
# CryptoOne-System : Sicherheit im Computernetze

## Bachelorarbeit

## Vorgelegt von:

Fabrice Dufils Siyapdje

Betreuer:

H. Prof. Dr. Martin Damm, Hochschule Mannheim

H. Christian, Hochschule Mannheim

Fakultät für Informationstechnik, Hochschule Mannheim

Paul-Wittsack-Straße 10, 68163 Mannheim

Mannheim, 15. Februar 2016

*"Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegeben Quellen und Hilfsmittel benutzt habe."*

# Abstract

# Table of Contents

Abkürzungsverzeichnis

[SGK] : Symmetric Group Key [ITS] : IT-Infrakstruktur [PGP] : Pretty Good Privacity [SPKI]: Simple Public Key Infrastructur [SDSI]: Simple Distributed Security Infrastructure [SRP]: Secure Remote Password Protocol [AKE]: Asymmetric Key Exchange [SPA]: Single Page Application [MVC]: Model View Controller [MVVM]: Model View ViewModel [HTTP]: Hypertext Transfer Protocol [DNS]: Domain Name System [UDP]: User Datagram Protocol [TCP]: Transmission Control Protocol [TLS]: Transport Layer Security [FTP]: File Transfert Protocol [SSL]: Secure Sockets Layer [AES]: Advanced Encryption Standard [RSA]: Rivest, Shamir und Adleman [REST]: Representational State Transfer [CRUD]: Create Read Update Delete [CA]: ertificat Authority [PKCS]: Public Key Cryptography Standards [SHA]: Secure Hash Algorithm [IIS]: Internet Information Services [NIST]: National Institute of Standards and Technology [PGP]: Pretty Good Privacy [SPKI]: Simple public Key Infrastructur [ACL]: Access Control List [JCA]: Java Cryptography Architecture

# Chapter 1

# Einführung

## 1.1 Gliederung

Diese Arbeit lässt sich in drei große Abschnitte aufteilen: Kapitel 2 behandelt die Anförderungen eines sicheren Dokumentaustausch sowie der Authententifizierungmechanismen die für das Verständnis der weiteren Kapitel wichtig sind. Im folgenden Kapitel 3 werden beispielhaft die aktuelle Stand der Technick vorgestellt und ihre technische Umsetzung aufgeführt. In Kapitel 4 wird anhand der Problemstellung ein Konzept für die Dokumentaustauschplattform erstellt, welches in den Kapiteln 5 und 6 konkretisiert und implementiert wird. Die letzten beiden Kapitel 7 und 8 fassen die Ergebnisse dieser Arbeit zusammen und machen Vorschläge für eine Verbesserung des Systems.

## 1.2 Motivationen

Der Austausch von vertraulichen Informationen im Netz ist grundsätzlich problematisch. Wie können Informationen zwischen Parteien augetauscht werden, ohne dass Unberechtigte diese mitlessen können.

Der Austausch von vertraulichen Informationen mittels schriftlicher Aufzeichnungen ist grundsätzlich problematisch. Wie sollten Informationen zwischen Parteien ausgetauscht werden, ohne dass Unberechtigte diese mitlesen können. Die Lösung des Problems besteht darin, die Nachricht verschlüsselt zu übertragen. D. h. die ur- sprüngliche Nachricht wird so verändert, dass es Unberechtigten deutlich erschwert wird, den Inhalt einer abgefangenen Nachricht zu erfassen. Bereits in der Antike wurden vertrauliche Informationen verschlüsselt übermittelt. Schon damals bestan- den schon folgende Probleme, die noch heute - trotz aufwendigerer Verschlüsselung – relevant sind. • (1) Wer kann Nachrichten ver- bzw. entschlüsseln und wie? • (2) Wie werden die Schlüssel zwischen Sender und Empfänger ausgetauscht? Die Notwendigkeit eines sicheren Dokumentenaustauschs hat sich in den letzten Jahren als immer drängender erwiesen, da zum einen ein Austausch via Internet sehr schnell und einfach möglich ist. Zum anderen ist spätestens durch die NSA-Affäre deutlich geworden, dass eine Datenübertragung via Internet nicht für vertrauliche Informationen ohne weitere Maßnahmen geeignet ist.

## 1.3 Stand der Technick

### 1.3.1 Email ( SMTS )

### 1.3.2 FTP Server

### 1.3.3 Web EDI

#### 1.3.3.1 AS2

#### 1.3.3.2 Web-Upload

Das Speichern von Dokumenten auf einem Internet-Server ist weit verbreitet und weltweit von jedem Browser aus möglich. Eine Installation zusätzlicher Software, oder gar die Öffnung zusätzlicher Ports der Unternehmens-Firewall ist nicht notwendig. Die Benutzer-Authentifizierung erfolgt i.d.R. per Login/Password. Daten können im Internet mittels des https-Protokolls verschlüsselt übertragen wer- den. Fälschlicherweise wird angenommen, dass die übertragenen Dokumente dann auch beim Empfänger „sicher“ gespeichert sind. Jedoch werden lediglich die Dokumente auf dem Weg zum Server mit SSL verschlüsselt. Danach liegen sie zunächst unverschlüsselt vor. So werden von einem Server verschlüsselt übertragene Doku- mente vom Browser entschlüsselt und im Klartext auf dem lokalen PC gespeichert. Ebenso werden Dokumente, die vom Browser für die Übertragung verschlüsselt werden vom Server entschlüsselt und liegen am Server unverschlüsselt vor. Somit besteht dieselbe Problematik und auch derselbe Lösungsansatz wie bei Datei- Servern. In Folge dessen sollten Dokumente, die per Browser

auf einen Datei-Server geladen werden, vom Client-PC verschlüsselt werden. Die Dokumente müssen also vor dem Upload verschlüsselt worden sein, oder aber der Browser führt die Ver- schlüsselung durch. Eine Vorab-Verschlüsselung der Dateien hat den Nachteil, dass das Dokumenten- und Schlüssel-Management vom Anwender eigenverantwortlich durchgeführt werden muss. Dies ist i.a. den Anwendern zu aufwendig. Folglich sollte die Verschlüsselung durch den Browser quasi automatisch erfolgen. Dies wird aktuell nur sehr selten durchgeführt, da die Verschlüsselungs-Software auch vom Web-Server geladen werden müssen. Und es kann nicht garantiert werden, dass die geladene Software nicht Eindringlingen unbeabsichtigten Zugriff er- möglicht. In Folge dessen werden Dokumente SSL-verschlüsselt zum Server gesen- det. Die dort empfangenen, unverschlüsselten Dokumente werden sofort verschlüs- selt und als Datei abgelegt. Hier bestehen jedoch folgende Probleme: (1) Wie kom- men die notwendigen Schlüssel zum Server? (2) Ein Eindringling auf dem Server kann die Klartext-Datei und/oder die Schlüssel mitlesen. Zusammenfassung Ein Ansatz für ein sicheres Web-Upload ist bisher nicht bekannt.

# Chapter 2

# Stand der Technik

Dieses Kapitel beschäftigt sich mit der Erlaueterung von wichtigen Begriffen, sowie einer Forschung von grundlegenden und aktuellen Technologien.

## 2.1 Ueberblick

### 2.1.1 Email

eine im ersten Blick triviale Lösung was Austausch von kristischen Dokumenten angeht besteht darin diese Dokument zu verschlüsseln und die resultierende verschlüsselte Dokument per Email an der Kommunikationspartner zu senden. Diese Lösung ist solange ertragbar wenn der Benutzer sich mit Cryptographie bzw Cryptographiesoftware auskennt. Begrenzungen an diese Technik sind unerheblich und unheimlich viele :

- Diese Loesung setzt voraus dass der Kommunikationspartner sich auch mit der Kryptographie bzw. Kryptographiesoftware auskennt.
- mehr problemetik, setzt sich auch voraus dass der Kommunikationpartner neben der Kryptographie know-how, den passenden Software, den passenden kryptographischen Algorithmus und der Verschlüsselungschlüssel.
- Schlüsselaustauschsproblematik.
- Infrakstrukturproblemetik.

### 2.1.2 Web-upload

Das Speichern von Dokumenten auf einem Internet-Server ist weit verbreitet und weltweit von jedem Browser aus möglich. Eine Installation zusätzlicher Software, oder gar die Öffnung zusätzlicher Ports der Unternehmens-Firewall ist nicht not- wendig. Die Benutzer-Authentifizierung erfolgt i.d.R. per Login/Password. Daten können im Internet mittels des https-Protokolls ver-

schlüsselt übertragen wer- den. Fälschlicherweise wird angenommen, dass die übertragenen Dokumente dann auch beim Empfänger „sicher“ gespeichert sind. Jedoch werden lediglich die Doku- mente auf dem Weg zum Server mit SSL verschlüsselt. Danach liegen sie zunächst unverschlüsselt vor. So werden von einem Server verschlüsselt übertragene Doku- mente vom Browser entschlüsselt und im Klartext auf dem lokalen PC gespeichert. Ebenso werden Dokumente, die vom Browser für die Übertragung verschlüsselt werden vom Server entschlüsselt und liegen am Server unverschlüsselt vor. Somit besteht dieselbe Problematik und auch derselbe Lösungsansatz wie bei Datei- Servern. In Folge dessen sollten Dokumente, die per Browser auf einen Datei-Server geladen werden, vom Client-PC verschlüsselt werden. Die Dokumente müssen also vor dem Upload verschlüsselt worden sein, oder aber der Browser führt die Verschlüsselung durch. Eine Vorab-Verschlüsselung der Dateien hat den Nachteil, dass das Dokumenten- und Schlüssel-Management vom Anwender eigenverantwortlich durchgeführt werden muss. Dies ist i.a. den Anwendern zu aufwendig. Folglich sollte die Verschlüsselung durch den Browser quasi automatisch erfolgen. Dies wird aktuell nur sehr selten durchgeführt, da die Verschlüsselungs-Software auch vom Web-Server geladen werden müssen. Und es kann nicht garantiert werden, dass die geladene Software nicht Eindringlingen unbeabsichtigten Zugriff er- möglicht. In Folge dessen werden Dokumente SSL-verschlüsselt zum Server gesen- det. Die dort empfangenen, unverschlüsselten Dokumente werden sofort verschlüs- selt und als Datei abgelegt. Hier bestehen jedoch folgende Probleme: (1) Wie kom- men die notwendigen Schlüssel zum Server? (2) Ein Eindringling auf dem Server kann die Klartext-Datei und/oder die Schlüssel mitlesen. Zusammenfassung Ein Ansatz für ein sicheres Web-

Upload ist bisher nicht bekannt.

### 2.1.3 FTP-Server

Das Problematik bei FTP Server ist dass eine Dritte von aussen aus auf den auf der FTP-Server gespeicherte Datein nicht zugreifen kann. Zusätzlich muss der Benutzer der Verantwortung tragen die Datein zu verschlüsseln, und selber die Schlüsseln verwalten.

### 2.1.4 Cloud-Service

Cloud-Service hat sich in der letzen 5 Jahre wesentlich verbreitet. Und war auf einer guten Weg bis zum NSA-Affäre sich als defakto Standard einzusetzen. Heute auch trotz die Spionnageskandale, wird Cloud-Service bei viele Endbenutzer sehr beliebt. Das Risiko ihre geheime Dokumente gestohlen zu haben, was Endbenutzer eingehen, können sie sich Unternehmen nicht leisten. der Einsatz von Cloud-Service bei Unternehmen ist ein absolut No-go.

Es besteht hier die gleiche Problematik wie bei FTP Server

## 2.2 Schlüsselaustausch

der Schlüsselaustausch ist von grossen Bedeutung was Netz- und Informationssicherheit angeht. Auch bei etablierte Sicherheitsoftware ist Schlüsselaustausch problematik. Aufgrund seines Sensibilität gehört Schlüssel zu **kritische** Informationen.

## 2.3 Begriffe

### 2.3.1 kritische Informationen

Er handelt sich um Informationen bzw. Daten die auf keinen Fall nirgendwo in der verschieden Softwarekomponente unverschlüsselt abgespeichert werden dürfen, oder unverschlüsselt durch der Netz geschickt werden dürfen.

Zu diese Kategorie gehören beispielweise wichtige Benutzersdokumenten, oder Benutzerscredentials.

### 2.3.2 Schlüssel

Hier handelt es sich um kryptographische Schlüssel oder anders ausgedruckt Chiffrierschlussel. Diese kann verschiedene Formen haben, und jenach Schlüsselart entweder zur kritischen oder nichtkritischen Informationen gehören.

#### 2.3.2.1 Schlüsselpaare

Anhand der RSA Algorithmus werden Sclüsselpaare benötigt. Schüsselpaare besteht aus zwei Schlüssel : eine geheime und eine öffentliche Schlüssel. Öffentliche Schlüssel wird eingesetzt um Chiffrierung durchzuführen, geheime Schlüssel dagegen führt die Dechiffrierung durch.

**geheime Schlüssel auch bekannt private Schlüssel ist eine kritische Information**

#### 2.3.2.2 Symmetrische Schlüssel

Es handelt sich um eine geheime Schlüssel, die Anhand der AES Algorithmus ( Symmetrische Verschlüsselungsverfahren ) eingesetzt wird, um Chiffrierung und Dechiffrierung durchzuführen. **Da die syymetrische Schlüssel sowohl zur Chiffrierung als auch zur Dechiffrierung eingesetzt wird, ist die Schlüssel eine kritische Information.**

### 2.3.3 Passwort und Passphrase

- Unter Passwort versteht man der nur beim Benutzer bekannte Zeichenkette, den ihn ermöglich sich in den System anzumelden.
- Passphrase ist auch nur von der Benutzer bekannt, und darf nicht in irgendeine Form persistent gehalten. Den Passphrase wird benutzt um Benutzer geheimschlüssel zu verschlüsseln.

### 2.3.4 Symmetrische Verschlüsselung

### 2.3.5 Asymmetrische Verschlüsselung

### 2.3.6 Publickeys Infracstructur

## 2.4 Zusammenfassung

Schlüsselmanagementssystem und PBK-Infrakstruktur tragen exclusiv die Verantwortung über :

- Schlüsselerzeugung
- Schlüsselmanagement
- Schlüsseldeployment
- usw.

An sich ist dies nicht problematisch, da die Software machen genau das, wofür sie konzipiert wurden, wobei wie schon besprochen einige Einschrängkungen bestehen was Portabilität und Flexibilität angehen. Dateiablegerung und Dateiaustausch System erfüllen auch genau die Aufgabe wofür sie konzipiert wurden, dabei bestehen jedoch gravierende Sicherheitsproblematik.

- Vorabverschlüsselung von Datei
  - Vorabinstallation von Schlüssel
  - Übertragung von Schlüssel
- Offnung weitere Port wie bei FTP-Server
- eingeschränkte Einsatzt von HTTPS bzw. Endknote-problematik
- Redeployment

Diese Arbeit setzt sich als Ziel, eine System zu konzipiert, der die oben genannte Lücke erfüllen, und von die beide Technologie eins macht, sowie eine starke Authentication und zuverlässige Vertrauenmechanismus.

sicher ist ( zB: HTTPS ) aber nich den Empfang bei Kommunikationspartner.

# Chapter 3

# Anforderungen

Bei der Anförderungsanalyse unterscheidet man zwischen funktionalen und nichtfunktionalen Anforderungen. Während funktionale Anforderungen den gewünschte Verhalten und die Funktionalität vorgeben, beschreiben nichtfunktionale Anforderungen Rahmenbedingungen wie Performance oder Zuverlässigkeit.

## 3.1 Funktionale Anforderungen

Ziel des System ist die Dokumentaustausch zwischen Partei von unterschiedliche Unternehmen zu gestalten, und der dabei relevant sicherheitmechanismus anzufertigen. Die folgenden funktionalen Anforderungen sollen dabei erfüllt wer- den.

### 3.1.1 Administratorrecht **ADMIN_ROLE**

- Der Administrator muss in der Lage sein, neue Benutzer im System hinzuzu- fügen und zu entfernen.
- Der Administrator darf nicht in der Lage sein Benutzer kritische Informationen zu modifizieren oder zu

### 3.1.2 Benutzerrecht **USER_ROLE**

- Der Benutzer kann eine Vertrauensbeziehung zu anderen Benutzern erstellen und sie wieder zerstören.
- Der Benutzer kann eine Gruppe erstellen und entfernen.
- Der Benutzer kann vertraute Benutzer in einer Gruppe hinzufügen und entfernen
- Der Benutzer kann den Zugriff auf seine Dateischlüssel an alle Mitglieder einer Gruppe freigeben und diese Freigabe auch wieder zurückziehen.

### 3.1.3 Registrierung und Login

Bei der Registrierung und Login dürfen keine Password oder Passphrase durch die Netz übertragen werden.

### 3.1.4 Data upload

unchiffrierte Datei dürfen nicht durch der Netz übertragen werden, da Datei auch als kritischen Daten gilt, müssen die vorab lokal chiffriert werden befor sie dann an RemoteServer geschickt werden.

### 3.1.5 graphische Zusammenfassung von funktionale Anforderungen

Dabei ist noch anschaulich das **LocalServer** und **RemoteServer** zwei unterschiedliche Softwaresystem sind die getrennt voneinander laufen. Noch bedeuntender ist es dass man der **RemoteServer** nur duch der **LocalServer** ansprechen kann.

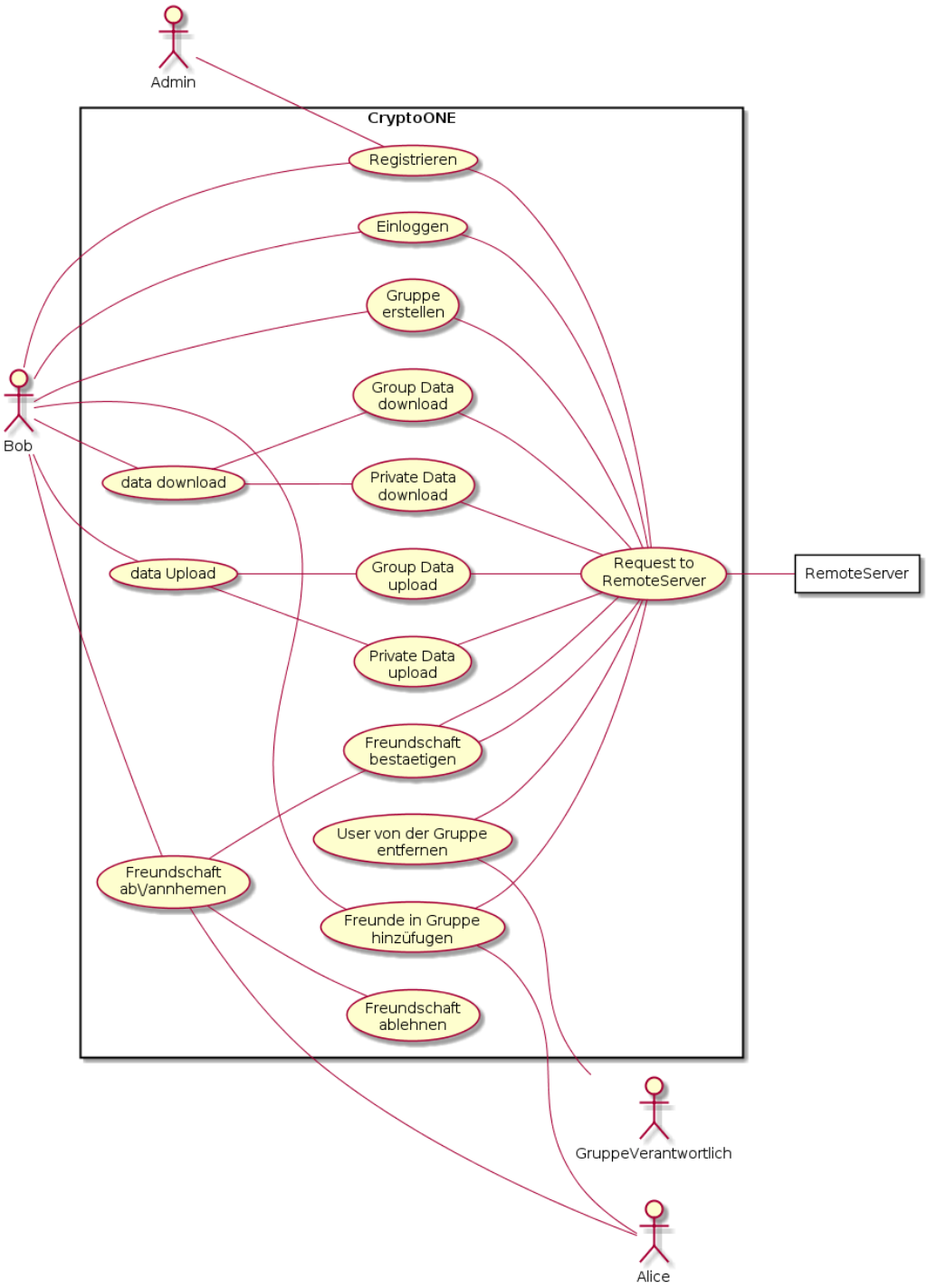


Figure 3.1: Funktionale Anforderungen

## 3.2 Nichtfunktionale Anforderungen

### 3.2.1 Overall nichtfunktional Anforungen

- Kein Einsatz von HTTPS
- RemoteServer darf **keine** Chiffrierung/Dechiffrierung durchführen
- LocalServer soll von ein USB-Stick getart werden, und soll auch von dort aus im hintergrund laufen.
- Benutzerinteraktion erfolgt durch ein Browser sodass keine zusätzliche Software erforderlich ist.

### 3.2.2 Wartbarkeit und Änderbarkeit

Die resultierende Software dieser Arbeit, soll in Zukunft gewartet, erweitert und geändert werden. Neu Features sind schon festgelegt ( sollen aber in der jetzige Version nicht implementiert werden )

### 3.2.3 Portierbarkeit und Plattformunabhängigkeit

Defakto ist der LocalServer portierbar, **LocalServer läuft auf USB-Stick** . Localserver soll auch plattformsunabhängig sein. Was RemoteServer angeht soll auch plattformunabhängig sein. Alle Einstellungen des Remoteserver müssen sich durch externe Konfigurationsdateien durchführen lassen.

### 3.2.4 Daten-und Serverintegrität

Der Benutzer soll in der Lage sein die Integrität von RemoteServer zu prüfen und der auf der letzer abgespeicherte Daten.

# Chapter 4

# konzept

## 4.1 Übersicht

## 4.2 Authentifizierung

Authenfizierung spielt systemweit eine bedeutende Rolle. Dabei passieren alle notwendige Prüfung von [LocalServer], [RemoteServer], [RemoteServer Integrität] und natürlich Authenfizierung von Benutzer.

Es durfte systemweit keine Einsatzt von Zertifikat/SSL-Verbindung oder Aufbau eine zustandbehaft Verbindung kommen, spricht die Kommunikationskanal ist unsicher.

Um Benutzercredentials von [LocalServer] auf [RemoteServer] zu übermitteln unter Anhaltung von Spezifikation, wurde ("SRP Secure Remote Password Protrocol," n.d.) Algorithmus wie im RFC2945 spezifiziert benutzt. Beim Einsatz von SRP-Secure Remote Password Protocol lasst sich auch einfach der gegenseitige Aunthenfizierung von [LokalServer] und [RemoteServer] realisieren, diese geschehet auch beim Authentifizierungsphase.

### 4.2.1 SRP-Secure Remote Password Protocol

[SRP] ermöglich es die Benutzercredentials zu übertragragen ohne dabei kritischen Informationen zu verraten. Beim Registrierung werden Benutzerspassword und Benutzername in eine **nichtzurückkehrbare** Operation zu eine korrespondierte Information berechnet nämmlich **Verifier**. Auch wenn eine Unbefugte die Verfier bekommt kann der nichts damit anfangen.

# Chapter 5

# Implementierung

Bei diese Abschnitt geht es um die konkrete Implementierung von der verschiedenen Softwareteils, nämmlich : * LocalServer * RemoteServer * CryptUtils * Frontend * und Inbetriebnahme-programm

## 5.1 Einleitung

Es wird als erste eine Unterkapitel über die wesentliche Technologien die für die Anfertigung des Projektes benötigt wurde, gefolgt von einer Beschreibung von die Technologie/Framework. Insbesondere wird Wert gelegt auf die Funktionalität von die Framework gelegt, die eine bedeutende Role in der Implementierung haben, und die Gewährleistung von relevante Sicherheitmechanismen zur Erfüllung die vorgegebene Anförderungen.

## 5.2 Überblick

| | Programmiersprache | Tehnologies/Framework | build-tool |
|---|---|---|---|
| CryptUtils | JAVA | JCE, Guava | Maven |
| LocalServer | JAVA | Spring, JCE, Guava | Maven |
| RemoteServer | JAVA | Spring, Hibernate, Guava, Spring-Security | Maven |
| Frontend | JavaScript, HTML, CSS | AngularJS, Bootstrap | Grunt |

## 5.3 Allgemein Designentscheidungen

### 5.3.1 JSON-Format

Systemweit wird JSON-Format bevorzugt um die Daten zwischen die verschiedene Softwarekomponente zu transpotieren. Explizit ausgedruckt, heisst es dass alle High-end Funktionen bzw. die Funktion die durch eine eine Softwarekomponent zur aussenwelt verfügbar gemacht wurden exportieren Daten in JSON-Format.

Diese Entscheidung lasst sich bei der Interoperabilität gründen, sowie auch Kriterien wie Einheitlichkeit von Softwareschnittstellen, was bei der Weiterentwicklung von grossen Bedeutung ist. Durch den Einsatz von JSON als Export-Format wird beispielerweise das Ersetzen von Softwarekomponent einfach.

Bei Einsatz von JSON wurde die von Google entwickelte (“GSON JSON Manipulation Framework,” n.d.)(“Gson”) Bibliothek benutzt.

### 5.3.2 UTF-8 und Base64 Encoding

Base64 ist mitte

### 5.3.3 Http Headers

Headers sind mächtige Standard wenn es zum Internet kommt, und wichtiger noch im Bereich Security von Webbasierte Anwendungen. Bei der Entwurf von dieser Arbeit, wurde die Entscheidung getroffen soviel wie möglich auf die Standard zu halten, insbesondere bei sicherheitrelevante Bereiche dieser Arbeit. **Die richtige Einstellung/Konfiguration von manche Headers tragen wesentlich bei, um der Sicherheitgrad eine Webanwendung zu erhöhen.**

Es wird noch mal über Headers die Rede sein, bei alle Softwareteil wo sie gesetzt werden ( LocalServer, RemoteServer, Frontend ), aber hier ist schon mal wichtig darüber zu erwähnen und eine gesamte Überblick über die Headers die Systemweit eingesetzt werden.

| Headername | Wert | gesetzt bei | Laufzeit | Anmerkungen |
|---|---|---|---|---|
| Content-Security-Policy | script-src 'self' | LocalServer | Erste Request an LocalServer | |
| Authorization | SRP | LocalServer | Erste Loginrequest | |
| WWW-Authenticate | SRP | LocalServer | Erste Loginrequest | |
| realm | realm | LocalServer | Erste Loginrequest | |
| hash-algorithm | SHA256 | LocalServer | Erste Loginrequest | |
| X-XSRF-TOKEN | X_XSRF_TOKEN | LocalServer | Erste Request an LocalServer | |
| AUTH-TOKEN | auth_token | RemoteServer | Nach erfolgreiche Authentifizierung | |
| EXPIRES-IN | expires_in | RemoteServer | Nach erfolgreiche Authentifizierung | |
| client-public-key | client_public_key | LocalServer | Erste Loginrequest | |

Figure 5.1: Headers

- (1) `Content-Security-Policy` spiel eine bedeutende Rolle um XSS-

Attack zu vermeiden. mit dem Wert `script-src 'self'` weist die Header hin, dass alle JavaScript source Datei nur von Server geladen werden dürfen. In unsere Fall von LocalServer.

- (2) (3) (4) und (5) Informieren den Webbrowser über dem Authentication Algorithmus bzw. dem Hash-Algorithmus, der eingesetzt wird.

## 5.4 CryptUtils ( Cryptographics Utilities )

Es handelt sich um Hilfbibliothek, die von [JCA] zur Verfügung gestellte kryptographische Methode abtrahieren. diese weitere Abstraktion führt zur eine einfache Benutzung von kryptographischen funktion wie die unterstehende beispiel zeigt.

### 5.4.1 Allgemein design

Symmetrische und asymmetrische Schlüsselerzeugung geschehen mithilfe zusätzliche Parametern um die Sicherheitgrad von Schlüssel zu erhöhen. Sicherheit ein Algorithmus, hängt nicht von der Algorithmus selbst, aber allein an die Stärke der Schlüssel Diese Parametern sind nämmlich :

- Salt
- Iteration bzw. Count ( Zähler )
- Password ( nur in der Fall von Passwordbasierte Schlüsselerzeugung )

Schlüssellänge sind standardmässige auf die maximale gelegt. Diese ist streng reglementiert. der Schlüssellänge kann aber durch der Client freikonfiguriert werden, in Bezug von Inlandregeln in der Bereich. Diese Bibliothek hält sich an der standard erlaubte Schlüssellänge. siehe unter stehende Tabelle ("JCA Java Cryptography Architectur Reference Guide," n.d.) zufolge.

| Algorithmus | max. Schlüssellänge |
|---|---|
| DES | 64 |
| DESede | * |
| RC2 | 128 |
| RC4 | 128 |
| RSA | * |
| alle andere | 128 |

Dritten Provider können anstatt von [JCE] benutzt werden.

Eintragname von exportierte JSON, verwendete Algorithmus, Schlüssellänge, lassen sich durch eine Konfigurationdatei konfiguriert. siehe beispiel-Konfigurationdatei [Anhang A] .

Alle exportierte Daten sind in JSON-Format und mit Base64 codiert.

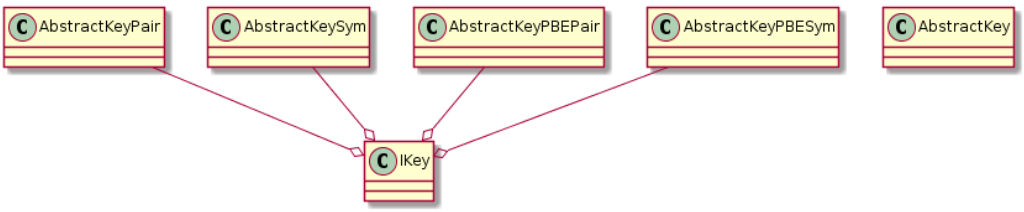


Figure 5.2: Schluessel-klasse-diagramm

### 5.4.2 Java Cryptography Architecture ( JCA )

### 5.4.3 Schlüsselerzeugung

#### 5.4.3.1 Klassendiagramm

### 5.4.4 Chiffrierung bzw. Dechiffrierung

#### 5.4.4.1 Klassendiagramm

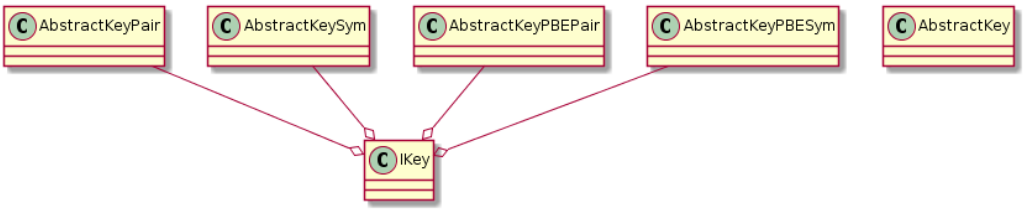


Figure 5.3: De/Chiffrierung-diagramm

```
String password = new String("mypassword");
String keypair = CryptUtils("PBE-RSA").generateKey( password );
```

CryptUtils-module wurde als Fabrik-Method-module und Dekorator entworfen.

Es wird eine Schnittstelle für die Erzeugung von Objekten definiert. Die Entscheidung, welche konkrete Klasse zu instanziieren, zu konfigurie- ren und schließlich zurückzugeben ist, wird konkreten (Unter-)Klassen überlassen, die diese Schnittstelle implementieren. [PatternKompakt]

## 5.5 Frontend

In diesem Kapitel handelt es sich um der Clientoberfläche in Form eine Webapplication, die der Endbenutzer auf irgendeinem Rechner der über eine Webbrowser fervügt aufrufen kann. Hier ist noch mal zu erinnern, dass eine der wichtige Anforderung von dieser Arbeit war das Software so konzipiert, dass es kein zusätzliche Softwareinstallation benötigt wird.

> Das Webapplication wurde in Form eine sog. `Single Page Application`, wie bereits erwärnt, handelt es sich um eine Technik Webseite zu entwerfen, so dass die Bedinung von Webapplication ähnlich ist wie von Benutzer schon bekannt Computersprogramm.
>
> Neben der Usability-und Portierungsargumente kommt auch die strenge Haltung von wichtigen Softwarearchiktekture und Regeln die mit Einsatz von ***AngularJS*** verbunden sind, nämmlich ***Separation of Concern*** und ***MVVM***.

### 5.5.1 AngularJS und Security

An der Webbrowser wird vorwiegend Angularjs eingesetzt. Was Sicherheit angeht, werden nämmlich den Angular Speicher strategie, **cookies** , die über den angular-service[1] einsetzbar ist.

1. Cookie speicher Configuration Object

[1] https://docs.angularjs.org/api/ngCookies/service/$cookies

- path
- domain
- expires
- secure

Was path und domain angeht wurden die Default Werte gelassen, und zwar Cookie steht zur Verfügung für aktuelle Pfad und alle untergeordnete Pfäder bzw. Cookie steht zur Verfügung nur für die Application domain.

War hier konfiguriert wurde war den Parameter **secure** mit den wert **true** und **expires** mit eine Date Instance in Form eine Zeichenkette, der konfigurierte die Lebensdauer der Cookies.

1.a. Unterschied zwischen mit truthy secure und falsy secure

Wie es zu sehen ist, kann man sehr leicht durch den Browser die in Cookie gespeicherte Daten sehen wenn secure nicht gesetzt ist. was im gegenteil nicht möglich ist wenn secure gesetzt ist.

### 5.5.2 [2] Cookie configuration object

```
var d = new Date( new Date().getTime() + 600000);
var n = d.toUTCString().toString();

var cookie_config ={
  secure : true,
  expires : n
};
```

[2] https://docs.angularjs.org/api/ngCookies/provider/$cookiesProvider#defaults

### 5.5.3 Ausstatung von Aktionen

Die unterstehende Grafik repräsentiert die mögliche Aktionen, die den Benutzer mithilfe der Webbrowser auslösen kann. Es lass sich dadurch nochmal eine graphische abstrahierende Darstellung von funktionale Anforderungen darstellen.

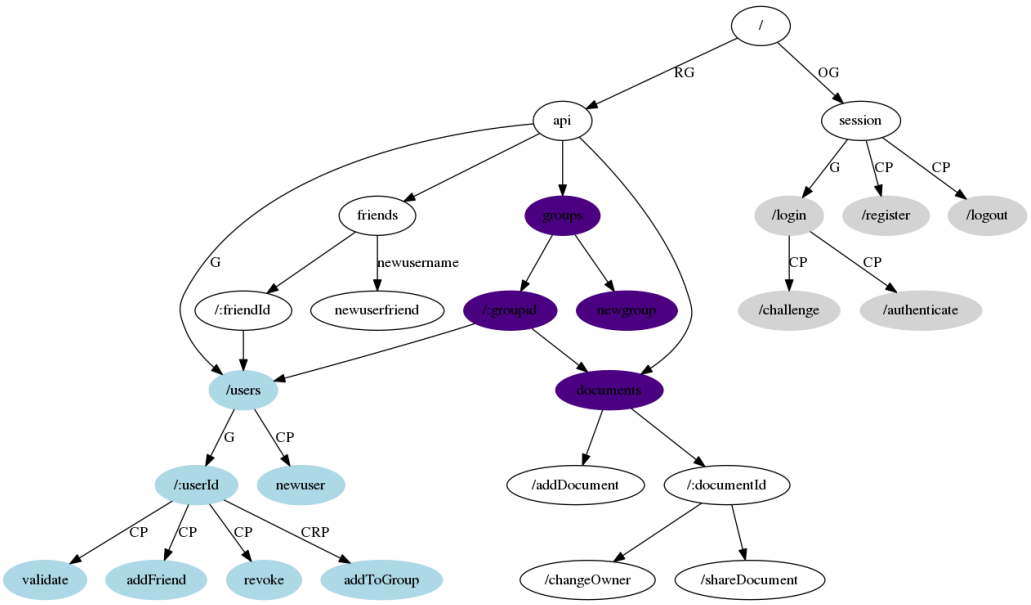


Figure 5.4: Aktionen

- G bzw. P : GET bzw. POST HTTP-methode
- C : Request mit kryptographische Aktion
- R(Registered) : Benutzer muss registriert sein.
- O(Open) : gegenteil zu R(Registered)

## 5.6 RemoteServer Implementierung

## 5.7 LocalServer Implementierung

# Chapter 6

# Test und Evaluation

This result was proved in [**?**].

This result was proved in [**?**].

winnt see (“MS Windows NT Kernel Description,” n.d.)

Blah blah (see “MS Windows NT Kernel Description,” n.d., 33–35; also 1963, ch. 1).

Cousteau1963

# References

Cousteau Jacques, and Dugan James. 1963. *The Living Sea: By Jacques-Yves Cousteau*. Book. London: Hamish Hamilton.

"GSON JSON Manipulation Framework." n.d. http://google.com/gson.

"JCA Java Cryptography Architectur Reference Guide." n.d. http://java.com/jca/jca_reference_guide.

"MS Windows NT Kernel Description." n.d. http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm.

"SRP Secure Remote Password Protrocol." n.d. http://tools.ietf.org/html/rfc2945.