JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Java Security

Systemtechnik Labor
5BHITT 2015/16, Gruppe Z

Mathias Ritter

Version 1.1

Note:

Betreuer: Prof. Micheler

Begonnen am 15. Jänner 2016

Beendet am 17. Jänner 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

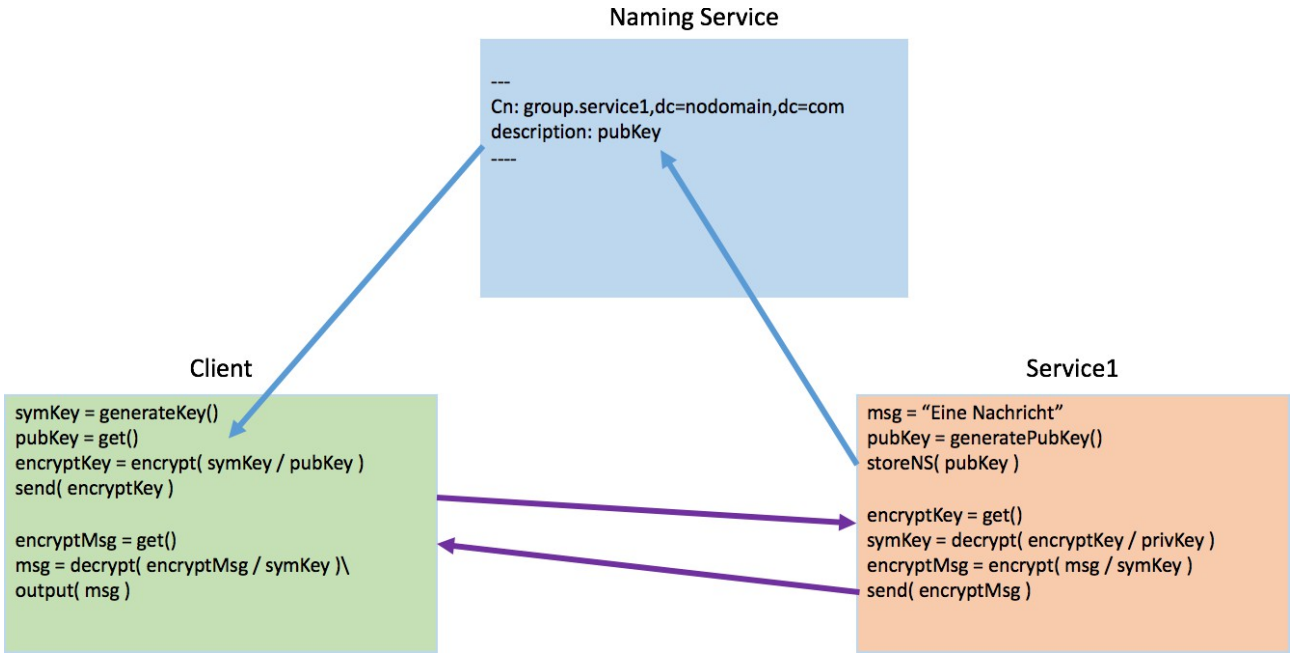
1.3 Aufgabenstellung

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:



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)

2 Ergebnisse

2.1 Kommunikation mit LDAP

Zur Kommunikation mit dem LDAP-Verzeichnisdienst wurde bereits in der letzten Übung eine Klasse `LDAPConnector` implementiert. Diese muss allerdings erweitert werden, um das Ändern eines Attributs zu ermöglichen, da dort der Public-Key abgespeichert wird.

Um das Attribut der zu verwendenden Gruppe zu finden, muss zuerst ein Suche nach der Gruppe ausgeführt werden. Danach kann die Value des gefundenen Attributs abgerufen werden:

```
NamingEnumeration results = this.search("dc=nodomain,dc=com",
      "(&(objectclass=PosixGroup)(cn=" + this.group + "))");
return getAttributeValue(results, "description");
```

Um ein Attribut zu bearbeiten, legt man zuerst eine Liste mit den zu bearbeitenden Attributen an und ruft danach `modifyAttributes` auf:

```
this.context.modifyAttributes(inDN, mods);
```

2.2 Kommunikation zwischen Service und Client

Sockets

Die Kommunikation zwischen Service und Client erfolgt über Sockets. Dafür wurden die Klassen `Networking`, `Networking` und SocketClient implementiert.

Die abstrakte Klasse `Networking` enthält Methoden, welche sowohl vom Client als auch vom Server benötigt werden. Dazu gehört das Senden und Empfangen von Nachrichten und das Beenden einer Verbindung. Die Methode zum Aufbauen zur Verbindung ist abstrakt, da eine unterschiedliche Implementierung bei Server und Client notwendig ist.

Beim Senden einer Nachricht wird zuerst der Typ der Nachricht, danach die Länge und schlussendlich der Inhalt übertragen (siehe „Aufbau einer Nachricht" weiter unten).

```
out.writeChar(message.getType().getValue());
out.writeInt(message.getLength());
out.write(message.getContent());
```

Beim Empfangen werden diese Parameter in der selben Reihenfolge abgerufen. Das Empfangen von Nachrichten erfolgt in einer Schleife in der Run-Methode. Wird eine neue Nachricht empfangen, so wird anschließend eine Methode der Klasse aufgerufen, die die Nachricht weiterverarbeitet (Service oder Client).

Aufbau einer Nachricht

Die Klasse Message stellt den Aufbau einer Nachricht dar. Sie enthält einen `MessageType`, welcher den Typ einer Nachricht angibt. Dieser ist notwendig, damit Service/Client unterscheiden können, ob die empfangene Nachricht verschlüsselt ist, einen Public-Key enthält, dem Beenden der Verbindung dient etc. Folgende Nachrichtentypen werden benötigt und wurden implementiert:

```
ENCRYPTED_SYM_KEY('S'), ENCRYPTED_MESSAGE('E'), STORED_PUB_KEY('P'),
CLIENT_CONNECTED('D'), CLOSE_CONNECTION('C'), SERVICE_READY('R');
```

Weiters enthält sie ein Attribut `content`, welches den Inhalt der Nachricht als Byte-Array enthält.

2.3 Ablauf der Kommunikation

Beim Starten von Service und Client wird eine Verschlüsselung mittels symmetrischer Schlüssel eingerichtet. Die Kommunikation zwischen Client, Service und dem LDAP-Verzeichnisdienst läuft folgendermaßen ab:

1. Starten des Services: Dieser verbindet sich zum LDAP-Server und akzeptiert neue Verbindungen durch einen Client

2. Starten des Clients: Dieser verbindet sich zum LDAP-Server und zum Service.

3. Der Client sendet nach dem Verbindungsaufbau eine Nachricht an den Server (Typ `CLIENT_CONNECTED`)

4. Der Service empfängt diese Nachricht und stellt seinen Public-Key auf der LDAP-Gruppe zur Verfügung. Danach sendet der Service eine Benachrichtigung an den Client (Typ `STORED_PUB_KEY`)

5. Der Client empfängt die Benachrichtigung und ruft den Public-Key vom LDAP-Server ab. Er verschlüsselt damit seinen symmetrischen Schlüssel und sendet diesen an das Service (Typ `ENCRYPTED_SYM_KEY`)

6. Das Service empfängt den verschlüsselten symmetrischen Schlüssel und entschlüsselt diesen mit dem Private-Key. Das Service ist nun ebenfalls bereit, mit dem symmetrischen Schlüssel zu verschlüsseln und schickt eine Antwort an den Client (Typ `SERVICE_READY`)

7. Der Client empfängt die Benachrichtigung. Nun können Service & Client verschlüsselte Nachrichten austauschen.

Möchte der Client oder das Service die Verbindung beenden, wird eine Nachricht mit dem Typ `CLOSE_CONNECTION` gesendet.

2.4 Asymmetrische Verschlüsselung

Wie bereits unter „Ablauf der Kommunikation" beschrieben, wird die asymmetrische Verschlüsselung eingesetzt, um den symmetrischen Schlüssel auszutauschen.

Um einen neuen Private- und Public-Key zu generieren, wird vom Service der `KeyPairGenerator` verwendet:

```
SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");
KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA");
generator.initialize(1024, random);
return generator.generateKeyPair();
```

Das Service verwendet den `LDAPConnector`, um den Public-Key an das Service zu senden:

```
String pubKey =
DatatypeConverter.printHexBinary(this.keyPair.getPublic().getEncoded());
super.getConnector().setGroupDescription(pubKey);
```

Der Client sendet seinen symmetrischen Schlüssel verschlüsselt an das Service. Das Service kann diesen mit dem Private-Key entschlüsseln. Die Entschlüsselung erfolgt mittels der Klasse `Cipher`. Nach der Entschlüsselung wird der symmetrische Schlüssel als `SecretKeySpec` gespeichert:

```
Cipher cipher = Cipher.getInstance("RSA");
cipher.init(Cipher.DECRYPT_MODE, keyPair.getPrivate());
byte[] decrypted = cipher.doFinal(message);
super.setSymKey(new SecretKeySpec(decrypted, 0, decrypted.length, "AES"));
```

Das Service kann nun mit dem symmetrischen Schlüssel verschlüsselte Nachrichten austauschen (siehe „Symmetrische Verschlüsselung“ unten).

2.5 Symmetrische Verschlüsselung

Wie bereits unter „Ablauf der Kommunikation“ beschrieben, wird die symmetrische Verschlüsselung eingesetzt, um Nachrichten zwischen Client und Service verschlüsselt auszutauschen.

Um einen symmetrischen Schlüssel zu generieren, wird der `KeyGenerator` vom Client verwendet:

```
KeyGenerator keygen = KeyGenerator.getInstance("AES");
return keygen.generateKey();
```

Der Client ruft den Public-Key vom LDAP-Server mittels des `LDAPConnectors` ab. Dieser wird als Public-Key-Objekt gespeichert.

```
byte[] key =
      DatatypeConverter.parseHexBinary(
      super.getConnector().getGroupDescription());
X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(key);
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
this.publicKey = keyFactory.generatePublic(pubKeySpec);
```

Der generierte symmetrische Schlüssel wird mit dem Public-Key verschlüsselt und an das Service geschickt. Zum Verschlüsseln wird die Klasse `Cipher` verwendet:

```
Cipher cipher = Cipher.getInstance("RSA");
cipher.init(Cipher.ENCRYPT_MODE, this.publicKey);
byte[] encoded = cipher.doFinal(super.getSymKey().getEncoded());
```

Nun können mit dem symmetrischen Schlüssel Nachrichten vom Service und vom Client gesendet werden. Das Verschlüsseln erfolgt ähnlich wie oben angegeben auch mit dem symmetrischen Schlüssel. Der einzige Unterschied ist, dass statt einer RSA-Instanz eine AES-Instanz der Klasse `Cipher` abgerufen wird. Das Entschlüsseln erfolgt ebenfalls mit dem gleichen Code:

```
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.ENCRYPT_MODE, this.symKey);
byte[] encrypted = cipher.doFinal((text).getBytes());
```

Sowohl bei der Implementierung der symmetrischen als auch bei der Implementierung der asymmetrischen Verschlüsselung hat mir die „Java Cryptography Architecture Reference Guide“ weitergeholfen. [1]

3 Quellen

[1] Java Cryptography Architecture (JCA) Reference Guide [Online]
Verfügbar unter: https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html
[abgerufen am 17.01.2016]