Implementierung eines Keyserver

Kryptographie Praktikum

Hussam Saoud
FB Informatik
24.9.17

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Inhaltsverzeichnis

1 Einführung

In diesem Projekt geht es darum ein Konzept für eine Client/Server Applikation zu entwickeln und diese anschließend mithilfe von der Bibliothek *BouncyCastle* zu entwickeln.

Die Applikation soll sicher kryptographische Schlüssel generieren, managen, ablegen und übertragen. Dazu gehört auch ein Webservice für den Zugang zu den Schlüsseln.

2 Funktionsweise

Wenn der User noch nicht registriert ist, dann kann er die Applikation nicht verwenden. Nach dem Einloggen steht dem User die Wahl zwischen Generation oder falls bereits geschehen auch Download von Schlüsseln.

2.1 Generation von Schlüsseln

Wenn man nach dem Einloggen auf der Startseite auf *Generieren* geht, dann kann man zwischen drei verschiedenen Schlüsseltypen wählen:

- PGP Keypair/Certificate
- X.509-/SSL-Certificate
- just keypair

Bei allen Schlüsseltypen muss man eine Beschreibung für den zu erstellenden Schlüssel angeben.

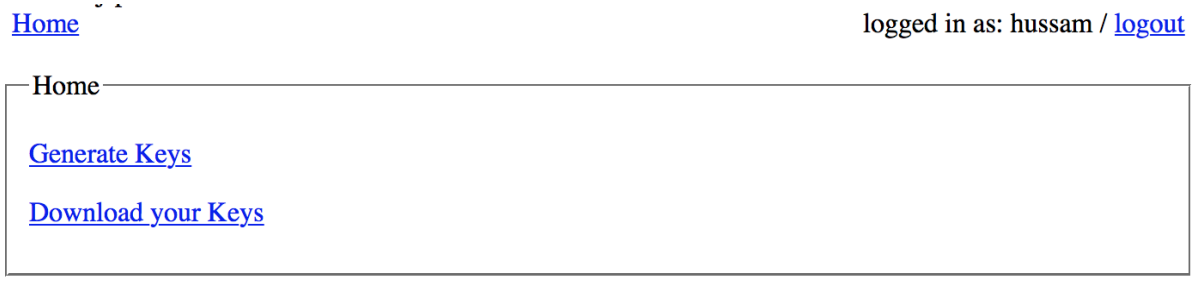

Bild 1: *Startseite mit Menüauswahl*

Je nachdem welche Option ausgewählt ist, erscheinen unterschiedliche Eingabefelder.
Für PGP ist das Eingabefeld „Passphrase“ erforderlich [Bild 2].

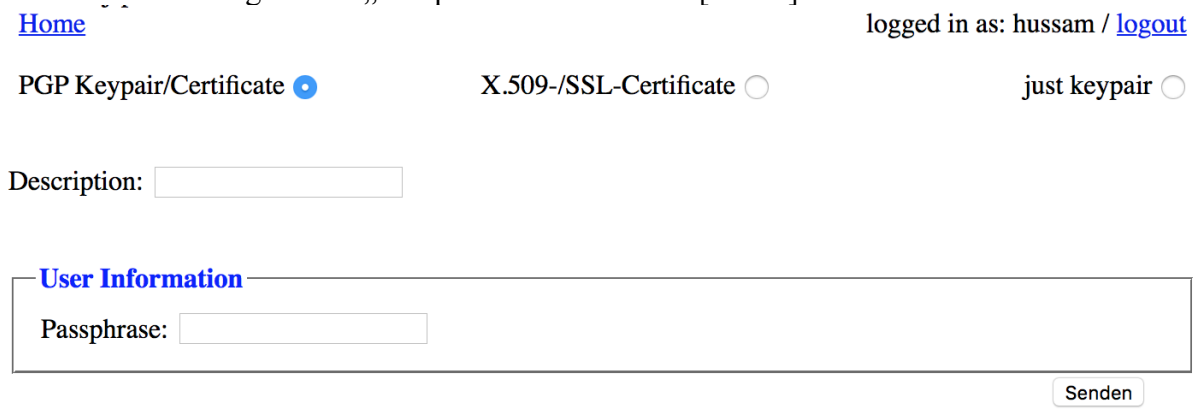

Bild 2: *Generierung von PGP-Schlüssel*

Für die Erstellung von X.509 Zertifikaten können mehrere Informationen angegeben werden. Wichtig ist die Information *Host Name* falls das Zertifikat für einen Server bestimmt ist [Bild 3]. In diesem Feld gehört der Domainname.

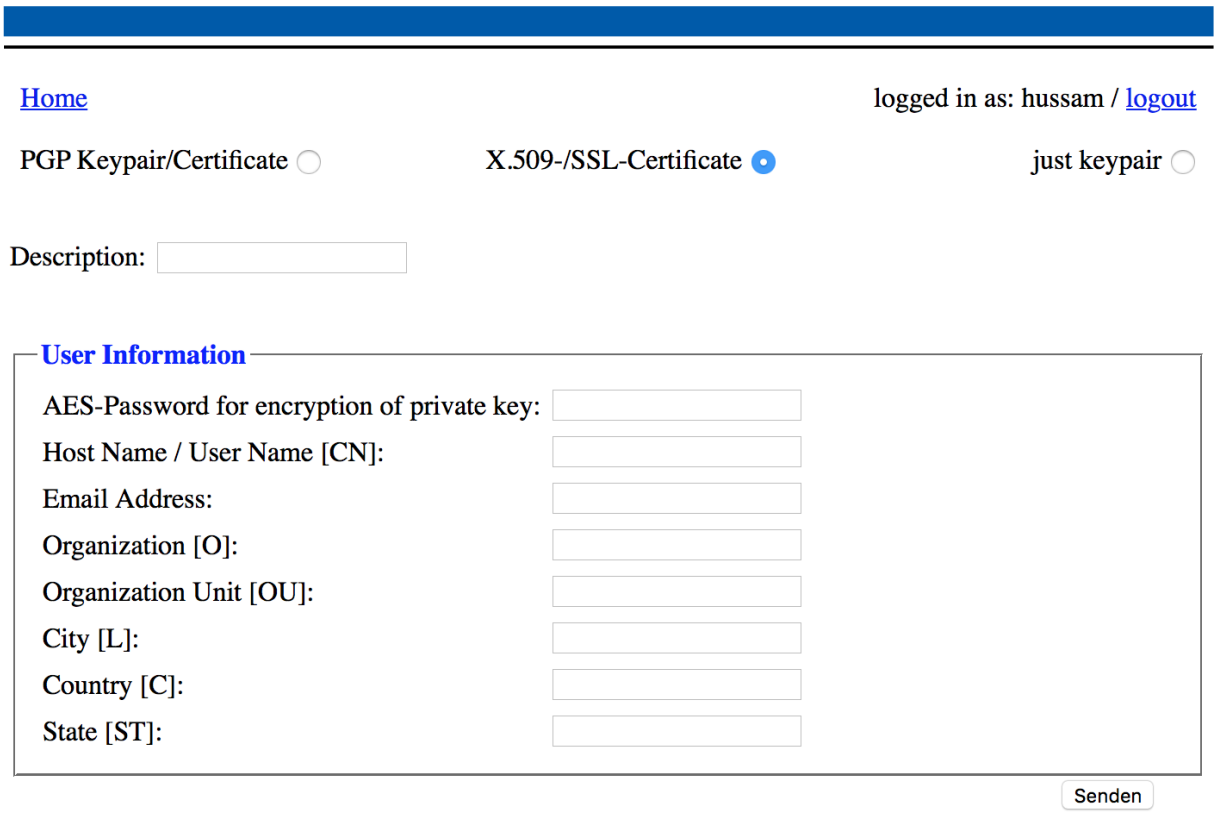

Bild 3: *Erstellung eines X.509 Zertifikats*

Der dritte Schlüsseltyp ist lediglich ein Public/Private Schlüsselpaar.

Wenn der Schlüssel erfolgreich generiert wurde, dann erscheint ein entsprechender Hinweis. Außer bei PGP wird bei allen Schlüsseltypen der private Schlüssel mit AES und 256 Bit in einem zip-Archiv verschlüsselt, den man mit jedem Archivierungsprogramm, das das *zip*-Format unterstützt nach Passworteingabe öffnen. Der private Schlüssel wird nach dem Archivieren zur Sicherheit gelöscht.

2.2 Download von Schlüsseln

Nachdem Schlüssel generiert wurden, können diese heruntergeladen werden. Hierzu wählt man im entsprechenden Menü die Schlüssel aus, die man herunterladen [Bild 4]. Es wird dann ein Link an die hinterlegte email-Adresse gesendet [Bild 5], über die man auf die Seite gelangt, auf der man schließlich die Schlüssel herunterladen kann [Bild 6].

Es ist jedoch aus Sicherheitsgründen nur vor Ablauf einer bestimmten Frist nach der Generierung des Schlüssels möglich, diesen herunterzuladen.

Home logged in as: hussam / logout

Date: 2017-07-24 15:57:03.0
Description: Ssl certificate
- PUBLIC
- PRIVATE
- PRIVATE

Date: 2017-07-26 21:15:58.0
Description: My key pair
- PUBLIC
- PRIVATE

Bild 4: *Anfordern von Keys zum Download*

```
hussam@praktikum-VirtualBox:~$ mail
"/var/mail/hussam": 1 Nachricht 1 ungelesene
>U   1 emailuser@praktiku Sa Sep 30 15:28   18/666   Requested Key
? n1
Return-Path: <emailuser@praktikum-VirtualBox>
X-Original-To: hussam
Delivered-To: hussam@praktikum-VirtualBox
Received: from macbook-pro-2.home (unknown [192.168.199.1])
        by praktikum-VirtualBox (Postfix) with ESMTP id 517C0C5E34
        for <hussam>; Sat, 30 Sep 2017 15:28:21 +0200 (CEST)
From: emailuser
To: hussam
Message-ID: <126932634.0.1506778101215@macbook-pro-2.home>
Subject: Requested Key
MIME-Version: 1.0
Content-Type: text/html; charset=utf-8
Content-Transfer-Encoding: 7bit
X-IMAPbase: 1506619978 2
Status: O
X-UID: 1

Here is the link for your requested Keys: https://localhost:8443/CryptoServer2/D
ownloadKeysForm.jsp?myurl=6e9b40fcebf5e1b98ebbd8c893216387
```

Bild 5: *email mit Link zur Downloadseite*

Home logged in as: hussam / logout

- Date: 2017-07-24 15:57:03.0
  Description: Ssl certificate

  PUBLIC
  PRIVATE
  PRIVATE

Bild 6: *Seite mit den angeforderten Schlüsseln zum Download*

3 Installation

Zur Entwicklung und Ausführung des Servers und Clients wurde Eclipse für Web-Entwickler[1] verwendet. Als Webserver wurde *Tomcat* verwendet[2], der als Plug-In in Eclipse eingebunden werden muss. Als Datenbank diente *MySQL*[3].
Für die Versendung von emails an den Client zum Download der Schlüssel kam Postfix[4] zum Einsatz, der auf einem Linux-System lief.

3.1 Eclipse für Web-Entwickler

Nach der Entwicklung der Entwicklungsumgebung Eclipse muss ein neues sogenanntes „Dynamic Web Projekt“ erstellt werden. In der Ordnerstruktur des Projekts werden neue Java-Servlets mit Java-Code und *.java*-Endung im Ordner *src* abgelegt. Clientseitiger Code wie *.jsp* (Java Server Pages), *.html, .js (Java-Script) und .css* Dateien werden dagegen im Ordner *WebContent* abgelegt.

Zur Einbindung von Bibliotheken muss zuerst unter *WebContent→WEB-INF* der Ordner *lib* erstellt, falls noch nicht vorhanden, und die Bibliothek-Dateien mit *.jar*-Endung in den Ordner kopiert werden und dann mit Rechtsklick auf der Bibliothek *Build Path→ Add to Build Path* auswählen.

Das *Tomcat*-Plug-In kann in Eclipse unter *Projekt→ Eigenschaften→Targeted Runtimes* installiert werden. Vorher muss es aber heruntergeladen werden.

Es muss noch der private Schlüssel *server_private.key* vom Tomcat-Server, der im Verzeichnis *CryptoServer2* abgelegt ist auf den lokalen System übertragen werden und der Pfad in der Methode *getPrivateKey* der Klasse *GenerateKeys* angepasst werden

3.2 MySQL-Server

Um den MySQL-Server zu verwenden, muss er lediglich installiert und je nach Betriebssystem noch die Pfad-Variablen gesetzt werden. Der Server wird in der Regel automatisch bei Neustart gestartet und ist nach der Installation sofort verfügbar.

In der Klasse *MyDB* muss der Benutzername und das Passwort für die Datenbank evtl angepasst werden. Es muss eine Datenbank mit dem Namen *MyDB* in MySQL erstellt werden.

Es müssen in dieser Datenbank folgende Tabellen erzeugt werden, damit die Applikation funktioniert.

```
CREATE TABLE `user` (
  `name` varchar(60) DEFAULT NULL,
  `email` varchar(60) DEFAULT NULL,
  `password` varchar(100) DEFAULT NULL,
  `id` int(6) unsigned NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`)
)
```

1 http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/neon3
2 https://tomcat.apache.org/download-90.cgi
3 https://www.mysql.com/de/downloads/
4 http://www.postfix.org/download.html

```
CREATE TABLE `mykeys` (
  `id` int(6) unsigned NOT NULL AUTO_INCREMENT,
  `title` varchar(35) DEFAULT NULL,
  `filename` varchar(200) DEFAULT NULL,
  `user_id` int(6) unsigned DEFAULT NULL,
  `date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
  `keytype` varchar(15) DEFAULT NULL,
  `keypart` varchar(15) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `user_id` (`user_id`),
  CONSTRAINT `mykeys_ibfk_1` FOREIGN KEY (`user_id`)
REFERENCES `user` (`id`)
)


CREATE TABLE `requestedkeys` (
  `id` int(6) NOT NULL,
  `mykeys_id` int(6) unsigned DEFAULT NULL,
  KEY `mykeys_id` (`mykeys_id`),
  CONSTRAINT `requestedkeys_ibfk_1`
FOREIGN KEY (`mykeys_id`) REFERENCES
`mykeys` (`id`)
)

CREATE TABLE `url` (
  `id` int(6) unsigned NOT NULL
AUTO_INCREMENT,
  `url` varchar(100) DEFAULT NULL,
  `expire_date` timestamp NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
)
```

3.3 Email-Server Postfix

Postfix ist für Linux verfügbar. Nach der Installation von Postfix sind für die einfache Verwendung nur wenige Konfigurationen nötig. In der *main.cfg* Datei im Verzeichnis */etc/postfix* muss unter *mynetworks* noch die erlaubten Netzwerke angegeben werden [Bild 7]. In dem vorliegenden Entwicklungsumgebung war es u.a. das Netzwerk 10.255.0.0/24.

```
mydestination = $myhostname, myemailserver, praktikum-VirtualBox, localhost.loca
ldomain, localhost
relayhost =
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128 10.255.0.0/24 10.255.0
.0/16
mailbox_size_limit = 0
recipient_delimiter = +
inet_interfaces = all
inet_protocols = all
```

Bild 7: *main.cfg Datei für die Konfiguration des Email-Servers postfix*

Zusätzlich müssen die Benutzerkonten bzw. Email-Adressen auf dem System erstellt werden. Jedes Benutzerkonto wird unter Linux folgendermaßen erstellt:

```
sudo adduser -s /sbin/nologin [username]        #Konto erstellen
sudo passwd [username]                          #Passwort für das Konto setzen
```

Im Projekt wurde *emailuser* mit Passwort *emailuser* verwendet. Wenn ein anderer Benutzername verwendet werden soll, dann muss es im Code in der Klasse *email.SendEmail* angepasst werden.

In der selben Klasse *email.SendEmail* muss auch die IP-Adresse des email-Servers entsprechend angepasst werden

3.4 Apache Tomcat Server

Um SSL auf dem Apache-Tomcat-Server zu unterstützen, muss die Server.xml Datei vom Server der unter abschnitt 3.1 installiert wurde bearbeitet werden und folgender Code hinzugefügt werden:

```
<Connector SSLEnabled="true" clientAuth="false"
keystoreFile="/Users/admin/Documents/tomcat.ks"
keystorePass="thehorse" maxThreads="150" port="8443"
protocol="HTTP/1.1" scheme="https" secure="true"
sslProtocol="TLS"/>
```

Dabei muss die keystore-Datei *tomcat.ks* auf dem lokalen System abgespeichert werden und der Pfad in dem Code unter *keystoreFile* angepasst werden. Der Server Tomcat befinder sich in der Regel im workspache-Verzeichnis, wo der restliche Code abgelegt ist. Die keystore-Datei ist unter CryptoServer2 zu finden.

3.5 Testen

Um die Applikation zu verwenden muss der MySQL-Server, der Email-Server Postfix im Hintergrund laufen. In Eclipse muss die Index-Datei mit dem Hauptmenü index.jsp geöffnet und dann unter Run→ run die Applikation gestartet werden. Der Tomcat-Server startet dann automatisch und es öffnet sich die Hauptmenü-Seite im Eclipse-Fenster.

4 Implementierung

Die Implementierung ist in Client- und Server-Seite geteilt. Zum Server gehören die Java-Servlets, die die Anfragen der User bearbeiten. Zum Client gehören die .html, .jsp, .css und .js Dateien, die die Benutzeroberfläche für den User darstellen.

4.1 Server-Seite

Java-Servlets bearbeiten die HTTP-Anfragen vom User die per GET oder POST versendet werden. In diesem Projekt gibt es für verschiedene Anfragen eigene Servlets und teilweise Java-Klassen als Hilfsklassen für die Servlets.

4.1.1 Login/Registrierung

Im Servlet *RegisterServlet* werden die eingegebenen Registrierungs-Informationen bearbeitet und falls die Eingaben korrekt sind, in die Datenbank geschrieben.

Im Servlet *LoginServlet* werden die eingegebene email-Adresse und Passwort überprüft und falls diese korrekt sind, dann wird der Benutzer eingeloggt und eine Session samt Session-ID für den User erzeugt, die bis zum Ausloggen erhalten bleibt und worüber der User identifiziert werden kann.

4.1.2 Schlüssel generieren

Im Servlet *GenerateKeysServlet* werden die gewünschten Schlüsseln generiert. Nachdem die Session-ID des Users abgerufen wird, werden die angegebenen Daten geprüft und anhand der Daten die gewünschten Schlüsseln erzeugt.

Je nachdem welcher Schlüsseltyp angefordert wurde, wird eine spezielle Java-Klasse verwendet, da die Schlüsseltypen sich hinsichtlich des Algorithmus unterscheiden. Bei PGP wird die Java-Klasse *GeneratePGPKeys,* bei einfachem Schlüsselpaar und X.509 *GenerateKeys* aufgerufen.

4.1.3 Schlüssel anfordern

Bevor Schlüssel nach dem Generieren heruntergeladen werden können, müssen diese angefordert werden. Man erhält eine email mit der URL die zur Download-Seite führt.
Nachdem in der User in der jsp-Datei *RequestKeysForm* die gewünschten Schlüssel angefordert hat, wird das Servlet *ProcessRequestedKeys* aufgerufen.

In diesem Servlet wird die URL zufällig generiert und das Verfallsdatum auf 1 Stunde nach der Anforderung gesetzt. Anschließend wird über die Java-Klasse *SendEmail* eine email an den User mit dem Link gesendet. In *SendEmail* wird der Absender als *emailuser* gesetzt. In der Variablen *outgoingMailServer* wird die IP-Adresse des email-Server gesetzt.

4.1.4 Schlüssel downloaden

Beim Download von Schlüsseln nach Auswahl *DownloadKeysForm* wird vor der eigentlichen .jsp-Seite *DownloadKeysForm* zuerst das Filter-Servlet *DownloadKeysFilter* aufgerufen, um zu prüfen ob User angemeldet ist und ob er berechtigt ist die per email versendete URL für den Download von Schlüsseln aufzurufen.

Wenn Filter negativ war, dann wird der User zur Login-Seite weitergeleitet, ansonsten wird *DownloadKeysForm* aufgerufen, wo der User die angeforderten Schlüssel herunterladen kann.

Wenn der User dann auf die Links zum Herunterladen geklickt hat, dann wird das Servlet *DownloadKeysServlet* aufgerufen. Dieses Servlet lokalisiert die Dateien und überträgt diese über einen Stream-Buffer zum User.

4.1.5 Datenbank-Zugriff

In der statischen Java-Klasse *MyDB* werden in verschiedenen Methoden SQL-Befehle an die Datenbank ausgeführt um Datensätze anzufordern.

Im Folgenden sind die wichtigsten Methoden erklärt:

- *InsertKeyToDB*: Beim Generieren eines Schlüssels wird diese Methode aufgerufen, um die Informationen zum Key in die Datenbank-Tabelle *Mykeys* zu schreiben. Zu den Informationen gehören der Dateipfad zu den Schlüssel werden Informationen zum Key Es wird ein *MyKey*-Objekt übergeben mit Informationen wie Dateipfad des Schlüssels, Generierungsdatum, Beschreibung, Schlüsseltyp und -teil.

- *InsertRequestedKeys*: Beim Anfordern von Schlüssel zum Download wird diese Methode aufgerufen. In dieser Methode wird zuerst die Methode insertIntoURLTable aufgerufen, um die URL mit dem Verfallsdatum in die Tabelle *ULR* zu schreiben. Dann wird in einer Schleife die Methode *insertIntoRequestedKeysTable* aufgerufen , um die Informationen zu den angeforderten Schlüssel in die Datenbank zu schreiben.

- *CheckIfURLBelongsToUser*: In dieser Methode wird geprüft ob der angemeldete User berechtigt ist, die URL für den Download von angeforderten Schlüsseln aufzurufen.

- *CheckUser*: In dieser Methode wird der Login geprüft.

- *InsertUser:* In dieser Methode wird ein neuer Benutzer in die Datenbank hinzugefügt.

4.2 Client-Seite

Für die UI wurde das Web-Framework JSP mit JavaScript und CSS verwendet. JSP erlaubt die optische Gestaltung von Webseiten mit HTML und die Einbindung Java-Code. Es wurde dabei JSTL verwendet, um Datenbankzugriffe zu erleichtern und Java-Code einzubinden.

In allen JSP-Dateien wird die JSP-Datei Login.jsp eingebunden, mit deren Hilfe immer zuerst der Login-Status abgefragt wird und im negativen Fall das Login-Servlet ausgeführt wird.

4.2.1 Schlüssel generieren

In der JSP-Datei *GenerateKeysForm* werden dem User drei Schlüsseltypen zur Auswahl angeboten. Je nachdem welchen Typ er auswählt, werden ihm dynamisch mithilfe von JavaScript andere Eingabe-Felder angezeigt. Hierfür wurde die JavaScript-Datei *generateKeys2.js* eingebunden. Als JavaScript-Framework wurde JQuery verwendet.

In der JavaScript-Code wird beim Absenden der HTML-Form geprüft welcher Schlüsseltyp ausgewählt wurde und an das Servlet *GenerateKeysServlet* weitergegeben, damit die Informationen entsprechend des Schlüsseltyps verarbeitet werden.

4.2.2 Schlüssel anfordern

In der JSP-Datei *RequestKeysForm* werden die Schlüssel vom angemeldeten User angezeigt, damit er die gewünschten Schlüssel auswählen und anfordern kann. Dabei werden die Daten aus der Datenbank-Tabelle *mykeys* geladen. Nach dem Absenden der Form das Servlet *ProcessRequestedKeys* aufgerufen, die die Daten entgegennimmt und bearbeitet.

4.2.3 Admin-Panel

Das Admin-Panel ist ganz simpel gestaltet. Die User-ID des Admin-Users wurde auf 1 festgelegt. Wenn der Admin-User sich anmeldet, dann bekommt er im Hauptmenü das Admin-Panel angezeigt. Im Admin-Panel kann der Admin sich das Protokoll von Schlüsselgenerationen und -downloads anzeigen lassen. Darin wird nach der Zugriffszeit sortiert der User, die Zugriffszeit und Schlüsselinformationen aufgelistet.

Für die beiden Informationstypen gibt es die JSP-Dateien *AdminPanelFormGeneratedKeys* and *AdminPanelFormDownloadedKeys*. In beiden Dateien werden zuerst die Daten aus der Datenbanktabelle geladen, damit diese in einer HTML-Tabelle angezeigt werden können. Vor dem Laden der Datei wird zuerst über das Servlet-Filter „AdminPanelFilter“ geprüft ob auch der Admin mit der User-ID 1 angemeldet ist. Im negativen Fall wird auf die Login-Seite weitergeleitet.

Bild 8 zeigt die Ausgabe für die Generierungen von Schlüsseln. Bild 9 zeigt die Ausgabe für die Downloads, die etwas detailierter ist.

Home logged in as: Admin / logout

Admin Panel

Generated Keys / Downloaded Keys

Generated Keys

Date	Name	Title	Keytype
2017-09-28 19:53:04.0	hussam	Newest2	PGP
2017-09-28 19:40:45.0	hussam	newest key	PGP
2017-09-27 18:08:01.0	hussam	6:06 pm key	509
2017-09-27 18:02:21.0	hussam	6:02 pm key	PGP
2017-09-27 18:00:45.0	hussam	6 pm key	PGP
2017-09-27 16:41:06.0	hussam	27.9. key	PGP

Bild 8: *Admin-Panel für Generierung der Schlüssel*

Home logged in as: Admin / logout

Admin Panel

Generated Keys / Downloaded Keys

Downloaded Keys

Date	Name	Title	Keytype	Keypart
2017-09-30 16:28:19.0	hussam	Ssl certificate	509	PRIVATE
2017-09-30 16:28:19.0	hussam	Ssl certificate	509	PRIVATE
2017-09-30 16:28:19.0	hussam	Ssl certificate	509	PUBLIC
2017-09-30 16:27:18.0	hussam	Ssl certificate	509	PRIVATE
2017-09-30 16:27:18.0	hussam	Ssl certificate	509	PRIVATE

Bild 9: *Admin-Panel für Downloads von Schlüsseln*

4.3 Verwendete Bibliotheken

Für das Projekt wurden einige Bibliotheken verwendet, die für die einzelnen Funktionen wie Email-Versand, Datenbank-Anbindung, Archivierung von Dateien und natürlich der Verschlüsselung mittels BouncyCastle notwendig waren.

Die wesentlichen Bibliotheken werden in den folgenden Unterkapiteln vorgestellt

4.3.1 BouncyCastle

Mithilfe von BouncyCastle lassen sich u.a. verschiedene asymmetrische Verschlüsselungsverfahren verwenden. In diesem Projekt werden die Verfahren zur Generierung eines PGP-Schlüssels, X.509-Zertifikats und eines einfachen Public-/Private-Schlüsselpaars verwendet.

Für die PGP-Verschlüsselung wird die Klasse *GeneratePGPKeys* verwendet. In dieser Klasse wird für die PGP-Verschlüsselung zuerst die Methode *init* aufgerufen, worin wiederum die interne Methode generateKeyRingGenerator aufgerufen wird, um ein *PGPKeyRingGenerator*-Objekt mit der email-Adresse des Users als *Identity* und das eingegebene Passwort als Parameter zu erzeugen und zurückzugeben. In dieser Methode wird dann folgendermaßen vorgegangen:

(1) *RSAKeyPairGenerator*-Objekt *kpg* erzeugen und mit mit 2048 Bit Schlüsselgröße initialisieren

(2) Master/Signing-Schlüsselpaar mit dem Generator aus (1) erzeugen und dann Encryption-Subkey-Schlüsselpaar

(3) *PGPSignatureSubpacketGenerator*-Objekt für das Signieren von Hash erzeugen und Meta-Daten zuweisen wie:
   (a) Aufgaben der *Signature* wie das Signieren und Zertifizieren von Daten festlegen
   (b) Bevorzugter symmetrische Algorithmus und Hash-Algorthmus
   (c) Aufforderung an Sender zusätzlich noch Checksumme zur Nachricht hinzufügen

(4) *PGPSignatureSubpacketGenerator*-Objekt für das Verschlüsseln von Hash erzeugen und Meta-Daten zuweisen wie:
   (a) Aufgaben der *Signature* wie das Signieren und Zertifizieren von Daten festlegen

(5) Zwei *PGPDigestCalculator*-Objekte erzeugen jeweils für das *PBESecretKeyEncryptor*-Objekt und den PGPKeyRingGenerator.

(6) *PBESecretKeyEncryptor*-Objekt erzeugen mit dem vorher erzeugten *PGPDigestCalculator*-Objekt und symmetrischen Verschlüsselungsalgorithmus für die Daten als Parameter.

(7) *PGPKeyRingGenerator*-Objekt erzeugen und u.a. mit dem zuvor in (6) erzeugten *PBESecretKeyEncryptor*-Objekt und *PGPDigestCalculator*-Objekt und Master/Signing-Schlüsselpaar als Parameter.

(8) Zuletzt wird zum in (7) erzeugten *PGPKeyRingGenerator*-Objekt das in (4) erzeugte *PGPSignatureSubpacketGenerator*-Objekt und in das in (2) erzeugte Encryption-Subkey-Schlüsselpaar hinzugefügt und dieses auch zurückgegeben. Es wird also ein *PGPKeyRingGenerator*-Objekt zurückgegeben.

Für die Erzeugung eines einfachen Schlüsselpaars und X.509-Zertifikats wird die Klasse *GenerateKeys* verwendet.

Das Zertifikat wird folgendermaßen erzeugt:

(1) Aufruf der Methode makeCertifikate mit dem neu erzeugten Public/Private-Schlüsselpaar und die Zertifikatsinformationen wie Hostname und User-Informationen als Parameter

(2) In der Methode wird das aktuelle Datum und Verfallsdatum erzeugt. Dann wird ein X509v3CertificateBuilder-Objekt mit allen Zertifkatsinformationen und dem Public-Key des Users erzeugt

(3) Ein *ContentSigner*-Objekt wird mit dem private-Key des Servers als Parameter erzeugt.

(4) Am Ende wird mithilfe von *CertificateFactory*-Objekt das Zertifikat erzeugt.

Um ein einfaches Public/Private Schlüsselpaar zu erzeugen wird die Methode *generateKeyPair* in der Klasse *GenerateKeys* aufgerufen. Dort muss als erstes der BouncyCastle-Provider als Security-Provider eingestellt werden und dann wird mit der Klasse KeyPairGenerator eine Instanz mit den Parametern für RSA und BouncyCastle („BC“) erzeugt und hinterher mit der Schlüssellänge von z.B. 4096 Bit initialisiert. Zum Schluss wird das Schlüsselpaar generiert und zurückgegeben.

4.3.2 Zip4J

Mit der Bibliothek Zip4J werden in diesem Projekt mit AES verschlüsselte Archive erzeugt, um private Schlüssel verschlüsselt ablegen zu können.

In der Klasse GenerateKeys in der Methode writeEncryptedZipFile wird die Bilbiothek verwendet.

Als erstes wird ein *ZipFile*-Objekt mit dem Dateipfad für das zu erstellende Archiv als Parameter erzeugt. Dann wird ein *ZipParameters*-Objekt erzeugt mit den einzelnen Eigenschaften. In diesem Fall wurde AES mit 256 Bit verwendet. Außerdem muss die Methode *setEncryptFiles* vom *ZipParameters*-Objekt auf true gesetzt und zusätzlich das Passwort zugewiesen werden.

5 Fazit und Ausblick

Die Applikation funktioniert nach derzeitigem Stand einwandfrei und uneingeschränkt. Hinsichtlich des Funktionsumfangs und des Sicherheitsmaß lässt sich natürlich einiges erweitern. Wegen der eingeschränkten Zeit musste auf die wesentlichen Sicherheitsfunktionen fokusiert werden.

Aktuell unterstützt die Applikation zusätzlich zur Login-Funktion unter anderem folgende Sicherheitsfunktionen:

(1) Die Kommunikation zwischen Client und Server ist durch SSL abgesichert

(2) Die Privaten Schlüssel werden mit AES und 256 Bit verschlüsselt als Zip-Archiv auf dem Server abgelegt

(3) User kann sein Schlüssel nicht direkt nach dem Generieren herunterladen, sondern er bekommt einen Link per email zugesandt, worüber er auf die Download-Seite gelangt

(4) Download-Link wird zufällig generiert

(5) Die Namen User-Verzeichnisse, worin die Schlüssel der einzelnen User abgelegt werden, entsprechen dem Hash der User-ID, sodass ein Unbefugter vom Verzeichnisnamen her nicht sagen kann, zu welchem User die Verzeichnisse gehören.

(6) Nach dem Generieren und dem Versand der URL per email hat der User nur in einer bestimmten Zeitspanne die Gelegenheit die Schlüssel herunterzuladen, bevor die URL ungültig wird.

(7) Es wird durch Servlet-Filter bei jedem Seitenzugriff geprüft ob der User eingeloggt ist und berechtigt ist die Seite zu besuchen oder den Download-Link aufzurufen. Dafür gibt es für Login den LoginFilter, für den Zugriff auf das Admin-Panel das AdminPanel-Filter und für den Zugriff auf die Download-Seite den DownloadKeys-Filter.

(8) Durch ein Admin-Panel kann der Administrator verdächtiges Verhalten erkennen und entsprechend reagieren

Trotz all dieser Sicherheitsfunktionen und -maßnahmen kann die Applikation trotzdem weiter sicherheitsmäßig ausgebaut werden.

Was den Funktionsumfang betrifft, so könnte das Admin-Panel erweitert werden um eine Suchfunktion mit Filter, um gezielter nach Protokollinformationen suchen zu können.

Was die Sicherheitsfunktionen betrifft, so können diese um einige weitere Funktionen erweitert werden.

Z.b. werden beim Registrieren und Einloggen zwar alle Daten, also auch Passwort, per SSL vom Client zum Server verschlüsselt übertragen, aber das Passwort sollte für zusätzliche Sicherheit trotzdem in ein Hash umgewandelt werden und als solche in der Datenbank abgelegt werden. Dafür muss auf der Client-Seite beim Absenden der Login- oder Registrierungsdaten aus Passwort ein Hash generiert und dieser Hash als Passwort an den Server abgesendet werden.

Nach einer bestimmten Frist nach dem Anfordern der Schlüssel durch den User sind diese nicht mehr gültig und der User kann nicht mehr darauf zugreifen. Es müsste überlegt werden

ob die Schlüssel auch hinterher ganz aus dem System gelöscht werden sollen oder nicht. Der private Schlüssel ist jedenfalls als Zip verschlüsselt abgelegt und damit abgesichert. Es wäre aber eine weitere Sicherheitsmaßnahme, es sei denn die User dürften mehrmals Schlüssel anfordern.

Der private Schlüssel des Tomcat-Servers sollte am besten wie die privaten Schlüssel der User verschlüsselt werden und nur beim Zugriff durch die Applikation entschlüsselt werden. Dieser wäre sonst nur durch die Firewall des Systems gesichert und die Verschlüsselung wäre ein zusätzlicher Schutz.

Zusätzlich zu dem Erwähnten kann noch eine Post-Quantum-Verschlüsselung für die Kommunikation zwischen Client und Server eingerichtet werden.