Eberhard Karls University Tübingen
Faculty of Science
Wilhelm Schickard Institute for Computer Science

Secure Data Storage in the Cloud via Smartphone - RSA Method

Bachelor Thesis

By Raphael Adam

Advisor: Prof. Klaus Reinhardt

Abstract

This bachelor thesis addresses the implementation of a secure cloud storage application. Cloud service providers often have access to the files of their customers. To prevent this and other security risks, a novel variant of the RSA encryption algorithm is employed. The application is web-based and therefore platform-independent. It is possible to create, open and edit text files directly in the browser. The files are encrypted and decrypted only locally on the computer of the user.

A special role in the implementation plays the user's smartphone which is used as a security token. The project comprises an Android application that basically generates and holds the decryption key. To open and decrypt a file, the user has to scan a QR code which is shown on the computer screen.

Contents

List of Figures

1 Introduction

1.1 Motivation

A popular method to store data is to use cloud services. Here is the data located on a server and can usually be accessed via the internet. The advantage is obvious, the data is not linked to a single device. When using a cloud storage, a file can, for example, be created on a desktop PC and later edited on a laptop without manually transferring the file from the PC to the laptop. But to attain this convenience, the data has to be given away to a service provider. In many cases, the provider can potentially access the files of a user. The cloud storage service Dropbox[1], for example, does encrypt the data. But Dropbox holds the decryption key and can still access the data. [1]

By using such services, the provider has to be trusted. This is not necessary when using the approach of this thesis. The user's files get encrypted before they are transferred to the server. For the encryption, a variant of the RSA[2] algorithm is used. To decrypt the data, the user has to scan a QR code with a smartphone. The full scheme can be seen in Chapter 3. A feature of this approach is that the private key of the user is basically located on the smartphone. So, even if an attacker places malware on the user's PC, he wouldn't be able to gain full access to all files of the user. Other cloud storage services that encrypt the files locally are vulnerable to such an attack if the decryption key is visible on the user's machine.

1.2 Objective of the Thesis

The approach as shown in Chapter 3 shall be implemented in a demo application. This demo comprises a website and an Android application. A user shall be able to create an account on the website. After the registration, he shall be able to create new text files and open existing files. The decryption is done by scanning a QR code with the Android application. Additionally, the user shall be able to edit the files in the web browser and to create files for other users.

[1] `https://www.dropbox.com/`
[2] See Section 2.1.

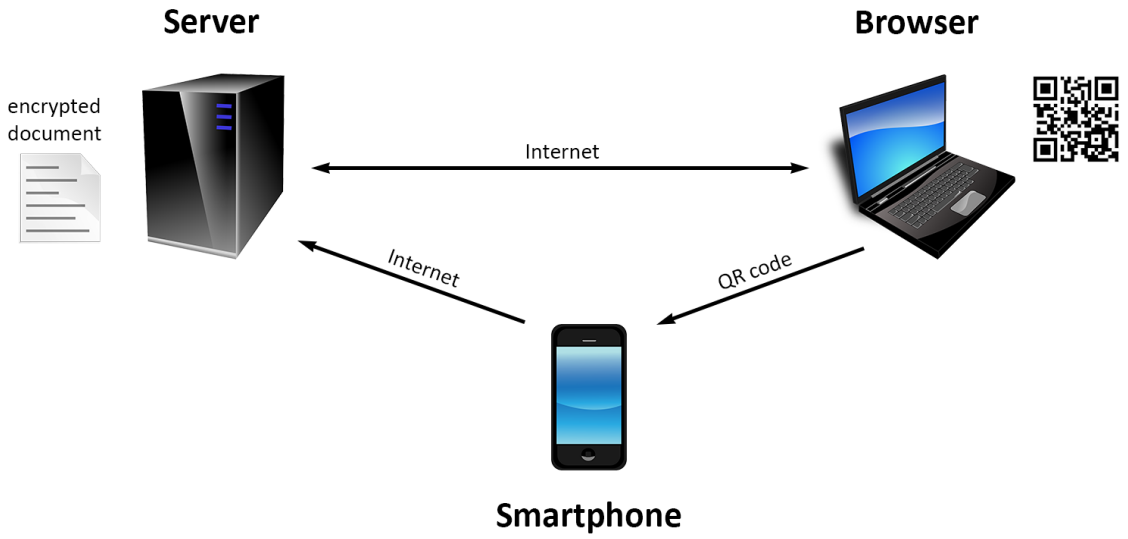

Figure 1.1: A rough overview of the communication between the devices

1.3 Structure of the Thesis

At first, the most important fundamentals for the comprehension of this thesis will be explained. This is mainly the RSA algorithm. After that, the concept of the encryption and decryption scheme will be presented. The concept will be followed by a detailed description of the implementation. At the end, the security and the applicability of the concept are discussed.

2 Fundamentals

2.1 RSA

RSA is a "public-key cryptosystem" developed by R. Rivest, A. Shamir and L. Adleman in 1977 [2]. Unlike symmetric-key algorithms, a public-key cryptosystem holds two different keys for encryption and decryption. The key for encryption can be made public while the decryption key must be kept private. For such a procedure, it is necessary that the private key cannot efficiently be computed when only knowing the public key. To achieve this, so called "trap-door one-way functions" are used. These functions are easy to compute in one direction but very difficult to compute in the other direction. If a special information is known, the other direction can be computed easily, which is the trap-door. This special information is the private key. [2, p.2-3]

When using a public-key cryptosystem, every user needs its own public and private key. The public keys are shared between all users that want to communicate with each other. Because a public key can be known by everyone, no secure channels are needed while exchanging the keys. If, for example, Alice wants to send a message to Bob she uses Bob's public key to encrypt the message and sends the ciphertext over the insecure channel to Bob. He uses then his private key to decrypt the ciphertext. When responding to Alice, he uses her public key. [2, p.3-4]

RSA is a practical implementation of this scheme. Breaking RSA is believed to be as difficult as factoring the modulus n[1] of the algorithm. As factoring is believed to be a hard problem, RSA can be considered secure if n is a sufficiently large number [2, p.11-13]. The following sections describe in detail how RSA works.

2.1.1 Key Generation

To generate a keypair, two primes p and q are chosen. Then n is computed which is the product of these two primes:

$$n = p \cdot q \tag{2.1}$$

[1] See Section 2.1.1.

The primes have to be very large and random to make sure n cannot be factored in a feasible time. The next step is to compute the Euler's totient function for n. This function returns the number of positive integers less or equal n that are relatively prime to n:

$$\varphi(n) = \left|\{a \in \mathbb{N} | 1 \leq a \leq n \wedge \gcd(a, n) = 1\}\right| \tag{2.2}$$

Because for a prime number p the totient function is $\varphi(p) = p - 1$ and because of elementary properties of the totient function, we get:

$$\varphi(n) = \varphi(p) \cdot \varphi(q) = (p - 1) \cdot (q - 1) \tag{2.3}$$

The public exponent e, $1 < e < \varphi(n)$ is then chosen such that $\gcd(e, \varphi(n)) = 1$. The private exponent d is computed such that:

$$ed \equiv 1 \pmod{\varphi(n)} \tag{2.4}$$

This can be done by using the extended Euclidean algorithm.

After doing this we have the **public key** (e, n) and the **private key** (d, n). For **encryption**, a message has to be represented as an integer M between 0 and $n-1$. To get the ciphertext C, the following computation has to be done:

$$C = M^e \bmod n \tag{2.5}$$

The **decryption** is almost the same step, but instead of the public exponent the private exponent is used:

$$M = C^d \bmod n \tag{2.6}$$

Only e and n can be made public. When knowing $\varphi(n)$, d can be obtained. And with p and q one can easily compute $\varphi(n)$. [2, p.6-7][3]

2.1.2 Correctness

The proof that (2.6) is valid is based on Fermat's little theorem. This states:

$$a^{r-1} \equiv 1 \pmod{r} \tag{2.7}$$

where r is a prime and a is an integer that is not divisible by r.

The formula for decryption resolves to:

$$C^d \bmod n = M^{ed} \bmod n \tag{2.8}$$

With equation (2.4), ed can be expressed as (k is some integer):

$$ed = 1 + k\varphi(n) = 1 + k(p - 1)(q - 1) \tag{2.9}$$

There are then two cases, the first one is that p doesn't divide M:

$$M^{ed} \equiv M^{1+k(p-1)(q-1)} \equiv M \cdot M^{p-1^{k(q-1)}} \equiv M \pmod{p} \tag{2.10}$$

If p divides M, M^{ed} is also congruent to M:

$$M^{ed} \equiv 0 \equiv M \pmod{p} \tag{2.11}$$

So $M^{ed} \equiv M \pmod{p}$ for all M. The same is valid if q is used instead of p. With this and the Chinese remainder theorem: $M^{ed} \equiv M \pmod{n}$. And because $M < n$: $C^d \bmod n = M^{ed} \bmod n = M$. [2, p.7-8][4, p.955,963]

2.1.3 PKCS#1 RSA Encryption Standard

A common scheme for a practical implementation of the RSA algorithm is PKCS#1 version 1.5 [5]. The encryption scheme of this standard shall shortly be explained here. It describes mainly how a message is encoded to represent it as an integer M.

The scheme can handle messages that are up to $k - 11$ bytes long where k is the length in bytes of the modulus n. The message (a string of bytes) will be denoted as m and its length in bytes as $mLen$. The first step is to build a string PS of randomly generated bytes with a length of $k - mLen - 3$. PS will be at least 8 bytes long. The next step is to concatenate m and PS to a byte string in this form:

$$\texttt{0x00} \mid\mid \texttt{0x02} \mid\mid PS \mid\mid \texttt{0x00} \mid\mid m$$

This byte string is then converted into an integer which is M. At decryption, the padding gets removed and only m will be returned. [5, p.23-26][3]

2.2 Pollard's Rho Method

The rho method is an algorithm by J. Pollard for factoring integers [6]. The procedure is heuristic, success and running time are not guaranteed. Despite of this, the method is highly effective in practice. [4, p.976]

The objective of the algorithm is to find a factor p of a number n. To achieve this, a sequence is considered such that x_0 is an arbitrary value and $x_{i+1} = f(x_i)$. The function f maps a value in $\mathbb{Z}_n$ to a pseudo-random value in $\mathbb{Z}_n$. Because the range of f is finite and every value of the sequence depends on the previous value, the sequence will repeat itself at a certain point. The algorithm tries to find two values x_i and x_j in the sequence with $x_i \equiv x_j \pmod{p}$. This is done by using Floyd's cycle detection algorithm. Since p is not known, the algorithm uses the property that if

```
x = 2;
y = 2;
d = 1;
while (d == 1) {
    x = (x*x - 1) MOD n;
    y = (x*y - 1) MOD n;
    y = (x*y - 1) MOD n;
    d = gcd(x-y,n);
}
if (d < n) {
    return d;
} else {
    return error;
}
```

Figure 2.1: Pollard's rho method (pseudo-code)

$x_i \equiv x_j \pmod{p}$ then $p|(x_i - x_j)$ and $\gcd(x_i - x_j, n) > 1$. If $\gcd(x_i - x_j, n) < n$, a nontrivial factor is found. [4, p.976-980][6]

The rho method uses the function $f(x) = (x^2 - 1) \bmod n$ and $x_0 = 2$ as start value [6]. A simple implementation in pseudo-code can be seen in Figure 2.1.

2.3 QR Code

A QR code (QR is an abbreviation for Quick Response) is a two-dimensional and machine-readable matrix code that was invented by the company Denso Wave in 1994. It represents information in a visual way, devices can read it by using a camera. Compared to other such codes, the QR code can hold a relatively high amount of data in a small space. QR codes also provide an error correction method by using the Reed-Solomon algorithm. So, even dirty or damaged codes can often be read. The QR code was originally designed for the automotive industry, but it is in wide-spread use nowadays. Most smartphones are able to read it in by using scanning applications. The QR code provides a way to quickly transfer a small amount of data, like a URL, to a smartphone. An example of a QR code can be seen in Figure 4.3. [7]

3 Concept

The concept of this bachelor thesis is based on a variant of the RSA algorithm[1]. Instead of using two exponents, e for encryption and d for decryption, the variant uses three. The encryption is the same as in the RSA algorithm, the decryption process differs from the original algorithm. How the decryption is implemented by using a smartphone is shown in Figure 3.1. The feature of this scheme is that the smartphone effectively works as the decryption key. Since the key is not saved on the machine that retrieves a file, a trojan horse on only that machine, for example, won't give an attacker access to all files on the server. The attacker would also need the secret data from the smartphone. By placing malware only on the smartphone otherwise won't let the attacker decrypt the files neither because the access to the encrypted files on the server is password-protected. This password is never seen by the smartphone. The Server also doesn't has the possibility to access the files. Encryption and decryption are both done locally on the PC of the user. The concept of this encryption and decryption scheme is explained in detail in this chapter.

When creating a new user in the cloud storage application, the smartphone generates all values for the RSA algorithm except the exponent d for decryption. These are p, q, n, $\varphi(n)$ and e. The primes p and q are discarded after n has been calculated. After the generation, the smartphone sends e and n to the server. This is the encryption key. When the user wants to encrypt a new file, the server sends the encryption key to the user. By computing $C = M^e \bmod n$ locally on the user's machine, the file gets encrypted[2]. After the encryption, the ciphertext C is sent to the server to save it there.

If a user wants to decrypt a file that he has saved on the server before, the server sends a browser-script to the user's machine. The script generates locally a random number r and a QR code which contains r. By scanning the QR code, r gets transmitted to the smartphone. The smartphone then generates the exponent d by calculating:

$$d = (r \cdot e)^{-1} \bmod \varphi(n) \tag{3.1}$$

This is not the same d as in the original RSA algorithm.

The multiplicative inverse of re in the ring of integers modulo $\varphi(n)$ exists if re is relatively prime to $\varphi(n)$. Since e has been chosen such that $\gcd(e, \varphi(n)) = 1$, we

[1] See Section 2.1.

[2] M is a part of the file, represented as integer. A file usually consists of many M, the explanation is only exemplary for one such block.

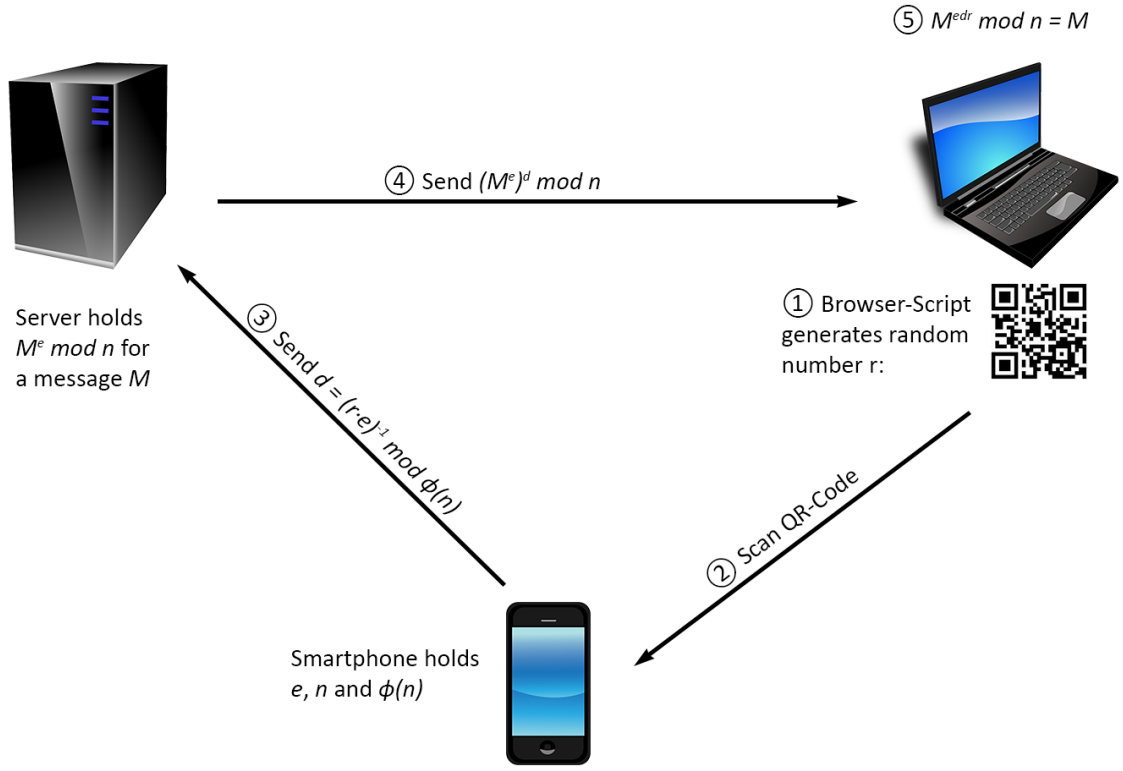

Figure 3.1: The decryption scheme

have to ensure that $\gcd(r, \varphi(n)) = 1$. This could be done by generating a random prime as r. But because r will be about 1000 bit long, the runtime of the generation of such a long prime would last too long in a browser. Instead of using a prime, Pollard's rho method[3] is used to make sure r has no small divisors. The case that $\gcd(r, \varphi(n)) \neq 1$ can occur with that method, but it is unlikely. If it occurs, a new r has to be generated.

The smartphone sends d then to the server. There

$$M^{ed} \bmod n \tag{3.2}$$

is calculated and sent to the user's machine. The file is still encrypted in that step. With the random number r the file gets finally decrypted on the user's machine by calculating:

$$M = M^{edr} \bmod n \tag{3.3}$$

Since $edr \equiv 1 \pmod{\varphi(n)}$, the correctness is proofed the same way as for the RSA algorithm, shown in Section 2.1.2.

The validity period of the random number r can be chosen with the smartphone after scanning the QR code. The user can select whether r is only valid for one file or for a certain time span.

[3] See Section 2.2.

4 Implementation

4.1 Use Cases and Programming Techniques

The following use cases will be implemented:

- **Create a new user:** One can create a new user by selecting a username and a password. The username must not be assigned yet.
- **Login:** A registered user can enter the member area by sending his username and password to the server.
- **Create new files:** A user can create new files for himself or for other users and save them encrypted on the server.
- **Retrieve files:** Files can be retrieved from the server. The files get decrypted by scanning a QR code with a smartphone.
- **Edit files:** Once a file is retrieved, it can be edited and saved again.
- **Login as guest:** One can log in without using a username and a password. The range of functions is limited then. The user can only create and save files for registered users for which he knows the usernames.

All the operations of the use cases are done by using a browser. Therefore, the main programming language for this implementation is HTML. The computations for the encryption and decryption process are done with JavaScript on the client-side. On the server-side, PHP is used for the computations and the processing of the user data. A lot of the communication between the client and the server is done with AJAX. This enables to send and receive data without loading a new web page. The application for the smartphone is written in Java for an Android device.

A restriction of using JavaScript in a browser on the client-side is that files can't be saved to the filesystem. Because of that, only text will be encrypted in this demo implementation. The text can be shown directly in the browser and doesn't need to be saved to the filesystem. A MySQL database is used on the server to save the encrypted text files. The database also holds the usernames and hashed passwords for all users.

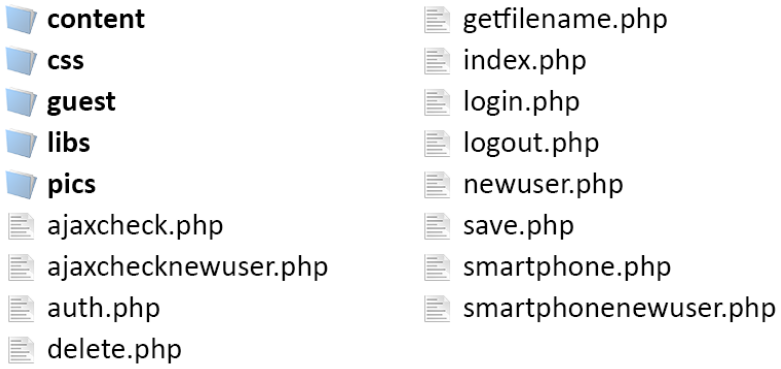

Figure 4.1: The root directory of the implementation

4.2 Structure of the Website

The entry point of the website is `index.php`. This file is delivered to the user on most servers if no specific file is requested. If a user, who is not logged in, opens `index.php`, the server redirects to `login.php`. If the user is logged in, the full functionality is accessible via `index.php`. The content of `index.php` is defined by the GET variable `show`. Different PHP files from the directory `content` are included into the body of `index.php` depending on the value of `show`. A request like `index.php?show=new` for example, shows `index.php` with the content of `new.php` in its body. This is done by using if-statements for some predefined values that `show` can assume. The files in the directory `content` should not be accessible over the internet but only by the server because it is only checked at the beginning of `index.php` if the user is logged in. To attain this, a `.htaccess` file is placed in this directory.

The style of the website is defined by CSS files in the directory `css`. JavaScript libraries and pictures are in the directories `libs` and `pics`. The file `rsa-variant.js` in the folder `libs` holds all JavaScript functions for encryption and decryption. All files for the guest mode are located in the directory `guest`. No login is needed to access these files. The entry point to the guest mode is another `index.php` in this folder. This file is similar to the `index.php` in the root directory but other PHP files get included. By executing `guest/index.php?show=new` for example, `guest_new.php` in the folder `guest` is included into the body of `guest/index.php`.

4.3 Login

In Figure 4.2 is shown what a user sees when retrieving `login.php`. The user can either log in, use the guest mode or create a new user. When logging in, the username and the password are sent to the server. The server then checks whether

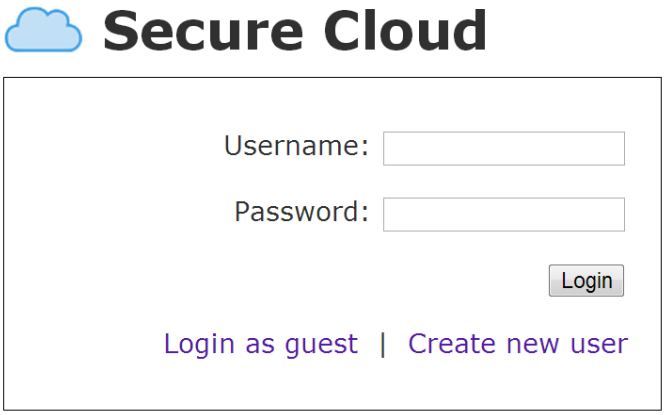

Figure 4.2: The login screen

the username exists in the database and whether the password is correct. If both applies, the server redirects to `index.php`.

The authentication is done by using PHP sessions. To establish a session, the server sends a unique session ID to the client. This ID is saved in a cookie. On every request, the session ID is sent to the server to identify the client. When a user sends the correct login credentials, a session variable is set which indicates that the user with this session ID is logged in. The authentication check is located in the file `auth.php`. This file is included at the beginning of every document which must only be accessible by users that are logged in. If the session variable is not set, `auth.php` redirects to the login page. To logout, the file `logout.php` has to be retrieved.

Due to security reasons, the passwords for the users are not saved in cleartext in the database on the server. When creating a new user, the password gets hashed first and is then saved. By computing the hash value of the entered password and comparing the value to that in the database, the password gets verified at the login.

The hash value is computed by using a cryptographic hash function. Such a function maps a variable length input to a fixed length output. Even small changes in the input should create a whole different output. One of the properties of a cryptographic hash function is that it is infeasible to compute the input when only knowing the output [8, p.323-324]. So by using a hash function, an attacker with access to the hash values doesn't know instantly the correlating passwords. However, if a user chooses a simple password, an attacker could still use "rainbow tables" for example to crack the password. Rainbow tables are effectively tables with precomputed hash values for a lot of passwords. These values are compared to the hash value. If one of them matches the user's value, a possible password is found. To prevent such an attack, a so-called "salt" is used. This means a certain value, the salt, is appended to the password before hashing. This value is different for every user and doesn't need to be kept secret. Rainbow tables can't be used when appending a salt to crack the password in a reasonable time, a new table would have to be computed for every different salt. In this implementation the username is prepended to the password

before hashing. As hash function, MD5 [9] is used. [10]

For the execution of most of the SQL commands, prepared statements are used. In contrast to regular statements, these are initialized before the user inputs are filled in. This is done to prevent SQL injections. A SQL injection means that a user sends an unexpected input which leads to a new and unwanted SQL command. Because prepared statements are initialized first, the database knows which command should be executed. If prepared statements can't be used, the input has to satisfy strong requirements which are checked on the server.

4.4 Creating a new User

To create a new user, one chooses a username and a password. The password can consist of arbitrary characters but the username must only contain letters and numbers. Both, the password and the username, can only be 20 characters long in maximum. The credentials are then sent to the server where it is checked whether the username is already in use. If not, the server hashes the password, saves it in a session variable and sends a script to the user which generates a QR code[1]. An example of such a QR code can be seen in Figure 4.3. The QR code holds 4 lines of text with the following information:

- **1. line:** The letter "N". Indicates that a new user gets created.
- **2. line:** The current session ID.
- **3. line:** The address to which the smartphone will send the confirmation.
- **4. line:** The selected username.

The QR code has now to be scanned by the Android application. The first line in the QR code tells the application to create a new user. The user has then to confirm if he really wants to create that user. If yes, the smartphone generates the values for the RSA algorithm as described in Chapter 3 by creating a new object of the Java class `RSA`. The random primes p and q are each 512 bits long. So, the modulus n which is the product of p and q has a length of 1024 bits. The values n, $\varphi(n)$, e and the username are then saved on the smartphone by using shared preferences. In this way the values are persistent and only this application has access to them.

After the computations are done, the application sends the username, the session ID and the values e and n via a POST request to the server. The address in the third line of the QR code tells the application where to send the data. A server-script,

[1]The QR code is generated by using the JavaScript library `qrcode`. Author: Jerome Etienne, MIT licence, `http://jetienne.com`

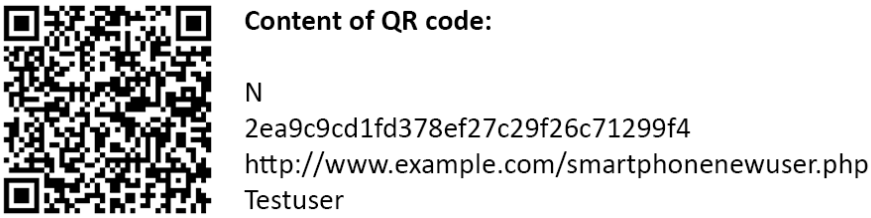

Figure 4.3: Example of a QR code while creating a new user

`smartphonenewuser.php`, gets executed which opens a session with the session ID sent by the smartphone. By doing this, the script has access to the session variables of the user. The server holds a table in the database called `users` in which the script creates a new record. This record holds the username, e, n and the hashed password that has been saved in a session variable. Additionally, a new table for this user is created in the database that looks like: `files_<username>`. The encrypted files of the user will be saved in this table later. After all entries are written in the database, the user can log in.

To redirect the user to the login page after scanning the QR code, "polling" is used. This means, a client-script is asking the server every 0.4 seconds whether a session variable is set. After the server-script has created the database entries, it sets this session variable. The next time the client checks the state of this variable, it knows that the user has been created and redirects to the login page. The polling is done in the background with AJAX. Because AJAX enables asynchronous requests, the page doesn't have to be reloaded and the user doesn't realize the requests. To signalize the connection to the user, a blinking dot is shown while polling. The polling process stops if there is no change after two minutes. Otherwise, there would be a constant traffic to the server.

4.5 Creating new Files

The user can either encrypt and save new files for himself or for other users for which he knows the usernames. The page for creating new files can be retrieved by a link in the navigation bar with the target `index.php?show=new`. By default, files are created for the user who is logged in. To upload files to the database of another user, a form gets shown by clicking on a link. If a valid username is submitted, the file will be created for this user.

The first step is to type in a name for the file. The filename doesn't need to be unique since consecutive file IDs are assigned to the files when saving them in the database. The text can be typed in a textbox. When clicking on a button, the text will be encrypted on the client by using JavaScript and sent to the server. The filename won't be encrypted.

Figure 4.4: The page for creating new files

On the server, the values e and n from the database of the user are written into the document that is delivered to the user. If the file will be uploaded for another user, his e and n are written into the code. By submitting the text, a JavaScript function gets executed. At first, the function converts e and n into `BigInt`[2] objects. This need to be done because of the big size of the used numbers. The `BigInt` library provides support of integers with arbitrary precision and several functions. The most important function of this library for this implementation will be `powMod()` that enables modular exponentiation for `BigInt` numbers.

After converting e and n into `BigInt` numbers, the function `encrypt()` from the library `rsa-variant.js` is called. This function can be seen in Figure 4.5. The parameters are the text that the user has typed in and the values e and n in `BigInt` representation. The first step is to split the text into blocks of the size of 117 characters. This is done, because the PKCS#1 encryption standard[3] is used. As described, this scheme can handle messages up to $k - 11$ bytes long where k is the length of n in bytes. Since n is 128 bytes long and every character is represented as

[2]Author: Leemon Baird, license: public domain, `http://www.leemon.com`
[3]See Section 2.1.3.

```
1  // text: String  e,n: BigInt
2  function encrypt(text,e,n) {
3      var bytes = 128; // bitlength: 1024
4      var blocks = Math.ceil(text.length/(bytes-11));
5      var result = "";
6      for (var i=0; i<blocks; i++) {
7          var pos = i*(bytes-11);
8          var str = text.substr(pos,(bytes-11));
9          var m = convertPlainTextToM(str,bytes);
10         var c = powMod(m,e,n);
11         if (i==0) result = bigInt2str(c,10);
12         else result = result + "/" + bigInt2str(c,10);
13     }
14     return result;
15 }
```

Figure 4.5: The function for encryption (JavaScript)

one byte, one block has a maximum size of 117 characters.

Every block is then converted into the PKCS#1 representation with the function `convertPlainTextToM()`. This function returns a `BigInt` which is the M in the algorithm of this thesis shown in Chapter 3. Then, C can be obtained by calculating $M^e \bmod n$. All encrypted blocks are finally merged by building a string with the decimal representation of every C. The blocks are separated by a "/". This string, which will be denoted as ciphertext, is returned by the function `encrypt()`.

The ciphertext and the filename are then sent via an AJAX POST request to the server-script `save.php`. The user, to which database table the file shall be saved, is noted in a session variable. The script writes the filename, the ciphertext and the current time in a new database record. At this point, a unique file ID is generated for this entry. If no error occurs, the string "success" is returned and a dialog informs the user that the file has been saved.

4.6 Managing Files

All files of the user are shown in a list when retrieving `index.php?show=files`. This page is accessible over a link in the navigation bar and is also the default view of `index.php`. The list shows the name of every file, the date of the last change and a button to delete the file. The name is also a link to open the file.

The script `files.php` of the folder `content` is included into `index.php` when retrieving `index.php?show=files` or only `index.php`. This script simply queries the

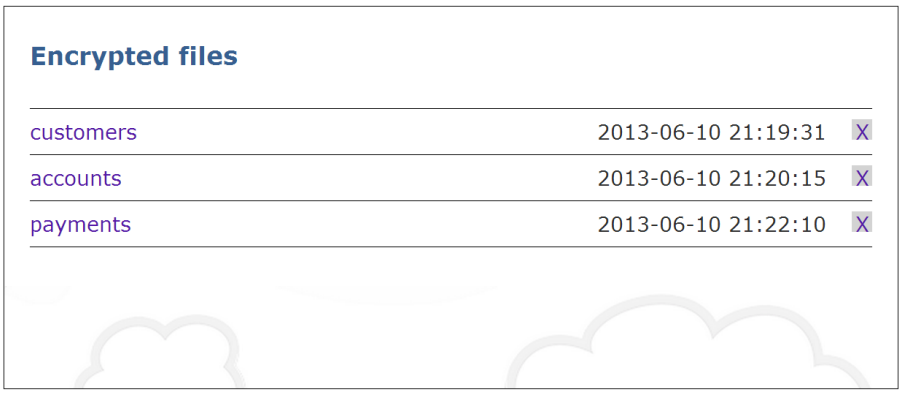

Figure 4.6: The list of the files

filename, the date of the last change and the file ID for every file of the user in the database. Except of the file ID, this information is written into a table to present it to the user. The file ID is necessary to identify the file when deleting or opening it.

To delete a file, a small button that shows a "X" besides the filename has to be clicked. The JavaScript function `deleteFile()` then gets executed. This function shows a dialog which asks the user to confirm that the file will be deleted. If the user confirms, the function retrieves the script `delete.php` in the root directory and commits the file ID via the GET variable `id`. The script deletes the file with this file ID from the database. After that, it redirects again to `index.php?show=files` which generates the list with the filenames again. This is done because the list has changed since a file has been deleted.

4.7 Retrieving Files

To open and decrypt a file, the user clicks on a filename in the list of the files. The user is then presented a QR code which he has to scan with the Android application. After scanning, the user can select on the smartphone how long the random value r will be valid. This can be either for one file only or for a specific period of time. When the user has made his choice, the page with the QR code redirects to a new page where he sees the decrypted text. If he has chosen that r is valid for a period of time, the next file can be accessed without scanning a QR code. Otherwise the procedure has to be reiterated.

The filenames in the list of all files are links to `index.php?show=openfile`. Via the GET variable `id`, the file gets identified. A file with the file ID 12, for example, would have the link `index.php?show=openfile&id=12`. If a user clicks on such a link the first time after he has logged in, `index.php` doesn't include `openfile.php` in its body but `scan.php`. This script shows a QR code to the user that holds the

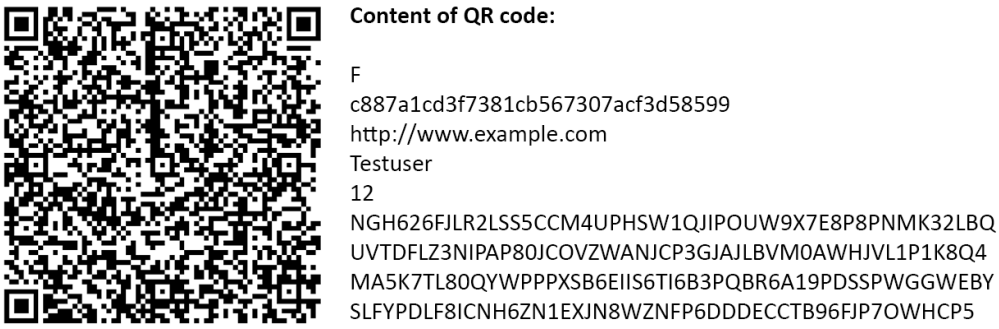

Figure 4.7: Example of a QR code while opening a file

following information:

- **1. line:** The letter "F". Indicates that a file gets opened.
- **2. line:** The current session ID.
- **3. line:** The root address of the server.
- **4. line:** The username.
- **5. line:** The file ID.
- **6. line:** The random number r in base-36 notation.

A example of such a QR code can be seen in Figure 4.7.

The random number r is generated with JavaScript on the user's machine by using Pollard's rho method which is described in Section 2.2. As explained in Chapter 3, we have to make sure that r has no small divisors. Such a r gets returned by the function `getRhoVal()` in the JavaScript library `rsa-variant.js`. It generates a random 1000 bit long number and checks if the rho method doesn't find a divisor for this number. If it finds a divisor, another random number is generated. This is repeated as long as divisors for the numbers are found. The algorithm differs from that shown in Figure 2.1, the number of loops that the while-statement can traverse is limited to 200. If no divisor is found in the 200 loops, this random number is returned by the function. The number, which is r, is then stored locally on the user's machine by using the JavaScript command `localstorage.setItem()`. With this technique, the data is persistent even if the browser gets closed. Finally, r is converted to a base-36 representation. This means, additionally to digits, every character from A to Z is used to represent the number. By doing this, the QR code has to hold a smaller amount of data.

After scanning the QR code with the smartphone, the Android application retrieves the name of the file by sending the file ID to the server-script `getfilename.php`.

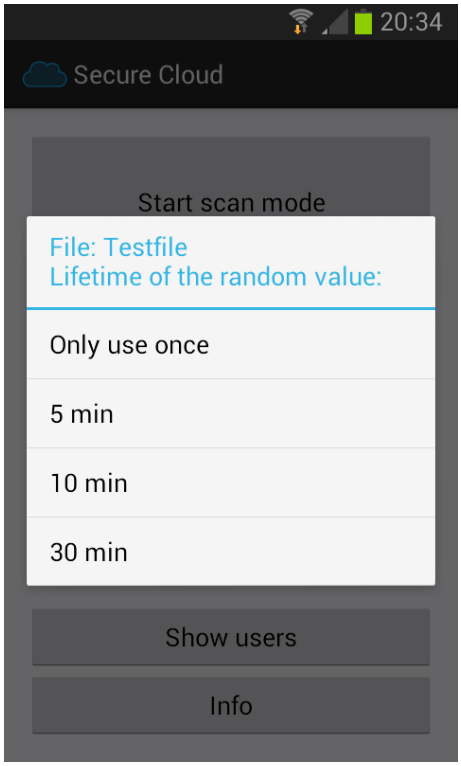

Figure 4.8: Dialog in app after scanning the QR code to open a file

This script returns the filename which is later shown to the user. Since the file is identified by the file ID, this has to be done to ensure that the user is accessing the file he intends to. If the filename would be written into the QR code additionally to the file ID, a trojan horse on the user's machine could, for example, present the user a QR code where the file ID does not match the filename. The user would think he decrypts one file but is actually decrypting another one selected by the attacker.

To calculate the value d that will be sent to the server, the application reads out the saved RSA values, which belong to the given username, from its storage. The values n, $\varphi(n)$ and e, that are read out, have been saved as text. With a constructor that accepts these three values as `String`, a object of the class `RSA` is created. The constructor converts the text values into `BigInteger` objects. The class `BigInteger` in Java is similar to the library `BigInt` in JavaScript. The method `calculateD()` of the `RSA` object is now called. It checks if $\gcd(r, \varphi(n)) = 1$ and computes $d = (r \cdot e)^{-1} \bmod \varphi(n)$.

The user is then presented a dialog on the smartphone as shown in Figure 4.8. He sees the filename that has been retrieved and can choose the lifetime of r. If he chooses "Only use once", the server opens the file in the so-called file mode. This means, a new random value r is generated when the user opens another file. As a consequence, the user has to scan a new QR code. If for example "5 min" is selected as lifetime, the file is opened in session mode. All other files that are opened in the next five minutes are decrypted with the same r. Therefore, the user doesn't need

to scan a QR code in this period of time.

When the user has made his choice, the application sends the session ID, the value d, the file ID and optionally the lifetime in seconds to the server-script `smartphone.php`. This script opens a session with the given session ID and saves d and the file ID in session variables. If the session mode had been chosen, the script also saves the point of time when the lifetime of r will be expired.

The page that shows the QR code is polling to the server while the user is scanning the code with the Android application. The client does this with AJAX by retrieving `ajaxcheck.php` every 0.4 seconds. This server-script returns "false" if the data from the smartphone is not received yet and otherwise "true". The page that is polling knows that it can redirect to the page that will do the decryption if it receives "true". For the file ID 12, for example, the page redirects to `index.php?show=openfile&id=12`. Since the value d is set now, `index.php` includes `openfile.php` instead of `scan.php`. However, if the file ID sent from the client differs from that sent from the smartphone, `scan.php` is included again. This is done to ensure again that no other file than the one the user has chosen gets decrypted.

At first, `openfile.php` reads out the filename and the ciphertext from the database. The ciphertext consists of blocks that are separated by a "/". The creation of this ciphertext has been explained in Section 4.5. Every block is the number $M^e \bmod n$ for a message M. As shown in Chapter 3, $M^{ed} \bmod n$ has to be calculated now. This is done on the server by taking every block to the power of d modulo n. After doing this, the server merges the changed blocks to a new string by separating the blocks again with a "/". This new string will be denoted as ciphertext'. If the file has been opened in the file mode, the value d is deleted on the server after this step. The server then writes n and the ciphertext' into the document and sends it to the user.

The final step is performed on the user's machine by computing $M^{edr} \bmod n$ which is M. To do this, r is regained from the local storage. The function `decrypt()` of the library `rsa-variant.js` then performs the decryption. This function expects the ciphertext', n and r as parameters. The values n and r have to be converted into `BigInt` numbers again before calling `decrypt()`. The function takes every block of the ciphertext' to the power of r modulo n. The changed blocks are then converted from the PKCS#1 representation to plaintext with the function `convertMToPlaintext()`. The decrypted plaintext of all blocks is merged to one string which is the original text the user has saved. This text is finally presented to the user in a textbox. The function `decrypt()` can be seen in Figure 4.9.

If the user has chosen the session mode, he can open other files in the selected period of time without scanning a new QR code. When clicking on a link to open a file, `index.php` checks at first whether the user is in file mode or in session mode. If in session mode, `index.php` further checks if the current date lies before the

```
// ciphertext: String  p,n: BigInteger
function decrypt(ciphertext,p,n) {
    var arr = ciphertext.split("/");
    var result = "";
    for (var i=0; i<arr.length; i++) {
        var c = str2bigInt(arr[i], 10, 0);
        var m = powMod(c,p,n);
        var str = convertMToPlaintext(m);
        result += str;
    }
    return result;
}
```

Figure 4.9: The function for decryption (JavaScript)

endpoint of the lifetime of the random value. This endpoint had been saved after the smartphone has transmitted the data to the server. If the lifetime is not over yet, the server uses the saved d to compute M^{ed} mod n. If the lifetime is over, d gets deleted and `index.php` includes `scan.php`. A new random value r is then generated and also d will be computed again.

4.8 Editing files

When the user has opened a file, the encrypted text is shown in a textbox. So, the user can immediately edit the text after opening it. By clicking on the button "Save changes", the text that is shown in the textbox gets encrypted again on the user's machine and the ciphertext is saved in place of the old one on the server.

The procedure when editing a file is similar to the creation of new files as shown in Section 4.5. At first the edited text gets encrypted by calling the function `encrypt()`. Then the ciphertext and the file ID are sent via POST variables to the server-script *save.php*. Since this script handles the creation of new entries in the database and also updates of existing entries, another POST variable indicates that an existing entry will be updated. The received file ID identifies the entry for the file to be updated. The ciphertext in this entry is then replaced by the new ciphertext in the POST variable. If no errors occur, `save.php` returns "success". Finally, the web page informs the user about the outcome by showing a dialog.

4.9 The Guest Mode

By using the guest mode, someone can upload files for any user. The uploader doesn't need to be logged in or even need to be a registered user. All he needs is the username for the user for which he wants to upload files. The guest mode can be accessed via a link on the login page. The target of the link is `guest/index.php`. All files for the guest mode are in the directory `guest`. The structure of the guest mode is similar to the normal login mode, `guest/index.php` is including files from the folder `guest` depending on the values of GET variables.

When receiving `guest/index.php`, a PHP session is started but no authentication is required. The server-script then includes `guest/guest_selectuser.php` into its body which shows the user a simple form with one input field. With this form, a user for which to upload files for can be selected. The username, that has been typed in, is then transmitted to `guest/index.php` via the GET variable `user`. The server then checks if this username exists. If it does, the values of this user for encryption are read out of the database and written into session variables. Also, the file `guest/guest_new.php` is included into the body of the document. The user sees then a the page, similar to the one shown in Figure 4.4, where he can encrypt and upload new files for the selected user. The process of creating a new file is the same as described in Section 4.5. During the lifetime of the PHP session, the values for the selected user are saved. So, multiple files can be uploaded without sending the username on every time to the server. But the user can be changed the same way as in the login mode by clicking on "Upload for other user".

4.10 The Android Application

The user can immediately scan a QR code when starting the application on a smartphone. After doing this, a dialog is shown. This can be either a dialog to choose the lifetime of the random value r, as seen in Figure 4.8, an confirmation dialog or an error message. Then, the application switches to the main screen with buttons to start the scanner again and to show a list with all users that are registered on this smartphone.

As scanner for the QR codes, a part of the Android application Barcode Scanner[4] is integrated in the application of this project. This is done by using a small library that is provided by the ZXing team [11]. It opens the scanner via an `Intent` and handles, among other things, the case where Barcode Scanner is not installed. `Intents` enable an application to invoke parts from other applications. With the method `onActivityResult()` in the main `Activity`, which is called after scanning, the content of the QR code is received. The complete code for integrating the

[4]Author: ZXing team, license: Apache License 2.0, `https://code.google.com/p/zxing/`

```
1  // Start the scanner
2  IntentIntegrator integrator = new IntentIntegrator(this);
3  integrator.initiateScan();
4
5  // The method that receives the results
6  public void onActivityResult(int requestCode, int
       resultCode, Intent intent) {
7      IntentResult scanResult =
           IntentIntegrator.parseActivityResult(requestCode,
           resultCode, intent);
8      if (scanResult != null) {
9          String content = scanResult.getContents();
10         String format = scanResult.getFormatName();
11         if ((content != null) && (format != null)) {
12             handleScanResult(content, format);
13         }
14     }
15 }
```

Figure 4.10: The code for integrating the QR scanner (Java)

scanner can be seen in Figure 4.10.

The scanning result is passed as parameter to the method `handleScanResult()`. This method checks at first whether the content of the first line in the text of the QR code is either a "N" or a "F". These are the two possible modes, creating a new user ("N") and open a file ("F"). If a file gets opened, the method checks further if the user is known by the device and whether the internet is accessible. If both applies, the value d is calculated and the filename is obtained by retrieving `getfilename.php`. When receiving the response, the dialog to choose the lifetime of r is shown to the user. The smartphone then sends all information the server needs via a POST request to `smartphone.php`. If a new user gets created, the method also checks if the internet is accessible and asks the user to confirm the action by showing a dialog. Then, the RSA values are generated and saved. The required information is finally sent to the server.

5 Security

The encryption taken by itself can be considered secure since basically the RSA algorithm is used which is believed to be based on a hard problem and is also in practical use for many years by now [2, p.11-13]. As key length (size of the modulus n), the company RSA Security currently recommends 1024 bits for corporate use and 2048 bits for extremely valuable data [12]. In this demo application, 1024 bits are used to achieve a better performance.

5.1 Malware Attacks

An implementation of the encryption scheme could possibly be attacked by using malware like trojan horses. Though, a trojan horse on only one of the three devices (smartphone, PC, server) is not able to attain full access to all files of a user. All possible malware attacks will be examined here.

Malware on the smartphone

If the smartphone is infected, the attacker would have access to $\varphi(n)$ and could decrypt the encrypted files. But since the smarphone has no knowledge of the password, which is needed to log in to the server, the attacker couldn't access the encrypted files.

Malware on the PC

By placing a trojan horse on the user's machine, an attacker could read all files the user retrieves and could attain the password. He would then have access to all encrypted files. But with no knowledge of $\varphi(n)$ or d, the attacker wouldn't be able to decrypt them.

Insecure server

An attacker who has access to the server, would have access to the encrypted files and to the value d which is sent by the smartphone. But the attacker would also

have to know the random value r in order to decrypt the files. This value is never sent to the server.

Malware on the smartphone and on one other device

As explained, an attacker would also need access to the encrypted files when having access to the secret data of the smartphone. The files could either be retrieved through an insecure server or by spying out the user's password on the PC.

Insecure server and malware on the PC

The combination of having access to the server and to the PC also allows an attacker to read all files of the user. The attacker would have access to the encrypted files and could decrypt them by using a matching pair of the values d and r. Though, this attack requires the user to compute a value d for a given r during the period of time the server and the PC are monitored by the attacker.

So, an attacker needs access to two arbitrary devices involved in the decryption scheme to read all files of the user. Because smartphones often get connected to laptops or desktop computers to exchange photos for example, this is a realistic thread. Malware on a PC could use such a connection to infect the smartphone. There is no final solution to prevent malware attacks, although installed updates and virus scanners can minimize the risk.

5.2 Other Security Risks

Since the communication between the PC and the server is not encrypted in this demo implementation, the password or the session ID could be intercepted. By doing this, an attacker could, at least for a period of time, log in to the account of the user. Especially in open WiFi networks this attack can easily be performed because all data traffic can be seen by all peers. Compared to other web applications, this attack is a rather low thread since the files of the user are still encrypted. Although, to gain access to the account can be part of an attack. In combination with malware on the smartphone, this can be a security risk. A solution is to use HTTPS, the traffic then gets encrypted by using the TLS/SSL protocol.

A minor security risk could be the circumstance that the filenames are not encrypted. An attacker could either gain useful information directly from the filenames or get to know which file is most worthwhile to decrypt. The implementation could be extended so that the filenames are also encrypted. But the user would have to scan a QR code before seeing a list of his files.

A traditional scheme for end-to-end encryption in a cloud storage would be to use a

symmetric encryption algorithm. The data is then encrypted and decrypted locally and the key is either saved on the user's computer or typed in by the user. In both methods, the key could be retrieved by malware on the computer. As explained above, this is not as easy possible in the scheme of this thesis. To sum up, the presented implementation provides a high level of security. Only the communication with the server would yet have to be encrypted by using TLS/SSL.

6 Conclusion

A cloud storage web application which is based on a new variant of the RSA encryption algorithm has successfully been implemented. The concept provides a high level of security, encryption and decryption are both performed on the client's side. The decryption key is neither stored on the user's machine, nor has the key to be typed in manually.

Though, there are some limitations of this implementation, especially that only text files can be saved on the server. For a wide-spread use of the application, a user should be able to save all types of files. This could be realized by creating a standalone program instead of using the browser, since a JavaScript file executed in a browser is currently not able to save to the filesystem. This is necessary, because the data received by the server has to be processed on the client for decryption.

A constraint of the concept itself is that the smartphone basically can't be used to retrieve files from the server. Another smartphone would be needed to scan the QR code, which is of course not viable for practical use. Skipping the step of scanning the QR code and reducing the concept to only use the smartphone and the server is no solution either. The security benefits of the scheme would be removed and a simpler encryption method could be used as well.

An improvement of the implementation could be to use a symmetric encryption algorithm, like AES, to actually encrypt the files and use the variant of the RSA algorithm to encrypt only the key for the symmetric algorithm. AES, for example, has a much better performance in computation time than RSA [13]. To every file, a encrypted AES key could be stored on the server. Since only small text files are encrypted in the implementation of this thesis, this method wasn't necessary here. But for larger files, the RSA algorithm would probably be too slow.

Despite of some constraints, the approach of this thesis provides security and also preserves the usability. Scanning a QR code is quickly done and the user doesn't have to remember another password for the decryption process. Also, the fact that many users choose weak passwords, can't be exploited here.

Bibliography

[1] Dropbox. How secure is Dropbox? https://www.dropbox.com/help/27/en. Retrieved July 23, 2013.

[2] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[3] D. Ireland. RSA Algorithm. http://www.di-mgt.com.au/rsa_alg.html, 2013. Retrieved Mai 10, 2013.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[5] PKCS #1 v2.2: RSA Cryptography Standard. http://www.rsa.com/rsalabs/pkcs/files/h11300-wp-pkcs-1v2-2-rsa-cryptography-standard.pdf, 2012.

[6] J.M. Pollard. A monte carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334, 1975.

[7] Denso ADC. QR Code Essentials. http://www.nacs.org/LinkClick.aspx?fileticket=D1FpVAvvJuo%3D&tabid=1426&mid=4802, 2011. Retrieved June 27, 2013.

[8] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., 1st edition, 1996.

[9] R. Rivest. The MD5 Message-Digest Algorithm (RFC 1321), 1992.

[10] J. Barneck. How to effectively salt a password stored as a hash in a database. http://www.rhyous.com/2012/06/18/how-to-effectively-salt-a-password-stored-as-a-hash-in-a-database/, 2012. Retrieved Mai 25, 2013.

[11] How to scan a barcode from another Android application via Intents. https://code.google.com/p/zxing/wiki/ScanningViaIntent, 2012. Retrieved June 26, 2013.

[12] RSA Laboratories. How large a key should be used in the RSA cryptosystem? `http://www.rsa.com/rsalabs/node.asp?id=2218`, 2012. Retrieved July 9, 2013.

[13] Shashi Mehrotra Seth and Rajan Mishra. Comparative analysis of encryption algorithms for data communication. *International Journal on Computer Science and Technology*, 2:292–294, 2011.

Statement of Authorship

Erklärung

Hiermit erkläre ich, dass ich diese schriftliche Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe.

Tübingen, den 12. August 2013

_____

Raphael Adam