JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Java Security

Systemtechnik Labor
5YHITM 2016/17, Gruppe A

Maximilian Seidl

Version 0.1
Note:
Betreuer: Th.Micheler
Begonnen am 21.10.2016
Beendet am 21.10.2016

Inhaltsverzeichnis

1 Einführung

Diese Übung zeigt die Anwendung von Verschlüsselung in Java.

1.1 Ziele

Das Ziel dieser Übung ist die symmetrische und asymmetrische Verschluesselung in Java umzusetzen. Dabei soll ein Service mit einem Client einen sicheren Kommunikationskanal aufbauen und im Anschluss verschlüsselte Nachrichten austauschen. Ebenso soll die Verwendung eines Namensdienstes zum Speichern von Informationen (hier PublicKey) verwendet werden.

Die Kommunikation zwischen Client und Service soll mit Hilfe einer Übertragungsmethode (IPC, RPC, Java RMI, JMS, etc) aus dem letzten umgesetzt werden.

1.2 Voraussetzungen

- Grundlagen Verzeichnisdienst
- Administration eines LDAP Dienstes
- Grundlagen der JNDI API für eine JAVA Implementierung
- Grundlagen Verschluesselung (symmetrisch, asymmetrisch)
- Einfuehrung in Java Security JCA (Cipher, KeyPairGenerator, KeyFactory)
- Kommunikation in Java (IPC, RPC, Java RMI, JMS)
- Verwendung einer virtuellen Instanz für den Betrieb des Verzeichnisdienstes

1.3 Aufgabenstellung

Mit Hilfe der zur Verfuegung gestellten VM wird ein vorkonfiguriertes LDAP Service zur Verfügung gestellt. Dieser Verzeichnisdienst soll verwendet werden, um den PublicKey von einem Service zu veröffentlichen. Der PublicKey wird beim Start des Services erzeugt und im LDAP Verzeichnis abgespeichert. Wenn der Client das Service nutzen will, so muss zunaechst der PublicKey des Services aus dem Verzeichnis gelesen werden. Dieser PublicKey wird dazu verwendet, um den symmetrischen Schluessel des Clients zu verschlüsseln und im Anschluss an das Service zu senden.

Das Service empfängt den verschlüsselten symmetrischen Schluessel und entschlüsselt diesen mit dem PrivateKey. Nun kann eine Nachricht verschlüsselt mit dem symmetrischen Schlüssel vom Service zum Client gesendet werden.

Der Client empfängt die verschlüsselte Nachricht und entschlüsselt diese mit dem symmetrischen Schlüssel. Die Nachricht wird zuletzt zur Kontrolle ausgegeben.

Die folgende Grafik soll den Vorgang verdeutlichen:

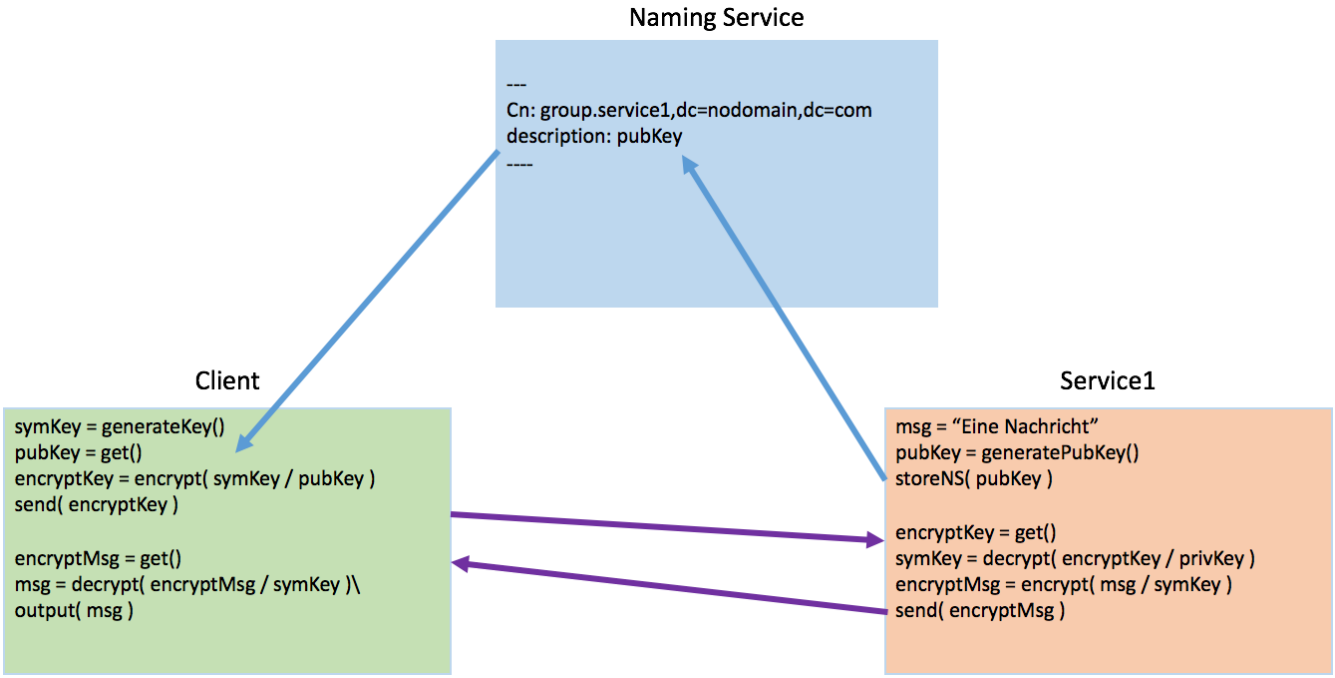

Abbildung 1: grafische Veranschaulichung von der Aufgabe

Gruppengrösse: 1 Person

Bewertung: 16 Punkte

- asymmetrische Verschluesselung (4 Punkte)
- symmetrische Verschluesselung (4 Punkte)
- Kommunikation in Java (3 Punkte)
- Verwendung eines Naming Service, JNDI (3 Punkte)
- Protokoll (2 Punkte)

Links: - Java Security Overview
`https://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html`
- Security Architecture
`https://docs.oracle.com/javase/8/docs/technotes/guides/security/spec/security-spec.doc.html`
- Java Cryptography Architecture (JCA) Reference Guide
`https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html`

Read the Java Security Documentation and focus on following Classes: KeyPairGenerator, SecureRandom, KeyFactory, X509EncodedKeySpec, Cipher

2 Ergebnisse

Es standen ein paar Beispielklassen zur Verfügung, welche von Moodle heruntergeladen wurden. Darin befanden sich zwei Klassen, eine davon beschrieb die den LDAP Connector (wird später noch im Protokoll erwähnt) und die andere erklärte ein paar Code-Snippets zur Verschlüsselung und Encoding. Unter anderem gab es schon zwei Utils-Methoden, welche das Encoding und Umwandeln deutlich erleichtert haben.

3 Naming Service

Im vorherigen Schuljahr gab es eine Aufgabe, bei der es sich um die Umsetzung und Konfiguration eines Naming Service für Java gehandelt hat. Da einigen Schülern diese Aufgabe nicht mehr zur Verfügung steht, wurde eine Virtuelle Maschine bereitgestellt, welche ein voll funktionstüchtiges LDAP besaß.

Wie die Grafik in der Aufgabenstellung beschreibt wird der **public**-Key vom **Service** im LDAP-Verzeichnis eingetragen und vom **Client** ausgelesen.

```
1 public void storePubKeyinLDAP(){
    LDAPConnector ldapConnector = new LDAPConnector(ldapHost);
3   try {
      ldapConnector.updateAttribute("cn=group.Service1,dc=nodomain,dc=com", "description", UsedUtils.
          toHexString(pubKey.getEncoded()));
5     System.out.println("public key stored in LDAP: "+ UsedUtils.toHexString(pubKey.getEncoded()));
      }catch (Exception e){
7     e.printStackTrace();
    }
9 }
```

Listing 1: LDAP store public-Key in LDAP

4 Aufbau, Verschluesselung und Kommunikation

Die asymmetrische Verschlüsselung passiert hierbei über den Namensdienst, dem Service und dem Client. Wie schon vorher erwähnt stellt der Service seinen **public**-Key in LDAP und der Client holt sich diesen. Er generiert einen **Secret**-Key für die symmetrische Verschlüsselung im späteren Verlauf. Der Client wrappt nun seinen **Secret**-Key mit dem **public**-Key vom Service und übermittelt diesen über **Sockets** an den Service. Der Service decrypted den verschlüsselten Secret-Key mit dem **private**-Key und mit diesem entschlüsselten **Secret**-Key wird eine Nachricht verschlüsselt. Diese Nachricht wird über **Sockets** zurück an den Client übertragen und von diesem entschlüsselt und ausgegeben.

5 Code

Der komplette Source-Code befindet sich in meinem Git-Repository unter dem Link: `https://github.com/mseidl-tgm/5YHITM/tree/master/DEZSYS`

5.1 Service

Die wichtigste Methode *genKeyPair* generiert ein Schlüsselpaar, welches in **public**- und **private**-Key unterteilt werden kann. Hierfür gibt es einen sogenannten *KeyPairGenerator*. Am Schluß werden noch die Attribute, welche die Keys enthalten sollen, gesetzt.

```
1   public void genKeyPair(){
        try {
3           KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA");
            SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");
5           generator.initialize(1024, random);
            this.keyPair = generator.generateKeyPair();
7
        } catch (Exception e) {
9           e.printStackTrace();
        }
11      pubKey = keyPair.getPublic();
        privKey = keyPair.getPrivate();
13
    }
```

Listing 2: RSA KeyPair generation

Der folgende Code-Snippet beschreibt die Funktion *encryptMessage*, in welcher die an den Client zu sendende Nachricht verschlüsselt wird. Es wird ein neuer **Cipher** generiert, welcher die Encryption übernimmt. Es wird der **Secret**-Key angegeben, um die Nachricht richtig zu Verschlüsseln.

```
1   public byte[] encryptMessage(){
        try {
3           Cipher cipher = Cipher.getInstance("AES");
            cipher.init(Cipher.ENCRYPT_MODE, this.secretKey);
5           byte[] msgBytes = toSendMessage.getBytes();
            byte[] tmp = cipher.doFinal(msgBytes);
7           System.out.println("encrypted Message: "+ UsedUtils.toHexString(tmp));
            return tmp;
9       }catch (Exception e){
            e.printStackTrace();
11      }
        return null;
13  }
```

Listing 3: Message encrypting

5.2 Client

Die Methode *encryptSecret* verschlüsselt den generierten Secret-Key mit dem **public**-Key aus dem LDAP-Verzeichnissystem. Hier wird wieder ein **Cipher** verwendet um die den Secret-Key zu "ummanteln".

```
1   public byte[] encryptSecret(){
        byte[] tmp;
3       try {
            Cipher cipher = Cipher.getInstance("RSA");
5           cipher.init(Cipher.WRAP_MODE, this.pubKey);
            tmp = cipher.wrap(secretKey);
7           System.out.println("Secret Key successfully encrypted!" + UsedUtils.toHexString(tmp));
            return tmp;
9       }catch (Exception e){
            e.printStackTrace();
11      }
        return null;
13  }
```

Listing 4: SecretKey encrypting on Client-Side

5.3 UML-Diagramm

Das Diagramm befindet sich auf der nächsten Seite.

5.4 Probleme

Bei den ersten Tests sind einige **Connection-Probleme** bei den Sockets aufgetreten, welche dann zu **Null-Pointer-Exceptions** bei der Codierung geführt haben. Diese haben sich aber dann durch Abstraktion der **readWrite**-Methoden aufgelöst.

5.5 Zeitaufzeichnung

Tabelle 1: Zeitaufzeichnung

Datum, Anzahl in h	Task	Ort
18.10.2016, 3 h	Diagramentwicklung	TGM/daheim
20.10.2016, 4 h	Code-Entwicklung, Debugging	TGM/daheim
21.10.2016, 2 h	Protkoll, Debugging	McDonald/daheim

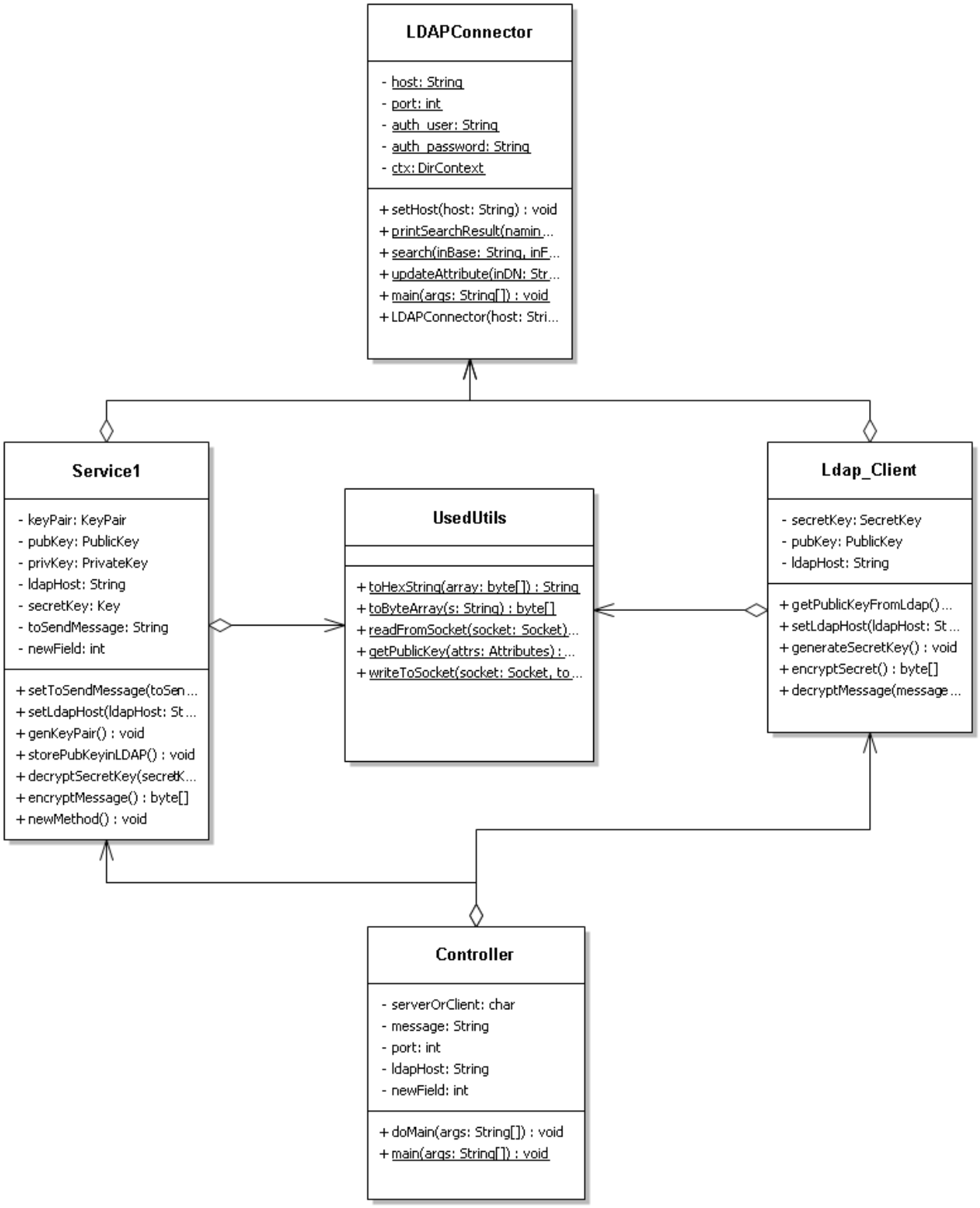

Abbildung 2: UML-Diagramm des Java-Programms

Tabellenverzeichnis

Listings

Abbildungsverzeichnis