CHALMERS

A Secure Cloud Based File Exchange System

The planning and implementation of a system to transfer files between users with a focus on security

Master of Science Thesis in Secure and Dependable Computer Systems

ERIK LARSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, September 2012

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.
The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

A Secure Cloud Based File Exchange System

The planning and implementation of a system to transfer files between users with a focus on security

© ERIK LARSSON, September 2012.

Examiner: Björn von Sydow

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden September 2012

Abstract

This thesis will present an approach on how to build a system to exchange files stored in the 'cloud', between two or more users in a secure way. At first different attack vectors will be presented to inform the reader why different security measures are taken throughout the system. The system will later be described in a theoretical aspect and different security aspects will be argued for. Different solutions will be proposed and discussed to achieve the best possible solution. The actual programming implementation will not be covered in details, although different guidelines on how to program in a secure way will be discussed.

At last the testing will covered. Both what can be done during the development phase but also different steps on how to test the final product. The thesis aim to show one way to build a cloud based web application to exchange data between users as secure as possible. The security will always be in focus, and other angles such as the overall user experience or time consumption for different solutions will not be considered in first hand.

Contents

1 Attack Vectors **9**
- 1.1 Cryptographic Attacks . . . 9
  - 1.1.1 Injection . . . 9
  - 1.1.2 Cross-Site Scripting . . . 10
  - 1.1.3 Cross-Site Request Forgery . . . 11
  - 1.1.4 Insecure Cryptographic Storage . . . 11

2 Cryptography **13**
- 2.1 Hash Algorithms . . . 13
  - 2.1.1 The SHA algorithm and SHA-256 . . . 14
- 2.2 Key stretching . . . 14
  - 2.2.1 Scrypt . . . 16
- 2.3 Encryption Algorithms . . . 16

3 System Description **18**
- 3.1 Authentication . . . 18
  - 3.1.1 Authentication Flow . . . 18
- 3.2 Data Transfers . . . 21
  - 3.2.1 Data Flow . . . 22
  - 3.2.2 Identity Based Encryption . . . 22
  - 3.2.3 Symmetric key Encryption . . . 23
  - 3.2.4 Authentication Models . . . 25
  - 3.2.5 System Description . . . 26
  - 3.2.6 CIA . . . 27
  - 3.2.7 Preserving CIAPAU . . . 30

4 Implementation **32**
- 4.1 Server-Side . . . 32
  - 4.1.1 Accessing the Database . . . 32
  - 4.1.2 Cryptography . . . 33
  - 4.1.3 Rendering the Response . . . 34
- 4.2 Client-Side . . . 34
  - 4.2.1 Javascript Client-side Implementation . . . 34

4.2.2 Data Transfers 35

5 System Testing **40**
5.1 Unit testing 40
5.2 Static Testing 40
5.3 Black-Box Testing 41
5.4 White-Box testing 41

6 Discussion and Conclusions **43**

A Appendix **48**
A.1 The Birthday Problem 48
A.2 Solving Captchas 48
A.3 OWASP Top Ten 50
A.4 Encodings 50

Acronyms

SSL Secure Socket Layer
NIST National Institute of Standards and Technology
SHA Secure Hash Algorithm
DES Data Encryption Algorithm
AES Advanced Encryption Standard
EES Escrowed Encryption Standard
MITM Man In The Middle
CBC Cipher-Block Chaining
XOR Exclusive OR
IV Initialization Vector
OWASP Open Web Application Security Project
XSS Cross-Site Scripting
CSRF Cross-Site Request Forgery
SQL Structured Query Language
IBE Identity Based Encryption
PGP Pretty Good Privacy
ORM Object-Relational Mapping
HQL Hibernate Query Language
API Application Programming Interface
UUID Universally Unique Identifier
RNG Random Number Generator
PRNG Pseudo Random Number Generator
JSP Java Server Pages
DOM Document Object Model
AJAX Asynchronous Javascript And XML
JVM Java Virtual Machine

Introduction

Transferring data between two or more parties without anyone else being able to intercept and read the data is a crucial and important task for almost every company in the world. For the company where this thesis was written it is crucial to send out reports to different clients without any third party being able to intercept the data.

Today, many companies will send such data using email attachments. These are however often size limited and many companies won't allow binary data to be passed as attachments. This leaves the user responsible for encrypting the data and upload it to a server somewhere. Encrypting files and distributing the key is however something that can be quite tricky and not something the average user wants to do. The result of this is that a lot of the material is being sent in an insecure way.

A lot of different companies provide solutions to send files between users in a way they usually call secure. Most of them will send the data over a secure link (SSL), but with the data itself being sent unencrypted. It will be encrypted first when it reaches the server. That however means the server is in possession of both the key and the encrypted data. Someone with access to the server can therefore retrieve and recover the data. This can either be by someone from the inside or a malicious user breaking in to the server.

This thesis will present a way to upload and store encrypted data in the cloud in such a way that only the sender and the intended receiver(s) can download and decrypt the data.

Requirements

There are a few requirements which the system must obey. This will limit the design choices to be made but also help to specify the system boundaries.

1. The server should be implemented in Java
2. The system should be a web based application, no software installations should be required
3. The application is to be hosted on a third-party server.
4. A MySQL database should be used
5. A custom authentication system should be implemented
6. The stored data should never be able to be decrypted by anyone but the sender and the receiver(s)

As a last requirement it will be decided how the system is supposed to be secure and for whom it is secure. The system will be hosted on a third-party server, controlled by another company. It can't be guaranteed that that company will not have access to the system. Even if the company itself does not have malicious intentions, a single employee could possibly disclose secret information. For the same reasons, why should a possible user trust this system or the system administrators to not disclose any of their data? This thesis will instead aim to design a system which does not rely on a trusted server. Even if the system were to be fully breached and all information on it were to be disclosed, it should still be infeasible to retrieve the original data. Since the server is not trusted, the system will instead lay over some of the responsibility to the users. The system should help out to make this task as small and easy as possible, but since something has to be trusted, it will be up to the sender to enforce this. If the user handles the information given, in a correct manner, a malicious user should not be able to disclose any sensitive information.

Chapter 1

Attack Vectors

1.1 Cryptographic Attacks

To better understand the terminology and counter measures which will be discussed later in this thesis, a few different attack vectors will be described. These are not all the different vectors but some of the most important ones as well as others that might be of peculiar interest to this project. The Open Web Application Security Project (OWASP) provides a top ten list of web application security risks[39]. The list does not cover all threats to take in to consideration, but provides a good starting point. For the full list, please see the appendix. In this chapter, only the following will be described in more detail.

- Injection
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- Insecure Cryptographic Storage

1.1.1 Injection

Injection is the number one threat according to OWASP. Although there are a few different types of injection, SQL injection is the main concern for this project. If an input field is connected to a SQL query without any filtering or validation, SQL injection is possible. A malicious user can use this to break off the SQL query by using an escape character. When broken off, it's possible to add custom logic to the query. As an example all rows in a table can be displayed or a user can be authenticated. The following is an example taken from the OWASP top ten[39];

```
String query = "SELECT * FROM accounts WHERE custID='"
+ request.getParameter("id") +"'";
```

```
http://example.com/app/accountView?id=' or '1'='1
```

The example above tries to retrieve all accounts for a certain customer ID. By escaping the query using an apostrophe(') it's possible to insert some logic that always will evaluate to true and therefore retrieve all accounts for all users. Using the above example the following query will end up being executed by SQL

```
String query = "SELECT * FROM accounts WHERE custID='' or '1'='1'";
```

This query will fetch every record in the accounts table instead of just the one corresponding to the correct customer ID which was originally intended.

1.1.2 Cross-Site Scripting

If injection is number one on the list, XSS is considered to be the second biggest threat. XSS is a method to inject client-side script onto a web page. The following snippet is taken from the OWASP top ten[39]

```
(String) page += "<input name='creditcard' type='TEXT'
 value='" + request.getParameter("CC") + "'>";
```

The CC parameter is then injected with the following data:

```
'><script>document.location= 'http://www.attacker.com/cgi-bin/cookie.cgi
?foo='+document.cookie</script>'
```

After the injection, the following code will execute:

```
<input name='creditcard' type='TEXT' value=''><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>''>
```

Once executed, the user's session cookie will be stolen and sent to the attacker's server. Cross-site scripting is usually divided into two different types; persistent and non-persistent.

Non-persistent

A non-persistent XSS occurs when the injected script is inserted and executed at the same time and thus never stored server-side. This is the more common of the two types. A user is usually directed to a web page containing the attack vector by clicking an innocent looking link. Clicking the link will cause the injected script to execute.

Persistent

A persistent XSS vector is stored on the server and executed every time the page is loaded unlike the previous type where a specific link needs to be clicked. When persistent, the injection is loaded from a database unlike non-persistent where it's loaded from something temporary like a request parameter.

1.1.3 Cross-Site Request Forgery

A CSRF attack is an attack vector where someone tricks the user into submitting a request to your website. Any website accessed by the user could do this. If the user is already authenticated there will be no way for the web-server to determine where the request is coming from. Unlike XSS where the trust a user has for a website is exploited, CSRF exploits the trust the server has for the user's browser. A CSRF attack will make requests to the web-server without having the user knowing the request was ever made.

1.1.4 Insecure Cryptographic Storage

OWASP only ranks this as number seven on their list but since this project relies heavily on encrypting data this vector will be described in more detail. There are many different angles to cryptography and a lot of things can be done incorrectly. Below are a few different things to take into consideration, the next section will then explain the cryptographic approach taken in this project in more detail.

1. Brute force attack. A brute force attack can occur when it's considered feasible to somehow try all possible combinations to either figure out the key, or if it's a hash function, the generated hash.

2. Collision attack. The fact that a hash is of a predefined size makes it possible to find two messages so that hash(m1) = hash(m2). All hash functions are vulnerable to a paradox called the birthday paradox. The birthday paradox is described in more detail in the appendix but a brief explanation is that given a set of people, the chances of two or more sharing a birthday is remarkably high even for a very small set. If there's more than 22 people in the set there's a 50% chance that two or more of them will have the same birthday. This is applicable to hash functions as well since a large enough set will have a large probability of containing two or more words that will generate the same hash.

3. Preimage Attack. Preimage attacks is a type of attack to find a message that has a specific hash value. It differs from a collision attack, which involves finding two arbitrary messages such as m1 $\neq$ m2 but hash(m1)

= hash(m2). Preimage attacks are usually split up into two different categories, first and second preimage attack.

- First preimage attack. Given a hash h, find a message m such that h = hash(m).
- Second preimage attack. Given a message m1, find a message m2 such that hash(m1) = hash(m2).

The ideal hash function will require $O(2^L)$, where L is the hash length in bits, attempts to do a successful preimage attack. This is a simple brute-force where all combinations are tried. If this is the complexity to do a preimage attack, the algorithm is said to be preimage resistant.

4. Dictionary Attack. A dictionary attack is a technique to guess a password using a set of likely possibilities. This can be done on both on-line systems as well as off-line password lists. The attacker will try to find the correct password by iterating a dictionary, one at a time for the targeted user. The knowledge that most people will choose a weak password, which can be guessed given a list of likely passwords is taken advantage of in this attack.

5. Man In The Middle Attack. Man In The Middle Attack or MITM, is a type of attack where the attacker eavesdrops on the communication between two parts. The different parts can communicate using any media that is possible to intercept. The two parts will always believe they are communicating with each other, not knowing it's the attacker that actually controls the information flow. The attacker can be both active and passive, passive meaning that the attacker merely eavesdrops without interacting. If active, the attacker will take an active part in the communication and change the messages being sent.

Chapter 2

Cryptography

This chapter will discuss the decisions made when choosing appropriate cryptographic functions for encrypting data and generating hashes.

2.1 Hash Algorithms

A few different hash algorithms were considered for the project. The hash function needs to be able to resist the different attacks described in the previous chapter.

As of today, NIST, which is the US department of standards and technology, approves five different hash algorithms[19].

1. SHA-1
2. SHA-224
3. SHA-256
4. SHA-384
5. SHA-512

The properties for the different algorithms are summarized in the table below. All numbers are the cryptographic strength in bits[1]

[1] The cryptographic strength in bits is a measurement on how many attempts are needed to break the hash value. 60 bits of strength equals 2^{60} attempts. 10 bits equals 2^{10} etc.

Algorithm	Collision Resistance Strength	Preimage Resistance Strength	Second Preimage Resistance Strength
SHA-1	less than 60	160	105-160
SHA-224	112	224	201-224
SHA-256	128	256	201-256
SHA-384	192	384	384
SHA-512	256	512	394-512

SHA-1 is considered to be broken since it is possible to find a collision in less than 2^{57} attempts[24]. NIST states that a hash function in use today should enforce at least a security of 112 bits[30]. All others can therefore be considered to be safe for usage. SHA-224 will provide exactly 112 bits of security, which is considered enough. This project will however use SHA-256. SHA-256 is widely used and the most commonly used in the SHA family. It gives enough security to be safe. SHA-384 and -512 would give more bits of security but would also increase the key size that must be stored and since SHA-256 is considered to be safe enough it will be used instead.

2.1.1 The SHA algorithm and SHA-256

SHA-256 takes an arbitrary input and outputs a 32 byte (256 bit) digest. There are no publicly known collision attacks on the algorithm and a preimage attack can therefore not be done in faster than 2^L attempts where L is the length in bits. Two passwords producing the same hash can be found in an average of $2^{(L/2)}$ attempts which is explained by the birthday paradox. This can also be seen in the table above under *Collision Resistance Strength* where the strength is always the output length divided by two.

2.2 Key stretching

Key stretching functions are a type of algorithms to make it more difficult for an attacker to retrieve a password when given a password file. SHA-256 is in the previous section described as a secure way to hash a string of text. Since it has more than 112 bits of security it is infeasible to attack the algorithm and retrieve the password. A real life system where the user is allowed to choose the password to be encrypted will however open up a new attack vector. In a recent attack on the social site Rockyou.com a study performed by data security firm Imperva showed that the most commonly used password was '123456' and the third most common was 'password'[34]. Real life users will often use passwords that are either easy to guess or related to the user in some way.

An attack where the attacker tries to guess a password using common password strings or known information about the user is known as a dic-

tionary attack. If a user picks a bad password which is easy to guess, a dictionary attack will be significantly more effective than the fastest known attacks on the SHA algorithms. Key stretching presents a method to make dictionary attacks less effective.

The idea behind key stretching was introduced by Robert Morris and Ken Thompson in 1974[37]. The term key stretching was however introduced in the paper 'Strengthening Password' by Abadi et al[25]. The idea is to build a function that deliberately calculates a password hash using a very slow method to do so. Robert Morris and Ken Thompson constructed an algorithm called CRYPT which was used to encrypt UNIX passwords using a key stretching algorithm.

Key stretching will limit the number of attempts possible to try on a hashed password. A dictionary used in a dictionary attack will have to contain less words and the chance to resist such an attack will increase. One way to measure key stretching functions is how expensive it would be to crack a password in one year or equivalently, how much the hardware would cost to crack a hash in a year. Note that these calculations are based on the fact that in average, only half the space needs to be looked at before finding a hit. The table below is from the specification of a popular key stretching function called scrypt which shows how expensive three different key stretching algorithms are, the higher numbers the better.[28].

Function	Running time	6 letters	8 letters	8 chars	10 chars
Running with a low cost (faster)					
PBKDF2 (1)	100 ms	< $1	< $1	$18k	$160M
bcrypt (1)	95 ms	< $1	$4	$130k	$1.2B
scrypt (1)	64 ms	< $1	$150	$4.8M	$43B
Running with a higher cost (slower)					
PBKDF2 (2)	5.0 s	< $1	$29	$920k	$8.3B
bcrypt (2)	3.0 s	< $1	$130	$4.3M	$39B
scrypt (2)	3.8 s	< $900	$610k	$19B	$175T

The table displays three different key stretching algorithms; PBKDF2, bcrypt and scrypt and the cost of breaking a hash using two different configurations as well as the running time of hashing a password using the specified key derivation algorithm. The cost is calculated from how much the hardware required to break one hash in one year would cost.

Based on this data, scrypt will be used within this project. PBKDF2 is slower which is a positive factor since it will increase the cost of a dictionary attack. However, scrypt requires a lot more memory to compute one single

hash and will therefore become significantly more expensive to break since much more hardware is needed. So when both execution time and the cost is considered, scrypt is the superior choice.

2.2.1 Scrypt

Scrypt works as a wrapper around the SHA algorithm as well as PBKDF2, which is another key stretching function. It is possible to configure how large the memory cost should be, as well as how parallel the algorithm should be. A salt must be passed in as well as the password to be hashed. A salt is typically a string or number used to further aggravate a dictionary attack. It is hashed along with the password and the attacker must use the correct salt along with the password to do a successful dictionary attack. The salt is typically stored alongside the password and will therefore not help during an off-line attack against the password file[31].

2.3 Encryption Algorithms

The project will need a symmetric-key encryption algorithm to encrypt and decrypt the data. The symmetric-key algorithm can be divided into two different parts:

- Block cipher. The block cipher is the actual encryption algorithm. The algorithm is called a block cipher algorithm since it encrypts one block at a time. The block size is determined by the algorithm. NIST currently acknowledges three different block cipher encryption algorithms[30]:

  - SkipJack
  - Triple DES (3DES)
  - Advanced Encryption Standard (AES)

  SkipJack is a cipher constructed by the National Security Agency (NSA) which is defined in FIPS 185[18]: 'Escrowed Encryption Standard (EES)'. It uses a 80-bit key to encrypt 64 bit data blocks. Due to the short key, this alternative was ruled out for this project. Triple DES applies the regular DES cipher algorithm three times. It uses a key size of 56, 112 or 168-bits. Due to a meet in the middle attack the security is reduced to 80-bits by NIST[30].

  AES, originally called the Rijndael, was selected by the Secretary of Commerce to supersede the older DES. It is specified in FIPS 197: 'Advanced Encryption Standard (AES)'[26]. It uses a key size of 128, 192 or 256 bits. AES is widely used and adopted by the US government. If configured properly, the cipher is considered to be infeasible to break.

There are some existing attacks, one, which will recover the key, is four times faster than brute-force. By using this attack, presented in the paper Biclique Cryptanalysis of the Full AES[22] the number of attempts will be reduced to $2^{126.1}$ which is still infeasible to brute-force. Given the widespread use and the strength of the algorithm, AES will be used in this project. All key sizes of it can be considered infeasible to break and this project will go ahead and use 128 bit keys.

- The block cipher mode. All above algorithms are block cipher algorithms, meaning that they encrypt one block at a time. AES-128 uses a block size of 128-bits. The different block cipher modes specifies different way of linking the blocks together. This project will take use of a block cipher mode called Cipher-block Chaining (CBC). The current block will in this mode be XOR:ed with the previous cipher text block before being encrypted.

This configuration will be used through out the system to encrypt and decrypt the data to be transferred. It's important to link the different blocks together, using a linking algorithm, such as CBC, if not, two blocks with the same content would result in the same cipher given the same key. By using the previous block to encrypt the new, they are all dependent of each other and it's therefore difficult to say anything about the unencrypted data given the cipher. Since the first block does not have a previous block, a random generated initialization vector (IV) is used. The IV is a random generated block, placed first in the cipher-text. The first data block uses the IV as previous block. The IV does not have to be secret, but hard to predict. It's also important to not re-use IV:s and instead always generate new ones.

Chapter 3

System Description

This chapter will go through and explain the full system in its different parts. First out will be the authentication flow, how users will authenticate themselves in order to upload data. Only the senders need to register an account, receivers can access their files without having to register an account first.

3.1 Authentication

This section will describe the authentication flow and how the users will identify themselves and access the system. Different attack vectors will be considered as well as arguments for why this is a secure approach.

3.1.1 Authentication Flow

Registering a New Account

A new user (Alice) will have to register for the system before it can be used. The proposed registration process contains the following steps:

1. Alice registers her email address, to which a link containing a unique validation token is sent.
2. Alice opens the validation link and can now input user information and a phone number.
3. A text message containing a random token is sent to Alice's phone.
4. Alice validates the phone number by entering the code and is now authenticated to the system.

The communication between the server, Alice and the required third party services looks as follows.

When Alice enters her email address, the server generates a unique random token long enough to not be brute forced. This token is saved server-side and sent to the specified email address. By doing this, Alice gets associated to an email account and the system will know that Alice can access the email account she specified. She will open the validation link which, on the server is associated with the email account. When opened, Alice is allowed to specify a password and some personal information. Alice will also be asked to enter a phone number, to which the system will send a unique six character token. When Alice enters the correct mobile phone code she is fully registered and can authenticate to the system.

The system can on its side associate Alice with both an email account and a phone number. The phone number is used along with the password to achieve *two factor authentication* which is explained under the login flow below.

Each authentication step requires a unique token which makes it impossible to bypass one step and still register an account. It is noticeable that the SMS code is only six characters long. Since it uses a mix between numbers (0-9) and letters (case sensitive) it will still be difficult to brute force. To make it even slower, a captcha is required to be solved after a few failed attempts. There is as of today no good method to solve a captcha automatically, a bit of research around this is shown in the appendix. If we assume 5 seconds per captcha and a total of 62 characters (26 capital, 26 small letters and 10 digits), a brute force would take around $6^{62} * 5 = 8.79726 * 10^{48}$ seconds or $\approx 10^{44}$ days which is clearly infeasible.

The registration flow will use SSL to prevent man in the middle attacks. The id, associated with the user, is a unique token rather than an integer to make it impossible to enumerate users throughout the system. One other way to map users could be to enter random email addresses at the first registration step and take note of the message given back by the server. If the message is different, it would be possible to figure out if the email is connected to a user or not. The system will therefore always respond with the same message whether or not the user already exist or if it is a new user that wants to register.

Login Flow

The proposed login flow between the client (Alice) and the server (Bob) is relatively simple and dependent on a proper implementation of SSL to be as secure as possible[17]. The SSL protocol is on a layer below the application layer, on which this project focuses and will therefore not be covered here. The communication between the client (Alice) and the server (Bob) looks as follows:

1. Alice sends her user-name and password to Bob.

2. Bob validates them and responds, if correct credentials are provided, by sending a text message to Alice's phone number.
3. Alice gives Bob the code retrieved from the text message and is now authenticated to the system.

By following this flow, *two factor authentication* is achieved. To avoid on-line dictionary attacks, Alice will be prompted to solve a captcha after a few failed login attempts. Once the SMS code has been used it's deleted and can never be used again. When trying to login, all passwords will be hashed even if it the user does not exist. Since scrypt is used to hash the password it takes a while to do, and it could therefore be possible to enumerate users based on how long it took for the server to respond.

Two Factor Authentication

Using only a password to authenticate to a system can be insecure for a number of reasons previously discussed. A way to increase security can be to use another factor besides password authentication. This concept is commonly known as *two factor authentication*. There are many different options to choose from and the term two factor authentication merely means two separate ways to access a system. A common approach is to have one factor which is something the user possesses and the other something the user knows[35]. One example could be a password and a token saved on the client computer. The password represents something you know and the token something you have. It's common for Swedish banks to require a hardware token generator as well as a pin code to access the Internet bank account. The hardware device represents something you have and the pin code something you know.

This thesis will use a solution where a phone is something the user has and a password is something the user knows. A separate hardware device would be to impractical to manufacture and a phone is therefore used instead. Almost everyone will have access to a cellphone and it's not unreasonable to require one to use the system. The user is forced to enter a phone number upon registration which will be verified using a text message. A text message containing a one-time code is then sent to the phone number every time the user wants to authenticate. A regular password represents something the user knows. Two-factor authentication protects the user in the sense that it's not enough for an attacker to know the password, but the attacker must also have access to the users cellphone.

It's important that the text message content never leaves the server other than when sent to the phone. If it is for example calculated at one server and sent from another, someone might be able to eavesdrop and retrieve the code without having access to the physical phone. It's also important to understand that two-factor authentication protects the user from password

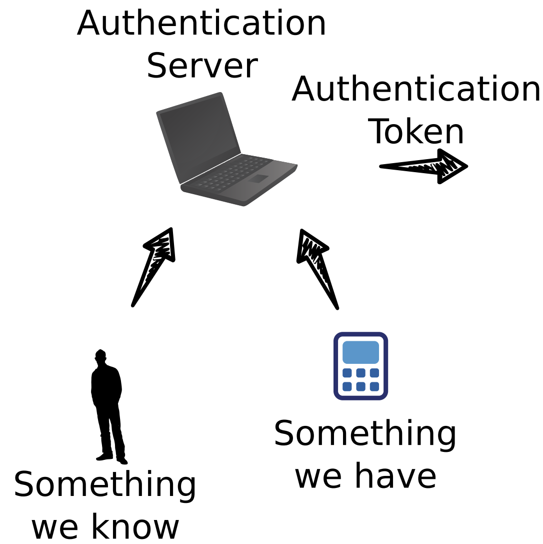

Figure 3.1: Two factor authentication; something we know, something we have.

stealing but does nothing for many other attack vectors like phishing and man-in-the-middle attacks[32].

3.2 Data Transfers

This chapter will discuss the core of the system, transferring data. Different possible approaches as well as the method taken in this thesis will be discussed. The data should be stored encrypted on a server hosted by the system but in a way so the server can never decrypt the data. The server should be responsible for coordinating the data transfers and give alternatives on how to deliver the decryption keys to recipients. Again, without ever knowing the actual key. The data flow is first described and discussed. Later, two different encryption schemes are considered. At last the complete system is described in more detail and a theoretical way on how to argue over the system security is presented.

Below is a very simple flow on how a transfer is made between a sender and a receiver:

1. The sender (Alice) authenticates to the system.

2. Alice creates a new package. A package consists of an authentication model (described further down) and a set of one or more files.

3. Alice notifies the receivers about the package and delivers a password to them.

4. A receiver (Bob) can now use the information from Alice to access the data. Bob does not have to be a registered user, everything needed to retrieve the package will be supplied by Alice.

5. Bob can download the data as soon as he accesses the system.

Each step is described in detail later in this chapter.

3.2.1 Data Flow

One of the main prerequisites is that the data should be shared only between the sender and the intended receiver. The server should store the data but never have enough information to be able to decrypt and access it. This entails that the data can't be encrypted server-side since that would give the server access to the raw unencrypted information. Even if the data is transmitted over an encrypted socket, it will still arrive to the server unencrypted and therefore readable. All data must therefore be encrypted on the client. This thesis will present a method where data can be stored on a non-trusted server and still provide methods on how to exchange keys in a secure way. The system should also be web-based and no additional software installations should be required.

Since the data is first encrypted on the client computer and then sent to the server, the server can never be allowed to see the decryption key in clear-text. The keys must be kept separate and can't be allowed to pass through the server. To understand the flow, the encryption schema must first be investigated. Two different approaches are explained below, Identity Based Encryption and a more traditional symmetric key approach. Both encryption schemes will rely on the encryption algorithms discussed earlier in the report.

3.2.2 Identity Based Encryption

Identity Based Encryption (IBE) is a public key encryption[1] scheme where the public key is derived from some unique information about the user, such as an email address. The concept was first introduced by Adi Shamir in 1984 [33]. He defined it as "A novel type of cryptographic scheme, which enables any pair of users to communicate securely and to verify each other's signatures without exchanging private or public keys, without keeping key directories, and without using the service of a third party.". The idea behind IBE is to take away the key-exchange part of the encryption flow. Instead something everyone knows (like the email address) could be used.

[1] Public key encryption is a system where one key is used to encrypt the data and another different one is required to decrypt. The encryption key can be known by everyone and is therefore referred to as the public key. Only the receiver can decrypt the data using another separate key, the private key.

One of the problems with IBE is described by Martin Luther in his paper "Identity-Based Encryption and Beyond"[23], which among other things discuss how to generate the private/public keys. To do this a separate Private Key Generator (PKG) must be used. The keys can't be generated client-side and must therefore be done so on a server. This project can't allow to have a trusted party like a PKG since it's important that no one except for the sender and intended receivers should be allowed to access the clear-text data. Shanqing Guo and Chunhua Zhang described, in 2008, a method to use IBE together with an untrusted PKG in their paper "Identity-based Broadcast Encryption Scheme with Untrusted PKG"[41]. Their protocol is however designed to be used with broadcast encryption which is not used in this thesis. It's also important that a schema like this is thoroughly tested by many parts before being considered secure. Such research is not available for this method and the problem with a trusted PKG will therefore still remain. This thesis will instead use a more traditional symmetric key encryption method.

3.2.3 Symmetric key Encryption

One of the requirements was that the receiver should not have to register to the system before receiving a package. That means no kind of public key encryption can be used, since there is no way of knowing the public key of an unregistered user. Instead, a symmetric key approach will be taken. To do so, PGP will be used. PGP follows the OpenPGP standard[36] originally presented by Paul Zimmerman in 1991. PGP usually uses a combination of hashing, symmetric key and public-key cryptography. This project will however use PGP for generating keys, symmetric encryption and compression. The standard does not require public keys to be used, although that is a common usage. There is, according to the OpenPGP standard, a few different ways to deliver the symmetric key to the receivers. They can either be passed together with the data, encrypted with the receivers public key. If there are multiple receivers, the session key will be encrypted once for every public-key. The symmetric key can also be kept from the data and be passed separately to each receiver. At last, the symmetric key can be derived by OpenPGP, given a secret that must be kept separate from the data. This secret can be a random generated password or something chosen by the user, as long as it's long enough to not be brute-forced. None of the two last approaches requires any public-key cryptography.

The sender should, in this project be able to send something to a non-registered receiver. As stated before, public keys can therefore not be used. This project will instead use the third alternative where a secret is given to OpenPGP and the session key is derived from that given secret. The symmetric key, which is normally derived randomly will instead be derived from other factors that are only partially random. Exactly how the secret is generated depends on the different authentication model used. Authentication

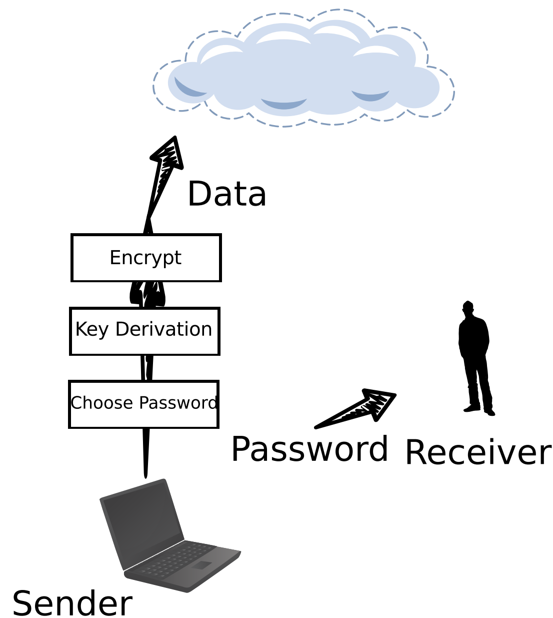

Figure 3.2: A simple figure of a proposed OpenPGP solution

models are described later in this chapter. For now, let's assume a random char sequence(the password) is provided along with a non-random token(the salt). The encryption flow of a file is then described as follows:

1. The password and the salt are passed through an hash algorithm. The output is the symmetric key.
2. A cryptographic checksum is calculated of the raw unencrypted file.
3. The file is encrypted using a symmetric key algorithm and the private key.
4. Another cryptographic checksum is calculated of the encrypted file.
5. The encrypted file is uploaded to the server.
6. The password is sent to the receiver.

It's important from a integrity perspective to calculate the checksum since it will verify that the raw file hasn't been modified while being transmitted or stored on the server. The password is generated randomly and must be long enough to make brute-forcing infeasible. As long as the server doesn't know the password it can't recreate the session key and therefore not access the data. It's noteworthy that the proposed schema is meant to be secure from a server perspective. The server should not be able to retrieve or modify any unencrypted data. From a complete system perspective, it's

difficult to argue that the schema is secure since it's up to the sender to deliver the password to the receiver. This is done outside the system by design. This thesis focuses on, as stated in the requirements on keeping the system safe from the server itself and it will therefore be assumed that the sender can deliver the password to the receiver outside of the system.

3.2.4 Authentication Models

An authentication model, or key exchange method, is a method to how the recipient authenticates to the system and derives the session key necessary to decrypt the data. The authentication model is chosen when the sender creates a new package. The password used to encrypt the data must always be kept away from the server and it will therefore be up to the user to deliver it to the receiver. To enhance the user experience, a few different authentication models will be presented.

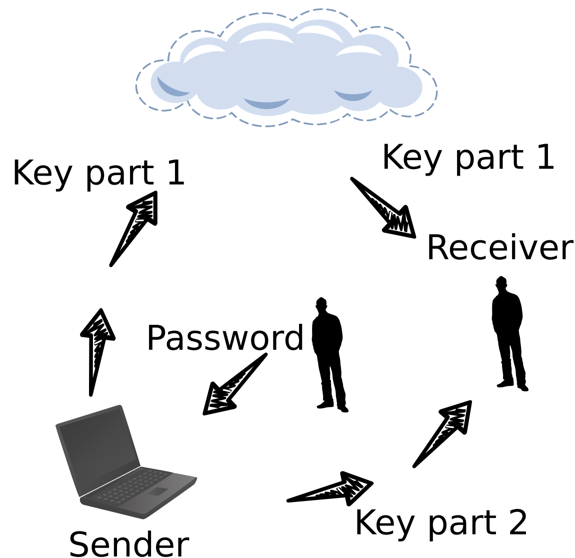

Figure 3.3: Proposed authentication model

In the proposed example the sender inputs a password, which is passed to the hash algorithm along with a salt from the server. From this a symmetric session-key is derived. The sender can then either send both parts (password and salt) to the receiver or let the server deliver the salt. Given this information a few different authentication models can be proposed.

1. Password and a random token (a token is a randomly generated string).
2. Random token and a SMS code.
3. Random token.

The list above shows three different approaches which all have in common that it's infeasible for the server to derive the symmetric key. In (1) The sender will have to deliver a password to each recipient without involving the server. The password is chosen by the sender when the package is created. This password must be long enough not to be brute-forced. The sender will only have to notify the recipients about the password, and not care about the token which will be delivered by the server. In (2) the sender will instead be responsible for sending out the token to all recipients. The token is generated on the client and never sent to the server. The system will instead choose a password which will be sent to each recipient's mobile phone whenever they try to access the data. In the last authentication model, only a token is used. The sender will be responsible for sending it to each recipient. Since only one part is used, something else must be used as a salt. This can be something related to the user, like a user id or an email address, as long as it's known by both sender and the system. In all three models, the server is always kept from knowing the password.

3.2.5 System Description

Based on the previous information in this chapter it is now possible to present the system in more detail.

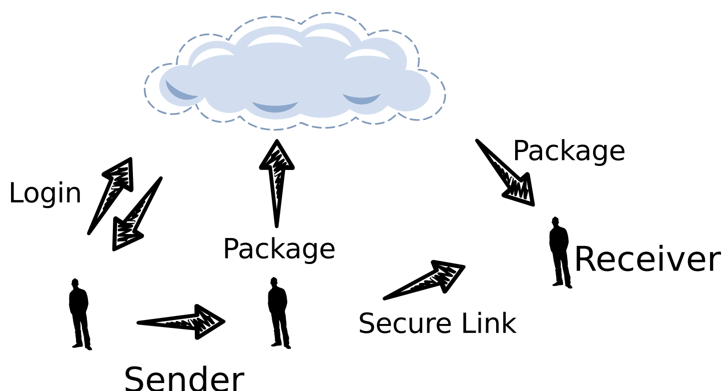

Figure 3.4: System Flow

As seen above, a user (Alice) will first authenticate server side. Alice will, upon success, be granted a token and an authentication cookie. She can now choose to upload a file. Once Alice choose to do so, a new package is created. A package contains of one or more files, a package code and an authentication model. Before the upload can start, the authentication information (cookie and token) and the package code is passed down to Alice. Before the encryption key can be derived, the authentication model must be set. If Alice picks a password based model, the key is derived from the password and a random generated key, otherwise it's derived from the

randomly generated key as well as something unique to the user, like the user-ID. Once this is all set up, Alice can select the files to be uploaded. Once Alice is done uploading, a URL, which is needed to download the package is generated. The URL will have the following format:

```
https://my-host-name/?packageCode=my-package-code/#derived-token
```

As described in the RFC3986 (Uniform Resource Identifier) standard, the information after a hash sign is meant to be kept client side. None of the major browsers will ever send information after the hash sign to the server. It's therefore an appropriate method to use to keep the password separated from the server and therefore disable it's decryption abilities while stile keep it contained in the URL. This link will now contain information about what package to access (using the package code) and one part of the secret needed to derive the encryption-key (the password, or in case of authentication model (1), the salt).

Alice can now distribute either the password or the link to the intended receivers. For authentication model 2 and 3 Alice will have to send the link herself, for authentication model 1, the system can send it since Alice will distribute a password of her choice to the receivers instead. The receiver (Bob) can point his browser to the received link and access the data. If the package was created using a secure link only, he can download the data instantly. If a password is needed he will need to input that first.

As seen above the server passes down the data together with a checksum and the signature. The decryption key can now be derived based on the input given by Bob. The signature and checksum is used to make sure the data is preserved unmodified. Bob can also ask Alice for the original checksum to make sure the server haven't tampered with the files.

3.2.6 CIA

CIA is an abbreviation for Confidentiality Integrity and Availability which is three important principles of information security[40][38]. Based on the principles it's possible to argue and discuss the system security in a more theoretical way. For many years there has been discussions about extending the triad and add on more principles. One proposal was made by Donn B Parker in his article *A Note on Our Terrible Security Model*[27] where Possession, Authenticity and Utility are added on. Below is a brief explanation of the six principles.

Confidentiality

The purpose of confidentiality is to protect information from disclosure to someone not authorized to see the information. The leakage can be both accidental and deliberate. Consider an internet credit card transaction where

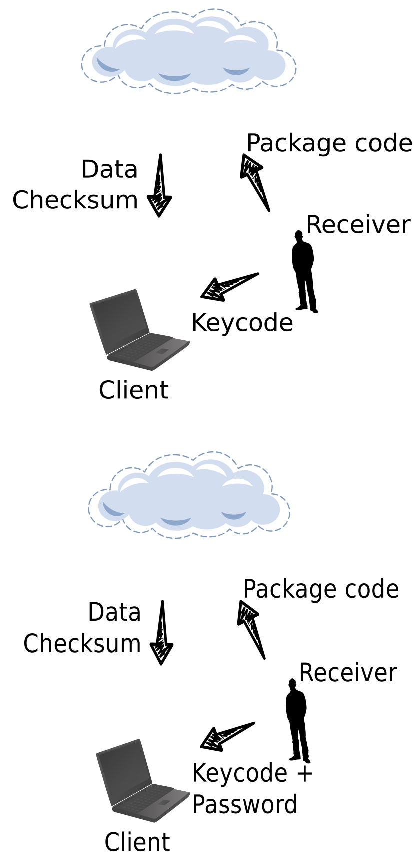

Figure 3.5: Top: Receive flow using secure link only. Bottom: Receive flow using password and a secure link

the credit card information is transferred from the buyer via the on-line store to the credit card company. No one but the three parts should be allowed to see the information. If that were to happen, there's a breach in confidentiality. The actual breach isn't necessarily for someone to somehow penetrate the system, but could also be for example to look over the users shoulder and retrieve the credentials. Confidentiality is usually achieved using any of the following two mechanisms[40]

- *Access control mechanisms* where illegitimate users is prevented access to the data.

- *Encipherment mechanisms* where the data is encrypted so not everyone can understand it.

Integrity

The purpose of integrity is to make sure the information stays untouched and can't be modified without any notice. Integrity is violated when a message is modified in transit. In the previous example, it could be to modify the credit card number when being transferred. Where confidentiality addresses that the information might be known, integrity addresses that the information might be changed. Integrity protection could be both to prevent the information from being changed, but also to detect when the information has been modified.

Availability

Availability means to provide the actual information within a specific time frame. The time frame may vary between different applications but the information should always be available whenever it's needed. In general there are three dimensions to the availability principle;

- *Hardware availability* which is the ability of physical computers to be available when needed. It's primarily achieved using redundancy and fault tolerance.

- *Application availability* is the ability of an application to survive a failure. This could for example be achieved by distribution or some recovery process.

- *Data availability* is the ability to access the database and retrieve data even after a hardware or softare failure. This can for example be achieved through some recovery technique.

Possession

Possession or control is not part of the original CIA triad. A breach in possession is when a thief is in control over some confidential information but wont necessarily use it. The rightful owner will still be concerned that the thief could do so at any time without the owner knowing so. Possession tries to prevent physical contact of the data. If for example a thief gets in control over a list of passwords, even if the thief wouldn't look at the passwords this would still be a breach of the possession principle. If the thief looks at the stolen information, it would be a breach in confidentiality as well. Possession and confidentiality is in that sense closely related but while possession merely means to have access to the information, confidentiality is breached when the intruder actually looks at the stolen data[21].

Authenticity

Authenticity means to avoid fraud, meaning to verify the origin or authorship of the information. A breach could for example be to change the origin email address to someone else in an email correspondence[21]. One could for example look at the handwriting characteristics of a handwritten document and sample that to other documents with a known origin and verify the document. In computer security, signatures are often used to prove authenticity.

Utility

Utility is a principle of proving usefulness. A breach of utility could for example be to encrypt information with a private key and then loose the key so the original data could never be retrieved. The data could possibly fulfill all other principles but wouldn't be useful if the unencrypted information can't be retrieved. Utility is related to availability but differs in the sense that availability looks at delivering data within a time frame and utility to the delivered data being useful.

3.2.7 Preserving CIAPAU

Based on CIA or CIAPAU, containing all six variables it's now possible to evaluate the system in a more theoretical way. Confidentiality is mainly achieved by the package and key code. The package code is long enough for no one to enumerate or guess which means only the people knowing the code will have access to the data. If, for some reason, the package code leaked or if someone managed to get over the data stored at the server it is still encrypted and therefore still not useful. The encryption algorithms are considered to be secure and it's therefore considered infeasible to retrieve the raw information. Integrity is achieved by the checksum. By calculating a cryptographic checksum before encrypting the data and doing the same when the receiver has downloaded and decrypted the package, equal checksums is a guarantee that it is the original data.

Availability is mostly preserved on a layer lower than the main scope of this project and is therefore not discussed in depth. The system will however be deployed on a third party server. The third party is a huge multinational company and guarantees the system to be available.

Possession could in this project be breached in mainly two ways.

- Someone would retrieve the data by accessing the server.
- Someone would manage to download the data by having the correct package code.

The first is again, not really part of the project and will not be discussed. The second is more interesting. The system relies on the package code to be

unique and long enough not to be brute-forced or guessed. In the presented solution, the server can validate the package code but never see the keycode. This presents a problem, since everyone with a valid package-code will be able to download the package. This is not a good solution since only people that could actually decrypt the data should be allowed to download it. This is solved by passing along a hash of the keycode and the package code when authenticating to download a package. The same hash is stored on the server when creating the package. The hash will not tell anything about the keycode but it's a good way to make sure only people with the correct keycode can download the files.

Since the delivering of keys is mainly done outside the scope of the system, it will be up to the package sender to prove his or her authenticity.

In the first draft of the project, the secure link was only exposed once, when the package was finished. This made it crucial to save the link and never loose it since that was the only key to access the package. This was considered a utility breach since it was just to easy to loose the keys. In the second draft, the key codes are stored locally on the users computer by the system. If the key is lost, the user can if on the same computer retrieve the key again. This was deemed necessary to fulfill the utility principle for the system.

Chapter 4

Implementation

4.1 Server-Side

One of the requirements was that the server should be implemented using Java. Java offers a solution to building web applications called Servlets. A servlet is basically the applications face towards the web browser. It serves the client request and receives a response from the server. All servlets must implement two functions;

```
void doGet(HttpServletRequest req, HttpServletResponse res);
void doPost(HttpServletRequest req, HttpServletResponse res);
```

Just as the name explains, the first function receives all GET requests while the second handles POST requests. The servlet can, when done handling the request, generate a HTML page by creating a request dispatcher which takes a HTML file as input. The servlet can set cookies, content type of the response and more. This project works mostly with HTML and JSON[10] as content types. JSON is a lightweight data-interchange format which can be used together with a client side language like javascript to send data between the client and the server without having to reload the web-page.

Data can therefore be passed to the server using either JSON requests or regular POST/GET requests. In whichever way the data is sent to the server it must always be validated and inspected for injection and XSS before used. Especially if the sent data is used by the database.

4.1.1 Accessing the Database

The project will use a MySQL database to store user information and packages. To abstract away from writing SQL commands and to be able to use a more object oriented approach it was decided to use a Object-Relational Mapping (ORM). One of the more widely used for Java is called Hibernate[9].

Because of the popularity and widespread use, it was decided to be used in this project. One major feature of Hibernate is that it controls and escapes the input to avoid SQL injection. The big software security community OWASP also acknowledges Hibernate as a good library to avoid SQL injection[15]. It's important to note that Hibernate allows users to pass in native SQL queries, using a technique called Hibernate Query Language (HQL). These queries would still be vulnerable to injection attacks, and will therefore be avoided in this project. Only relational mapping is used in this project to avoid the risk of SQL injection.

To further avoid the risk of making mistakes, the classes used to access the database is never exposed to the servlet classes but must pass through a filter first which will check the content. All tables that are in some way exposed to the users must be constructed in a way so enumeration attacks won't be possible. This is accomplished by using non-enumerable ID:s. Instead of using a number starting at, for example 0, each row is assigned a Universally Unique Identifier (UUID). A UUID is a 16 byte number which is represented by 32 hexadecimal digits. It's usually displayed as follows.

```
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
Example:
63d27a52-33ac-2f5d-452c-c23fa5243600
```

A UUID is considered infeasible to brute force and it would therefore also be infeasible to enumerate the table. Java has built in support to generate UUID:s, so no external API is necessary.

4.1.2 Cryptography

It was decided to use a third party library for all cryptographic functions. Implementing cryptographic functions is a difficult and time consuming task and something that should not be done if not absolutely necessary. The server is responsible for hashing passwords used during the login flow. For this, Scrypt is used[28]. It's also important that only secure random generators are used, never the standard Java implementation. SecureRandom is considered a strong Random Number Generator(RNG). This differs from Random, which is a Pseudo Random Number Generator (PRNG). SecureRandom complies with the statistical random number generators specified by NIST in the paper *Security Requirements for Cryptographic Modules*[29]. The requirements for a strong RNG is specified by RFC1750, Randomness Recommendations for Security[20]. Among other things, the seed must be unpredictable, and the output must be cryptographically strong.

4.1.3 Rendering the Response

Once all computations are done it's time to generate the client-side response. Since a web system is being built, the main part of the response is HTML and Javascript. Javascript will be discussed in the next part as part of the client side. The documents are however generated server-side and there's a few important things to think about when doing so. Javas solution to render web pages is called Java Server Pages (JSP).They contain a mix of HTML, Java objects and logic. Logic and Java objects are accessed using, what JSP calls tags. The standard library contains among other things tags to print data, create logical statements and different ways to iterate data. To avoid the risk of Cross-Site-Scripting (XSS), the tags for printing out data are never used. Instead a custom library was designed to print data to the HTML documents. The following tags are exposed;

```
encodeHTMLAttribute
encodeJavascript
encodeURL
encodeXMLAttribute
encodeXML
encodeHTML
encodeHTMLAttribute
encodeJavascript
```

The encoding is different depending on in what context the data is used. Illegal HTML characters are for exampled embedded between '&#' and ';' while illegal Javascript characters are preceded by '\\u' or '\\x'. For a full list of encodings, please see Appendix. No text should ever be rendered to the client response without first being encoded.

4.2 Client-Side

The client-side implementation is divided into two parts, general Javascript code and the file transferring part. Javascript is used alongside HTML and provides a way to use logic combined with the HTML markup. It makes the client side more complex and opens up the possibility to render programming code and not only displaying HTML. There exist plenty of different Javascript libraries to simplify the programming. A very well known and widely used is JQuery[3] which is used in this project.

4.2.1 Javascript Client-side Implementation

Javascript is a prototype based scripting language. It's weakly typed and it's possible to combine a functional programming approach with a more object oriented one. Javascript is standardized in the ECMA-262 standard[16].

While first introduced in Netscape 2.0 by Brendan Eich it is now supported by all major browsers. The standardization process started in 1996 and the first edition was accepted in 1997. Javascript provides methods to dynamically modify the Document Object Model (DOM) of the page. DOM is a convention to represent and interact with objects in HTML (and also in XHTML and XML). New content can be loaded to the DOM depending on user input or content retrieved from the web-server. This makes it possible to have the content of the webpage change dynamically without performing a reload. Javascript will also make it possible to validate user input before sending it to the server. Even so, it's important to remember that the client always controls what's sent to the server and any javascript validation can always be side stepped. All input must therefore be validated on the server as well, javascript can however enhance the user-experience.

JQuery provides a library to simplify JavaScript development. The following quote is from JQuery's web page: *jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.* AJAX (short for Asynchronous JavaScript and XML) is used to create asynchronous web applications i.e communicating with the server without reloading the web page. JQuery exposes different functions to modify the DOM structure of the web page. When using these it's important to be careful to not be exposed to XSS injections. When using functions that takes HTML markup as a parameter, the code must always be encoded. To do this, this project will use an algorithm from OWASP ESAPI project[1] which is a project to make security functions available easily. The algorithm encodes HTML and Javascript and will therefore prevent XSS injections when used.

JQuery is also used to communicate with the server using AJAX requests. An important policy for JavaScript is that it's only possible to communicate with the same server as it originated from. This policy is called Same-Origin-Policy. It's a fundamental policy which prevents a user from leaking information if a malicious user managed to inject code to the page using XSS.

4.2.2 Data Transfers

The transferring needs to be handled in such a way that no additional software installations are needed. This basically means that there are two approaches to take. Either accomplish it using only javascript or to build a Java Applet which will handle the file transfers. Both approaches will be considered.

Table 4.1: Browser statistics from different sources, May 2012

Browser	W3C[13]	StatCounter[2]	NetApplications[4]
Internet Explorer	18.1%	32.1%	54.1%
Mozilla Firefox	35.2%	25.6%	19.7%
Google Chrome	39.3%	32.4%	19.6%
Apple Safari	4.3%	7.1%	4.6%
Opera	2.2%	1.8%	1.6%

The Javascript Approach

It is possible to encrypt data using PGP in JavaScript. A third party library called OpenPGPJS[7] accomplish this using nothing but JavasScript. It's open source and can therefore be reviewed and validated. One problem is how to access the file system. The user starts the upload flow by selecting a file. Access properties are required on that file since it must be encrypted. It's not possible to interact with the file system in either Javascript or traditional HTML. HTML 5 will however come with that capability in a new feature called the FileSystem API. The HTML 5 standard is however not yet fully developed and still changing. The support for different features also varies a lot between browsers. As of this project, Internet Explorer lacks a lot of the HTML 5 capabilities and the FileSystem API is only supported by Google Chrome. As seen in table 3.1 the market shares for different browsers vary depending on the source, it is however easy to notice that a feature only supported by Google Chrome, covering 20-39% of the market depending on the source, is not feasible for a web application. One other drawback is that it is not possible to somehow validate that the javascript downloaded to the client is in fact the same script the developer put on the server. Since the server is not trusted, the scripts provided by the server should not be trusted. An applet can be signed, and the the Java Virtual Machine will then validate the signature before execution. The same is not possible to achieve in JavaScript. The JavaScript solution was because of these reasons ruled out from this project.

The Java Applet Approach

Java Applets is a technology to execute Java code in a web based system. It was first introduced in 1995[11] by Sun Micro Systems (now Oracle

Corporation)[11]. Since applets are launched inside the browser but executed by the Java Virtual Machine (JVM) they are browser independent. Although JVM versions may differ between operating systems as well as between users, cross browser issues are still limited. Since Applets are built using regular java code, there are plenty of libraries to use for cryptography. Java's standard functions to generate secure random numbers and send data to a server can also be used.

Applets are usually deployed using Java Archive (JAR) files which is how Java Applications are usually deployed. Java supports signature validation of jar files which can be used to prove authenticity of the jar file and prove that the code hasn't been tampered with. Based on this, Java is a perfectly valid solution from a security perspective and it was decided to be used in this project.

Applet Architecture

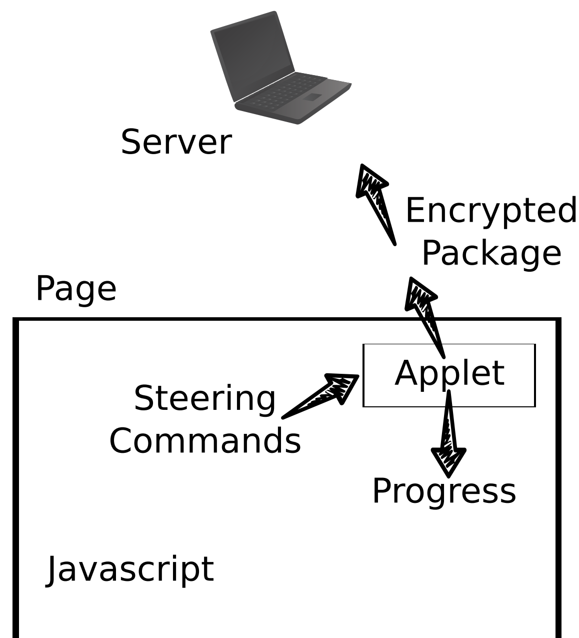

Figure 4.1: The communication between the server, the javascript code and the applet

From a user perspective it was decided to have as much of the UI as possible done outside of the applet and instead achieved by HTML and JavaScript. Public methods, defined in the Applet base class can be accessed from JavaScript. By doing so, most of the user interaction can be abstracted away from the applet and the rather slow user interfaces in Java. The only UI originating from the applet will be a dialog prompting the user to select

files on the file system. The download flow will work in a similar way where the file will be downloaded and decrypted to a location chosen by the user, again using a Java prompt. The different applet states will be reported to the main page using JavaScript method calls so the user can be updated on where the applet is in the process of encrypting/decrypting a file in addition to different upload states.

Deploying the Applet

One of the reasons behind using an applet was the support for code signing. Applets can be signed and the authenticity is validated by the JVM before the applet is launched. Besides the possibility to prove authenticity, it will unfortunately introduce another security vulnerability. Since applets are executed just by loading a web page they are usually executed within a sandboxed context without certain privileges which usual Java desktop applications usually have. However a signed applet will force the JVM to prompt the user to accept the certificate. If the user chooses to trust the certificate the applet will run outside of the sandbox with unrestricted privileges, just like a regular Java application. There is currently no way to enforce a signed applet to run in a sandboxed context unless the user chooses to not accept the signature.

This presents an issue to whether it is better to not sign an applet and have it execute in a sandbox or to sign it and thus prove the authenticity but having it executed with unrestricted privileges. The user experience will also be different since the user will have to accept the certificate if the applet is signed, an unsigned applet will launch silently without any warnings. The applet built for this project will work even if executed in the sandbox and there is really no reason to have it executed outside the scope of the sandbox. It might even be worse since a prompt telling the user that this applet will run with unrestricted access to the computer might impact or scare the user.

This project will however still use a signed applet. Mainly because of the ability to prove the authenticity. If a precautious user denies the certificate, forcing the applet to be sandboxed, it will still work as intended. It is however unfortunate that it's not possible to execute a trusted applet in the same context as an unsigned would.

Implementation Guidelines

The applet will mainly be responsible for three things.

1. Key derivation.
2. Encrypt/decrypt data.
3. Deliver the data to the server.

Similar to on the server, third party libraries will be used to handle cryptography. BouncyCastle[8] is a popular open source project to provide PGP encryption for Java and C#. It is a respected, well tested library and will be used to handle the encryption and decryption in this project. The key derivation is done using Keyczar[14], which is developed by the Google Security Team. It is built to provide key derivation in a simple way. Even if developed by Google it is open source and everyone can review the source code.

Since the applet is communicating using JavaScript calls, it is important to encode all information passed out from the applet. While developing the applet, a vulnerability where a malicious user could inject code by choosing a bad filename was discovered. The injected code would be executed whenever the filename was read.

The files are uploaded using a multipart/form-data. This is accomplished by standard Java functions, no third party libraries are necessary.

Chapter 5

System Testing

This chapter will focus on testing the system and how to reach and contain a certain code quality. Code principles and how to assure them will also be discussed. Methods used during the development will be in focus first and later methods to test the full system.

5.1 Unit testing

Unit testing is a method to test different modules individually while developing the code. In many Test Driven Development (TDD) approaches the unit tests are actually created before the system and the system is considered complete once all unit test are accepted. In this project, the tests were however constructed alongside the system development. Unit tests aim to break up the system into smaller modules and test those individually. In this project, unit testing is used as much as possible during the development. When different security breaches are found during other types of testing, unit tests can be developed to make sure the same attack vector won't open up again in future releases.

5.2 Static Testing

Static testing is used to investigate the raw source code before it's compiled. It can be used during the development to make sure certain guidelines are followed and that different security principles are not violated. In this project a tool called PMD[6] is used. PMD is mainly used to find dead code, overcomplicated expressions or duplicated code. This project will instead use a PMD plugin to test different security principles. Among others the following is tested using PMD.

1. No unvalidated data is passed in to the database.

2. All input parameters must be passed through a filter before reaching the database classes.
3. All data printed out in JSTL tags must be written out using an encoded library. The standard tag *c:out* should never be used.
4. No unencoded JavaScript parameters should ever be passed from the applet to the client-side JavaScript code.

These tests will mainly help to detect SQL injection and cross-site scripting from a code perspective. SQL injection should already be prevented by using Hibernate to access the database. The tool will however provide awareness and make the project less dependent on a third party library. Even if SQL injection is prevented, the system can still be vulnerable to XSS. The first and second item make sure no raw unvalidated data is passed to the database. How the data is validated is up to other testing methodologies, static tools will barely make sure it is passed through a filter of some sort. The third item will also prevent XSS attacks. By making sure no data is printed through the standard output, XSS can be avoided. Again, it does not say anything about how the encoding is done but merely that the function is used at all times. The last item will also provide XSS protection in a similar way.

5.3 Black-Box Testing

Black box testing is a methodology to test the actual product in a security related matter. The source code is supposed to be unknown and the system functionality is evaluated. This can be achieved without having any development background. The black-box testing is usually accomplished by testing the functionality exposed by the system. For a web-based system like the one in this report, all entry points are usually located and tested. Normally the testing should be done with someone without too much knowledge of the source code. The goal is to identify and execute against all different attack vectors. Since the author co-designed the system, it was difficult to perform any black-box testing with good results. Instead more focus was laid on white-box testing where code knowledge is not discouraged.

5.4 White-Box testing

White-box testing tests the internal structures of an application. This differs from black-box testing where the functionality is tested. White-box testing will test the full system from a developer's perspective. Different entry points will be tested with various input to check if the result is the expected. It's important to notice that there might be unused code that is not tested in this

approach since it's difficult to reach every single execution path from white-box testing. During the testing phase of this project a tool called Burp[12] was used. Burp works as a proxy and analyzes the data sent between the server and the client. It's possible to intercept and modify the traffic at any point. This makes it easy to manipulate the data sent from various input fields through out the system. Burp simplifies the way to pass raw data to the server. It overrides all possible JavaScript manipulation that can be done on the client before sending the HTTP request. It is also possible to intercept the server responses to make sure nothing can be bypassed by modifying the response code.

Chapter 6

Discussion and Conclusions

The system can from a design perspective be considered to be secure, given that the underlying cryptography and the users method to transfer a key is secure. Different code mistakes can still open up various attack vectors to make the system vulnerable, but from a design perspective, the system can be trusted. All encryption and hashing algorithms are standard and considered infeasible to violate. This is however an ever-changing subject and since the computational power continues to grow, so will the need for longer keys and new encryption standards. The current hashing algorithms used in this project (SHA-2) is soon to be replaced. NIST is currently hosting a competition to find a new SHA-3 standard to replace the older SHA-1 & 2. A winner is as of this thesis not yet presented but is said to be before 2013 ends.

This paper have used the National Institute of Standards and Technology (NIST) as a guideline on what cryptographic algorithms and key lengths to use. NIST is considered a reliable source and it is a common praxis to follow their recommendations. It is however a part of the United States Department of Commerce, which reports to the US government. They are not necessarily unbiased in their recommendations, but the recommended technologies are all standardized and for this thesis it was decided to follow their guidelines.

As a part of the requirements a custom authentication system was required to be used in the system. This is usually a part that opens up many different attack vectors. If this wasn't a requirement it could have been outsourced to a third party. One common approach is to use a technique called OAuth and OpenID which makes it possible to use a third party like Google or Facebook to authenticate the user. Once authenticated, the user bounces back to the original system. This is a good approach since the third party system is usually designed by a major company and therefore usually well tested and can in many cases be considered trusted. It is not a good idea to reinvent something that does not have to be invented but it was a system requirement and therefore it had to be done.

This project will uses two different libraries to handle cryptography; BouncyCastle and Keyzcar. BouncyCastle alone supports all features needed by the project. BouncyCastle is also a very well documented and tested source. The reason for using Keyzcar is mainly the simplicity within the API. While BouncyCastle requires a bit more work to get right, Keyzcar is designed to be as easy as possible to use. Unfortunately BouncyCastle is the only one to support full OpenPGP and is therefore necessary to the project.

The project was finished in time and is now used actively at the company to whom it was designed for. Although it is still being actively developed, a working product was delivered to the company after the completion of the thesis.

Bibliography

[1] http://code.google.com/p/owasp-esapi-js/.

[2] http://gs.statcounter.com/#browser-ww-monthly-201205-201205-bar.

[3] http://jquery.com.

[4] http://marketshare.hitslink.com/browser-market-share.aspx?qprid=0&qpcustomd=0&qptimeframe=m.

[5] http://msdn.microsoft.com/en-us/security/aa973814.aspx.

[6] http://pmd.sourceforge.net/.

[7] https://github.com/openpgpjs/openpgpjs.

[8] http://www.bouncycastle.org.

[9] http://www.hibernate.org.

[10] http://www.json.org.

[11] http://www.oracle.com/technetwork/java/javase/overview/javahistory-index198355.html.

[12] http://www.portswigger.net/burp/proxy.html.

[13] http://www.w3schools.com/browsers/browsers_stats.asp.

[14] www.keyczar.org.

[15] Preventing sql injection in java https://www.owasp.org, Jan 2008.

[16] Ecmascript language specification, June 2011.

[17] Rolf Oppliger Ralf Hauser & David Basin. Ssl/tls session-aware user authentication. Technical report, IEEE Computer Society, March 2008.

[18] James H. Burrows. Escrowed encryption standard (ees). Technical report, National Institue of Standards and Technology, U.S. Department of Commerce, 1994.

[19] Quynh Dang. Recommendation for applications using approved hash algorithms. Technical report, National Institue of Standards and Technology, U.S. Department of Commerce, 2009.

[20] S. Crocker J. Schiller & D. Eastlake. Randomness recommendations for security. Technical report, Network Working Group, December 1994.

[21] M.E Kabay. The parkerian hexad.

[22] Andrey Bogdanov Dmitry Khovratovich and Christian Rechberger. Biclique cryptanalysis of the full aes. Technical report, K.U. Leuven, Belgium; Microsoft Research Redmond, USA; ENS Paris and Chaire France Telecom, France, 2011.

[23] Martin Luther. Identity-based encryption and beyond. Technical report, IEEE Security Privacy Magazine, 2008.

[24] Stephane Manuel. Classification and generation of disturbance vectors for collision attacks against sha-1. Technical report, Springer Science+Business Media, LLC, 2010.

[25] Martin Abadi T. Mark A. Lomas & Roger Needham. Strengthening passwords. Technical report, Systems Research Center, 1997.

[26] Secretary of Commerce. Advanced encryption standard (aes). Technical report, National Institue of Standards and Technology, U.S. Department of Commerce, 2001.

[27] Donn B. Parker. A note on our terrible security model. *Computer Fraud & Security*, 2001(4):18, April 2001.

[28] Colin Percival. Stronger key derivation via sequential memory-hard functions.

[29] Jr Phillip J. Bond Donald L. Evans William Mehuron & Arden L. Bement. Security requirements for cryptographic modules. Technical report, National Institue of Standards and Technology, U.S. Department of Commerce, 2002.

[30] Elaine Barker & Allen Roginsky. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. Technical report, National Institue of Standards and Technology, U.S. Department of Commerce, 2011.

[31] Tom Rowan. Password protection: the next generation, Feb 2009.

[32] Bruce Schneier. Two-factor authentication: too little, too late. *Communications of the ACM*, 48(4), April 2005.

[33] Adi Shamir. Identity-based cryptosystems and signature schemes. Technical report, The Weizmann Institute of Science, 1984.

[34] Sheila Sproule. Password: Password. *The Canadian Manager*, 35(1), 2010.

[35] David Strom. Understanding two-factor authentication. *Baseline*, 2(86), 2008.

[36] J. Callas L. Donnerhacke H. Finney D. Shaw & R. Thayer. Openpgp message format. Technical report, Network Working Group, November 2007.

[37] Robert Morris & Ken Thompson. Password security: A case history. Technical report, Bell Laboratories, 1974.

[38] Huntley Vance. Data security in a real-time world requires 'defense in depth' strategy. *National Underwriter*, 114(25), July 2010.

[39] Jeff Williams and Dave Wichers. Owasp top 10 - 2010. Technical report, The Open Web Application Security Project (OWASP), 2010.

[40] Le Roux Yves. Information security - the cia model, Aug 1993.

[41] Shanqing Guo & Chunhua Zhang. Identity-based broadcast encryption scheme with untrusted pkg. Technical report, School of Computer Science and Technology, Shandong University, 2008.

Appendix A

Appendix

A.1 The Birthday Problem

The birthday problem, or paradox refers to the probability that, in a set of n randomly chosen people, two or more of them will have the same birthday. If the set contains 57 people the probability is 99% that two or more people will share a birthday. A 50% probability is reached with a set as small as 23 people. The paradox refers to the fact that a very small set is required to achieve a high probability of two people sharing a birthday. 100% is however not reached until the set contains 367 people since there are 366 possible birthdays including February 29. The paradox only states the probability of two people sharing an unspecified birth date. If a specific date is wanted, the paradox will not hold. The formula to calculate the probability is derived as follows:

$$P(n) = \frac{n! * \binom{365}{n}}{365^n}$$

n is the size of the set.

A.2 Solving Captchas

A captcha is a challenge-response test used to separate human users from computers. It is typically a distorted piece of text that is designed to be difficult to analyze for a computer but easy to read for a human.

There's mainly three different approaches to automate the process of reading and understanding a captcha.

1. Using character recognition software.

2. Use cheap human labor.

3. Exploit bugs in the CAPTCHA implementation.

Figure A.1: CAPTCHA examples.

Since no specific implementation is discussed here, the last option is outside the scoop of this thesis. There are companies specializing in providing services where a captcha is sent to the company, a human solves it and the solution is returned. Even if such a solution works pretty quickly, the average response time from one of the bigger suppliers is 17 seconds. Although this is certainly quick it will still be time consuming to perform a brute-force attack.

The third option is to use some kind of character recognition and decode the image. Character recognition is usually divided into three different parts, each one solving a smaller part of the problem.

1. Remove the background distortion. The background behind the text is usually covered in some pattern to make it more difficult to distinguish between character and background. Remove this by coloring it in one simple color.

2. Divide the captcha in smaller parts. Each part should contain one character. This is usually the most difficult of the three. As seen in the examples above the characters are usually connected somehow. Either by overlapping each other or by a line. To distinguish where one character ends and the next one starts is a cumbersome task.

3. Figure out what character is inside the smaller part. Once the word is split up, character by character it must be restored and translated to a real character.

Usually a method called Optical Character Recognition (OCR) is used. OCR is a method to read in words from pictures. It's commonly used to translate scanned images of documents to text documents. This project used a solution from Google called reCaptcha. ReCaptcha will run the generated captcha through an OCR. If Google's OCR couldn't solve the test it is outputted for the user to solve.

A.3 OWASP Top Ten

The following is the OWASP top ten list from 2010[39]. It contains a ranking of the ten most severe web application security flaws.

1	Injection
2	Cross-Site Scripting (XSS)
3	Broken Authentication and Session Management
4	Insecure Direct Object References
5	Cross-Site Request Forgery (CSRF)
6	Security Misconfiguration
7	Insecure Cryptographic Storage
8	Failure to Restrict URL Access
9	Insufficient Transport Layer Protection
10	Unvalidated Redirects and Forwards

A.4 Encodings

To avoid Cross-Site Scripting (XSS) it is important to encode dangerous JavaScript and HTML characters. Below is two tables displaying how the encoding was done in this project. The encoding is the same as the one used in the Microsoft AntiXSS library[5].

Table A.1: List of JavaScript encodings.

Character	Encoding
!	\x21
"	\x22
#	\x23
$	\x24
&	\x26
’	\x27
(	\x28
)	\x29
*	\x2a
+	\x2b
/	\x2f
:	\x3a
;	\x3b
<	\x3c
=	\x3d
>	\x3e
?	\x3f
@	\x40
[	\x5b
\	\x5c
]	\x5d
ˆ	\x5e
‘	\x60
{	\x7b
|	\x7c
}	\x7d
˜	\x7e

Table A.2: List of HTML encodings.

Character	Encoding
!	!
"	"
#	#
$	$
&	&
’	'
(	(
)	)
*	*
+	+
/	/
:	:
;	;
<	<
=	=
>	>
?	?
@	@
[	[
\	\
]	]
ˆ	^
‘	`
{	{
|		
}	}
˜	~