other methods.

createFolders method is private and is used to create all the missing folders for the key storage. It takes no values and checks the locations of folders know in advance.

6 Soonr link implementation

The next step is the encryption of the data, which has to be hidden from the external parties. The user provides a password for the locked key, chooses the key to use and then proceeds to decrypt the key. The password provided is hashed as in the case with the encryption key. The key name is used to find the exact file to decrypt. The path to the key is partly given in a predetermined location. The encrypted key is read from the file into a buffer, where the key data is split from the initialization vector. Since the length of each part of the data is known because of the way the code is written, the length of the key data depends on the fact that this is a decryption part. The encryption part would have another size given in advance as well. After this data is set up, the CryptAcquireContext is called with a PROV_RSA_AES variable to use the AES decryption. The key and the initialization vector are set by calling CryptImportKey and CryptImportKeyParam. The data to decrypt is copied to a buffer and passed to CryptDecrypt function in one call. If the decryption succeeds, the resulting data must contain the verification string, which is extracted from result and compared in the next step. If it matches, then it is assumed that the decryption is done correctly and the data is passed to further methods. This is done by copying it to the variables, which will be returned through pointers and then destroying the data in the temporary variables. The encryption of the data is done by passing the key to the AES encryption method, then passing the location of the data that needs to be encrypted. This data is a file, which will be read to a buffer. Other sources of data are not considered, such as console input. In addition to the key, an initialization vector is created by calling an OpenSSL method to generate random bits and passed to the encryption method. The data has the verification string added to it and the data becomes ready for encryption. The encryption method is set up by calling a CryptAcquireContext, CryptImportKey and CryptImportKeyParam. The data is encrypted by calling CryptEncrypt once with all the data in one go. The result is appended with the initialization vector and then the data is saved to a new file, which is then handled with further functions.

6.1 Soonr system method description

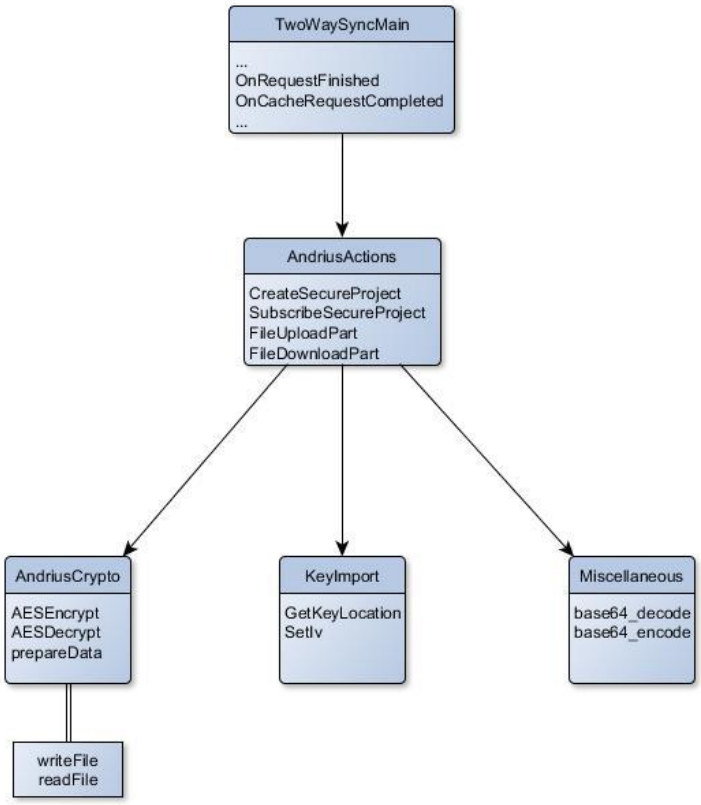

Figure 31. Soonr class interaction diagram

The soonr system TwoWaySyncMain class is used for the notifications of file download, upload, storing in cache and the errors that happen during any of the steps. In addition to that, the creation commands of new projects also passes through this class, so any code additions happen inside. This is a soonr class.

The method OnRequestFinished is called, when the file is successfully transferred to the cache location from server. The method has an internal call to another method by the name of OnDownloadRequestFinished, which handles the notification itself, but when the file has been fully pushed to cache from server, the controls go through this method, so the newly inserted statement goes inside the method. The method is used for projects, which have already been subscribed to and the state(private/secure or not) is already determined. Inside the method a request for testString is called when the file comes in. If this gives some value, then it is assumed that the file is from secure project and the encryption method FileDownloadPart is called. The return of the method is the size of the encrypted data along with pushes of older data back to the sync core(old size).

The method OnCacheRequestCompleted is called, when the file is copied from folder to cache location. The method has an internal call to OnUploadRequestFinished, but before that the method records the values of the uploaded data. That includes size, md5 values and timestamps. Before listing the values, a request for testString is called. If the testString is not empty, then the encryption method by the name of FileUploadPart is called. After it finishes, the method proceeds to notify the sync core to continue.

The method CreateProject is called, when a new project needs to be created. This method takes the encrypted string as an optional value and if the value is provided, the project is created as secure type. Otherwise it is created as a standard project with no extra encryption.

AndriusActions class is used for the encryption related methods. All the classes, which are added to the project, are accessed through this class.

CreateSecureProject method is used, when the Soonr client is trying to set up a new project, which has to be secure type. The method takes testString, password for the key and key location as variables. The method converts the password into byte type, sets up the link to the other classes used(described in later part) and then extracts the key from the location linked to the access management tool. After that, the test string is created and it is encrypted by using an AES algorithm with the key extracted. The resulting data is encoded to base64 to have a string output. The return is a Boolean value based on whether there were errors in the process.

SubscribeSecureProject method is used, when the client is synchronizing with an already existing encrypted project. The method takes testString, projectID and the password to the key. The password is converted to a byte array and the key from the access control management folder is extracted. The testString is decoded from base64 to regular string and then it is decrypted with the extracted key. The resulting data is compared with a predetermined testString and if they match, the key is considered to be valid. If they do not match, the password is considered incorrect and false is returned to the upper method.

FileUploadPart method is used for encrypting data and providing changed data values(size of the data). The method takes the values of the project ID, the path to the file that needs to be encrypted and the password. The key is decrypted using the same way as mentioned before and the data is encrypted using the extracted key and the file location. The encrypted data is saved in the same location as the original. If the method completes the steps successfully, a true value is returned.

FileDownloadPart method is used for decrypting the received files if the project is of secure type. The method takes the project ID, the location of the file and the password of the key. The key is extracted and the file is decrypted. The result of the file is placed in the same location as the encrypted file. The return is a Boolean value indicating, whether the method successfully completed the decryption.

MoveTheKey method is a continuation of the project creation. The method takes the location of the key and records it in the soonr location.

AndriusCrypto class is used for the encryption algorithms. This has the AES cryptography methods.

AESEncrypt method is used for the encryption of the files, which need to be uploaded and the test string. The method takes the key used for encryption and the initialization vector for the encryption. The data, which needs to be encrypted, can be provided from a buffer or as a file location. A sub-method is called to put the data in a buffer. Then a CryptAcquireContext, CryptImportKey, CryptSetKeyParam and CryptEncrypt is called to do the encryption itself. If the data was provided as a file name, the writeFile method is called to write the data to some location. If it was called from a buffer, then this is returned as a variable to the calling method. If everything goes well, the method returns a true to the method, which called.

AESDecrypt method is used for the decryption of the downloaded files, stored keys or test strings. It takes the key and the data as its variables. The data can be provided as a buffer or a file location. A sub-method is called to prepare the data for decryption. Then CryptAcquireContext, CryptImportKey, CryptSetKeyParam and CryptDecrypt is called to do the decryption. If the data is a regular file, then the result is written to a file. If the data was a key file, it is returned as a buffer. Before that, the key goes through test data check to see, if the key was correctly decrypted. If the data was a test string, it is returned as a buffer as well. The difference from key return is that, the verification data in the key file is removed from the copying to the buffer. The test string has no verification data to remove.

prepareData method is used to prepare the given information for encryption or decryption. The method takes the data from the buffer or file location. If the filename is provided, the method calls a sub-method to read the data from a file. If there is no filename and the method is called from encryption method, then the data is taken from a buffer. Last case with no filename and decryption, reads the data into the buffer and extracts the initialization vector.

readFile method is used for reading the data from a file to the buffer. The method takes the location of the data and the state, whether it's encryption or decryption. In case of encryption, the data is read as it is into the buffer. If the file will be decrypted, the vector is extracted separately from the rest of the data. Both elements are returned as variables.

writeFile method is used for writing the data to a file. The method takes the location for output, the data to write to the file and the type of method, which called – encryption or decryption. If the output is for encrypted data, the initialization vector is written before the rest of the data. Otherwise, only the data is written to the file.

KeyImport class contains the methods for saving the key location to Soonr folder and setting the initialization vector.

GetKeyLocation method is used to create a pointer in soonr folder. The file is used to point to a related key file after the project has been created or assigned. The method takes the location to key folder, the key location in the key management solution and the project ID. The method creates the folder for key pointer storage if it's not yet created. Then the file is created and the location of the key is stored inside. After that the same method is used to read the key location. The return is a string, which contains the key location.

SetIV method does not take any variables. It creates the initialization vector for AES encryption. The data is created randomly by calling Microsoft Crypto method – CryptGenRandom. The Microsoft implementation is chosen to provide compatibility with the rest of the Soonr software. The method returns the random data.

Miscellaneous class has the methods for data encoding and decoding. This is used for the test string transfer.

Base64_encode method takes the data that needs to be encoded and returns a base64 string.

Base64_decode method takes the encoded data and returns it as a string.

7 Evaluation

7.1 Performance of crypto in Soonr

In order to have a successful implementation of data encryption and decryption, the process must be completed in an acceptable time frame. This time frame is determined by personally chosen values. The following measurements were carried out by reading data, then encrypting it and finally storing it in the final location after which the upload process will begin. The time is measured in this interval, since the modifications and additions impact this segment only.

Size (MB)	Time (ms)
0.000977	0
1	16
10	47
98.6	812
507	4578

Table 8. Symmetric encryption speed

The data is measured for files measuring from one kilobyte to approximately 507 megabytes. The time taken to finish the encryption process providing various sizes indicates that this is a linear progression. Symmetric encryption design does not contain any increases or decreases in encryption time depending on the size, so this is considered to be “correct” implementation, correct meaning, that the appropriate order of the processes has been maintained, not that the chosen methods have the best performance values. The average sizes are difficult to determine because of various scenarios the tools are used in. The time for 507 megabyte file is 4.5 seconds and is considered to be acceptable.

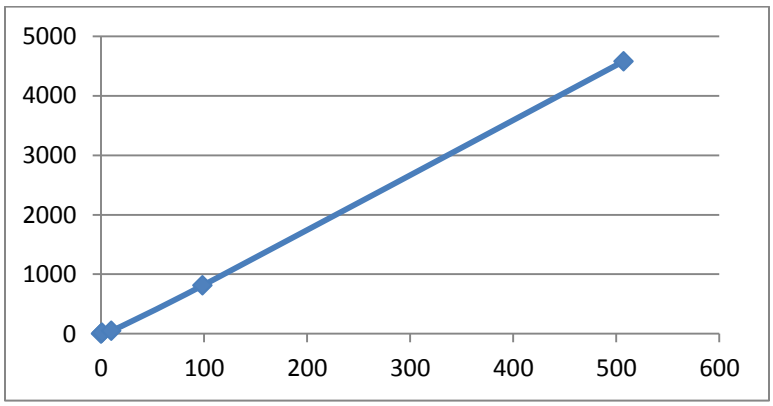

Figure 32. Symmetric encryption speed

In case of decryption, the following measurements were carried out by reading data, then decrypting it and finally storing it in the final location after which the transfer process to the final location will begin. The time is measured in this interval, since the modifications and additions impact this segment only.

Size (MB)	Time (ms)
0.000977	0
1	15
10	47
98.6	812
507	4328

Table 9. Symmetric decryption speed

The data is measured for files measuring from one kilobyte to approximately 507 megabytes as in the case of encryption. The time taken to finish the encryption process providing various sizes indicates that this is a linear progression as well. Symmetric decryption design does not contain any increases or decreases in time depending on the size, so this is considered to be “correct” implementation, correct meaning, that the appropriate order of the processes has been maintained, not that the chosen methods have the best performance values. The time for 507 megabyte file is 4.5 seconds and is considered to be acceptable. The values between encryption and decryption are comparable to each other, so the implementation is also considered to be “correct”.

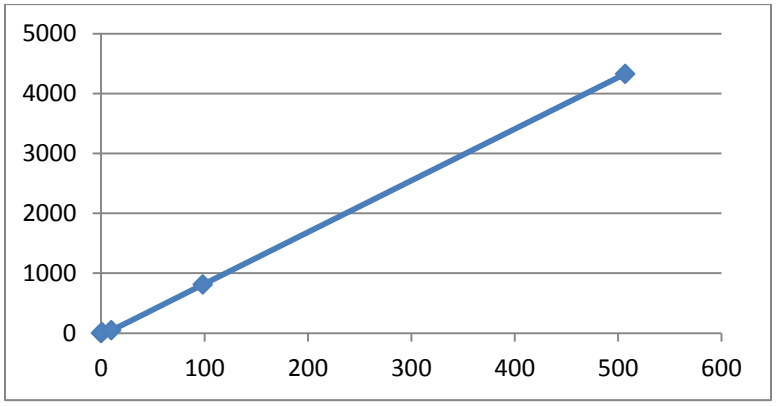

Figure 33. Symmetric decryption speed

The measurements for upload and download speed reflect the state of the network, not just the desktop solution. The following table displays upload time for files of various sizes. This is measured just for the transfer part and not for the preparations before transfer, such as moving to cache and

encrypting. The speed is displayed in kilobits rather than kilobytes to avoid conversion for comparison data used later.

Time (min)	Size (MB)	Speed (Kb/s)
0	0.000977	976
1.5	10	816
16	98.6	744
90	507	784

Table 10. Transfer time

The data for download is comparable to upload when the speed is the same range. In order to put this in perspective, average network speed is taken from “netindex.com”[30], which is 19 megabits per second and upload is 8.4 megabits per second for global average speed. The fastest case scenario is the download part. The speed of download in this scenario would be approximately :

Time (seconds)	Size (MB)	Speed (Kb/s)
0	0.000977	19456
3.7	10	
36.7	98.6	
217.6	507	

Table 11. Approximate transfer speed

Using global average speed data, it is possible to calculate the percentage of the time it takes to do encryption/decryption during the entire transfer to/from server from/to user device.

Time (%)	Size (MB)
1	10
2.2	98.6
2	507

Table 12. Time taken to encrypt/decrypt in relation to transfer time

The increase in transfer time is negligible and unnoticeable by the user, unless he/she is benchmarking the entire transfer.

8 Conclusion

The project consisted of analysis of the state of the art technology, problem identification, possible solution proposal and the implementation of the design following the proposed solution. Then the evaluation of the implementation was carried out. This included the flow of the controls and the transfer speed analysis.

8.1 Achieved results

The goal of the project to build an addition to the Soonr system, which offers a secure communication room, has been carried out.

The implementation of the model confirmed that it was possible to transfer the data management control from the server side to the user. The data storage location remains the same as initially, but the user becomes the only one capable of accessing the data. In addition to that, the user is provided with an extra layer of protection on the local device to lock out other people using the same device. The implementation maintains acceptable speed values, which do not increase the overhead drastically compared to pre-implementation state.

8.2 Future work

Since the initial design of the implementation proves the concept, further improvements to the design rely on making the implementation more convenient to the user. Additions to the key transportation and storage would increase the flexibility of the secure room by allowing more devices to use it. The design of the key management is linked to a single standard, which is not the only one used. Expansion to include more choices to the key would provide the user higher compatibility with more software solutions. Modifications to the implementation could allow users to use chatting or calling functions, while keeping the conversation hidden from the service provider.

9 Bibliography

1. Wen-Gong Shieh, Wen-Bing Horng . “Cryptanalysis on Sun-Yeh’s Password-Based Authentication and Key Distribution Protocols with Perfect Forward Secrecy” in “Computational Collective Intelligence”, pages 95-103. 2010

2. Hui-Feng Huang, Chin-Chen Chang. “A Novel Conference Key Distribution System with Re-keying Protocol” in “Web and Communication Technologies and Internet-Related Social Issues – HSI 2005”, pages 282-290. 2005

3. Youngjoo Cho, Changkyun Chi, Ilyong Chung . “An Efficient Conference Key Distribution System Based on Symmetric Balanced Incomplete Block Design” in “Computational Science and Its Applications -ICCSA 2004”, pages 647-654. 2004

4. Ananth Raghunathan, Gil Segev, Salil Vadhan. “Deterministic Public-Key Encryption for Adaptively Chosen Plaintext Distributions” in“Advances in Cryptology –EUROCRYPT 2013”, pages 93-110. 2013

5. Florian Mendel, Tomislav Nad, Martin Schlaffer .“Improving Local Collisions: New Attacks on Reduced SHA-256” in “Cryptology –EUROCRYPT 2013”, pages 262-278. 2013

6. “Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting” in “Cryptology – EUROCRYPT 2013”, pages 371-387. 2013

7. Adam Young, Moti Yung. “Malicious Cryptography Exposing Cryptovirology” , pages 66-68, 71-73, 76, 77, 332. 2013

8. Colin Boyd, Anish Mathuria. “Protocols for Authentication and Key Establishment”, pages 23-30, 38-40. 2003

9. Mihir Bellare, Phillip Rogaway. “Entity Authentication and Key Distribution”, 1993

10. Tadayoshi Kohno , Niels Ferguson, Bruce Schneier. “Cryptography engineering design principles and practical applications” 43-88, 195-212, 275-280, 2010

11. Anthony Harrington .“Cryptographic access control for a network file system”, 2001

12. M. Satyanarayanan. “A Study of File Sizes and Functional Lifetimes “, 1981

10 References

1 “This Brilliant Washing Machine Is a Roadmap for the Internet of Things”
Link: http://www.wired.com/2014/04/this-brilliant-internet-connected-washer-is-a-roadmap-for-the-internet-of-things/ [2014-06-17]

2 “HOUSEHOLD DOWNLOAD INDEX”
Link: http://www.netindex.com/download/allcountries/ [2014-06-17]

3“Edward Snowden”
Link: http://en.wikipedia.org/wiki/Edward_Snowden [2014-06-17]

4 “Dropbox says Security Breach Caused by Stolen Employee Password”
Link: http://www.thewhir.com/web-hosting-news/dropbox-says-security-breach-caused-by-stolen-employee-password [2014-06-17]

5 “Missed Alarms and 40 Million Stolen Credit Card Numbers: How Target Blew It” Link: http://www.businessweek.com/articles/2014-03-13/target-missed-alarms-in-epic-hack-of-credit-card-data [2014-06-17]

6 “Google tried to resist FBI requests for data, but the FBI took it anyway”
Link: http://venturebeat.com/2013/06/06/google-tried-to-resist-fbi-requests-for-data-but-the-fbi-took-it-anyway/ [2014-06-17]

7 “Microsoft challenges US gov over attempts to search overseas data”
Link:
http://www.theregister.co.uk/2014/06/11/microsoft_challenges_us_government_on_international_data_searches/ [2014-06-17]

8 “How we know the NSA had access to internal Google and Yahoo cloud data”
Link: http://www.washingtonpost.com/blogs/the-switch/wp/2013/11/04/how-we-know-the-nsa-had-access-to-internal-google-and-yahoo-cloud-data/ [2014-06-17]

9 “Glenn Greenwald: how the NSA tampers with US-made internet routers”
Link: http://www.theguardian.com/books/2014/may/12/glenn-greenwald-nsa-tampers-us-internet-routers-snowden [2014-06-17]

10 “NSA Bugs Can Spy on 100,000 Computers, Even When They're Offline”
Link: http://mashable.com/2014/01/15/nsa-hacked-computers-offline/ [2014-06-17]

11 “Exclusive: NSA infiltrated RSA security more deeply than thought - study”
Link: http://www.reuters.com/article/2014/03/31/us-usa-security-nsa-rsa-idUSBREA2U0TY20140331 [2014-06-17]

12 Anthony Harrington, Christian D. Jensen. “Cryptographic Access Control in a Distributed File System”, 2003
Link: https://www.cs.tcd.ie/publications/tech-reports/reports.03/TCD-CS-2003-28.pdf [2014-06-17]

13 “Dropbox Accused Of Misleading Customers On Security”
Link: http://www.darkreading.com/risk-management/dropbox-accused-of-misleading-customers-on-security/d/d-id/1097769? [2014-06-17]

14 “Dropbox users claim email addresses leaked to spammers, company blames 2012 security breach”
Link: http://www.theverge.com/2013/2/28/4041382/dropbox-users-claim-email-addresses-leaked-to-spammers [2014-06-17]

15 “How I compiled TrueCrypt 7.1a for Win32 and matched the official binaries”
Link: https://madiba.encs.concordia.ca/~x_decarn/truecrypt-binaries-analysis/ [2014-06-17]

16 “Lest We Remember: Cold Boot Attacks on Encryption Keys”
Link: https://www.usenix.org/legacy/event/sec08/tech/full_papers/halderman/halderman_html/ [2014-06-17]

17 “TrueCrypt encryption software 'not secure'”
Link: http://www.cbc.ca/news/technology/truecrypt-encryption-software-not-secure-1.2658354 [2014-06-17]

18 “Vodafone admits many governments have direct access to user data”
Link: http://www.theverge.com/2014/6/6/5785446/vodafone-secret-direct-access-wires-government-surveillance [2014-06-17]
19 “Block cipher mode of operation, Common modes”
Link: http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Common_modes [2014-06-17]
20 “Over 3 years later, “deleted” Facebook photos are still online”
Link: http://arstechnica.com/business/2012/02/nearly-3-years-later-deleted-facebook-photos-are-still-online/ [2014-06-17]
21 Bruce Schneier . “Security Pitfalls in Cryptography” in “Information Management & Computer Security”, 1998
Link: https://www.schneier.com/essay-028.html [2014-06-17]
22 Anthony Harrington, Christian D. Jensen. “Cryptographic Access Control in a Distributed File System”, 2003
Link: https://www.cs.tcd.ie/publications/tech-reports/reports.03/TCD-CS-2003-28.pdf [2014-06-17]
23 “Announcing the ADVANCED ENCRYPTION STANDARD (AES)” in “Federal Information Processing Standards Publication 197”, 2001
Link: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf [2014-06-17]
24 “Advanced Encryption Standard Crypto Speed”
Link: http://blog.jauu.net/advanced-encryption-standard-crypto-speed.html [2014-06-17]
25 “RSA key lengths”
Link: http://www.javamex.com/tutorials/cryptography/rsa_key_length.shtml [2014-06-17]
26 “How big an RSA key is considered secure today?”
Link: http://crypto.stackexchange.com/questions/1978/how-big-an-rsa-key-is-considered-secure-today [2014-06-17]
27 “Android bug batters Bitcoin wallets”
Link: http://www.theregister.co.uk/2013/08/12/android_bug_batters_bitcoin_wallets/ [2014-06-17]
28 Sam Trenholme. “Rijndael's key schedule”
Link: http://www.samiam.org/key-schedule.html [2014-06-17]
29 “AES Calculator”
Link: http://seit.unsw.adfa.edu.au/staff/sites/lpb/src/AEScalc/AEScalc.html [2014-06-17]
30 “Global broadband speed” Link: http://www.netindex.com/download/allcountries/ [2014-06-18]