JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Java Verschlüsselung mit LDAP

Systemtechnik Labor
5BHIT 2015/16, GruppeB

Philip Vonderlind

Version 0.1
Begonnen am 14.10.2016
Beendet am 21.10.2016

Note:
Betreuer: Prof. Micheler

Inhaltsverzeichnis

1 Einführung

Diese Übung zeigt die Anwendung von Verschlüsselung in Java.

GitHub Link `https://github.com/pvonderlin-tgm/DEZYSYS_Cryptographie`

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 Verzeichnisdienste
- Administration eines LDAP Dienstes
- Grundlagen der JNDI API für eine JAVA Implementierung
- Grundlagen Verschlüsselung(symetrisch, asymetrisch)
- Einführung in Java Security JCA (Cipher, KeyPairGenerator, KeyFactory)
- Kommunikation in Java (IPC, RPC, Java_RMI, JMS)
- Verwendeung einer viruellen 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:

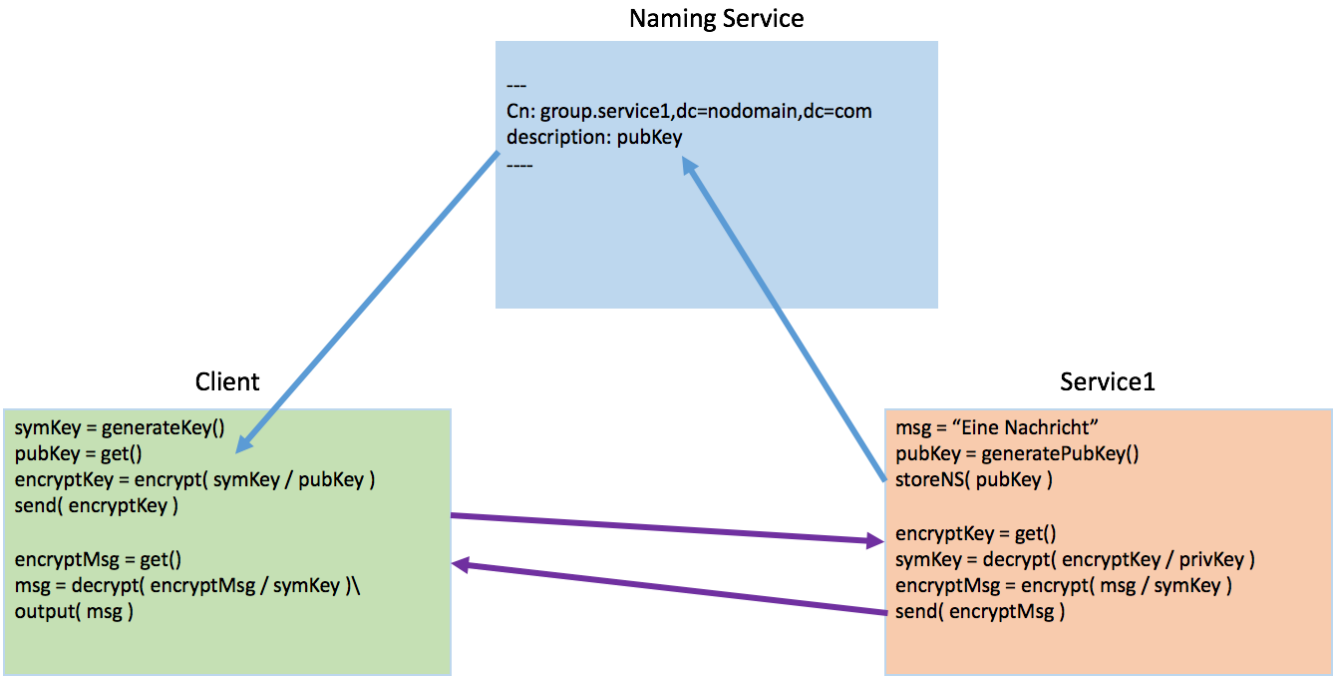

Abbildung 1: Ablauf der Client-Server Struktur

2 Ergebnisse

2.1 Aufsetzen des LDAP Servers

Da wir im vorigen Jahr im Laufe einer Übung einen LDAP fähigen Server aufgesetzt hatten, habe ich diesen nur leicht angepasst und weiterverwendet.
Die für LDAP benötigten Packete sind 'slapd' und 'ldap-utils'. Um die Konfiguration des LDAP anpassen zu können, muss man mit dpkg slapd neu konfiguieren. Der Befehel dafür lautet wie folgt:

```
1   sudo dpkg--reconfigure slapd
```

Listing 1: Reconfigure von LDAP

Von bekommt man nun eine Reihe von Fragen zu Konfiguration gestellt. Bei diesen kann man fast überall die Standartoption auswählen außer bei dem Domainname, dem Passwort für LDAP, dem Organisationsnamen und dem Erlauben des LDAP v2 Protokolls.

```
DOMAIN_NAME_SERVICE = syt.tgm.ac.at --> Diese Adresse haben wir auch letztes Jahr im Lab verwendet, Old
    Habits die hard ...

LDAP_PASSWORD = LDAP
4
 Organsationsname = syt

LDAP v2 = YES
```

Listing 2: Optionen bei dpkg-reconfigure

Durch die auch schon von letzem Jahr vorinstallierte Weboberfläche 'phpldapadmin' kann man einen guten Überblick über sein LDAP erlangen. Hier erstelle ich dann auch später die neuen Common Names für den Private Key. Das Service von phpldapadmin ist über 'IP_DES_SERVES/phpldapdamin' erreichbar. Nach fertigem Konfigurieren und dem hinzufügen eines cn und eines Attribut dessen, sollte die Oberfläche wie folgt aussehen.

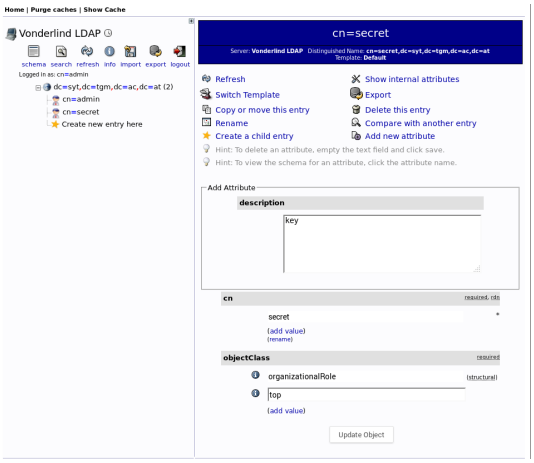

Abbildung 2: LDAP nach CN Erstellung

2.2 Der Server

Wie aus der Graphik ablesbar, gibt der Server einen asymmetrisch generierten Private-Key aus einem Key-Pair an den LDAP Server weiter, wo dieses in einem Attribut eine Common Names gespeichert wird. Danach empfängt er von ein Client der sich verbunden hat einen symmetrisch verschlüsselten Secret-Key, mit welchem er dann eine selbst verschlüsselt Nachricht (auch symmetrisch zurückschickt).

2.2.1 Der Privatekey

Die meisten dieser Methoden wurden schon bereitgestellt und mussten von mir nur noch abgeändert und in eine Klasse eingebaut werden. Um die Reihenfolge des Ablaufes zu visualisieren ist hier der Constructor der Server Klasse:

```
    public Service_1()
    {
        createKeyPair();
        generatePublicKey();
        uploadPrivateKeyToLDAP();
        startServerSocket();
    }
```

Listing 3: Server Constructor

Als erstes wird ein Key-Pair erzeugt, also ein Private- und ein Public-Key zum ver- und endschlüsseln der Daten, die zwischen dem Server und dem Client ausgetauscht werden. Mittels einem KeyPairGenerator und einer SecureRandom Zahl wird mit dem RSA-Algorithmus ein asymmetrisches Schlüsselpaar mit 1024 Bit erzeugt.

```
     private void createKeyPair()
    {
        KeyPairGenerator keyPairGenerator = null;
        SecureRandom random = null;
        try {
            keyPairGenerator = KeyPairGenerator.getInstance("RSA");
            random = SecureRandom.getInstance("SHA1PRNG", "SUN");
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (NoSuchProviderException e) {
            e.printStackTrace();
        }

        keyPairGenerator.initialize(1024, random);
        this.keyPair = keyPairGenerator.generateKeyPair();
    }
```

Listing 4: Erstellung eines asymmetrischen Key-Pairs

Danach wird aus eben diesem Key-Pair der Public-Key für spätere Verwendung herausgefiltert.

```
    his.publicKeyForLDAP = this.keyPair.getPublic().getEncoded();
```

Listing 5: Erstellen des Public Keys

Nun muss dieser Private-Key nur noch in das LDAP System hochgeladen werden. Dazu benutzt man die bereitgestellte Klasse LDAPConnector. Diese beinhaltet eine Methode updateAttribute() mit der man ein bestimmtes Attribut eines Common Names, kurz cn, updaten kann. Durch diesen Mechanismus wird der Private-Key als Attribute auf dem LDAP Server verfügbar für den Client gemacht.

```
    private void uploadPrivateKeyToLDAP()
{
    LDAPConnector ldapConnector = new LDAPConnector();
    try {
        ldapConnector.updateAttribute("cn=secret,"+LDAP_DOMAIN_NAME, "description", this.
            publicKeyForLDAP.toString());
    } catch (NamingException e) {
        e.printStackTrace();
    }
}
```

Listing 6: Upload des Private-Keys auf das LDAP

2.2.2 Der Secret Key

Damit nur der Client die symmetrische Verschlüsselung decrypten kann, gibt dieser dem Server einen asymmetrischen verschlüsselten Secret-Key, mit dem der Server wiederum die Nachrichten symmetrisch verschlüsselt zurücksenden kann. Als erstes muss eine Methode definiert werden die eben diesen Secret-Key aus seinem asymmetrischen 'Wrapper' befreien kann. Dazu verwendet man einfach seinen eigenen Private-Key aus dem Key-Pair. Die Cipher Klasse stellt hier große Funktionalitäten im Bereich der Ver- und Entschlüsselung bereit, die auch verwendet werden. Als Algorithmus steht hier 'AES' bereit.

```
    private void unwrapSymetricalKeyFromClient()
{
    Cipher cipher;

    try {

        cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.UNWRAP_MODE, this.keyPair.getPrivate());
        this.symetricalKeyFromClient = cipher.unwrap(this.wrappedKEyFromClient, "AES", Cipher.
            SECRET_KEY);

    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (NoSuchPaddingException e) {
        e.printStackTrace();
    } catch (InvalidKeyException e) {
        e.printStackTrace();
    }
}
```

Listing 7: Unwrapping des Secret-Keys

Die Nachricht wird nun mit diesem Secret-Key wieder verschlüsselt um später über einen Socket gesendet zu werden. Dazu wird wieder Cipher benutzt.

```
    private byte[] encryptMessageSymetrical()
{
    Cipher cipher;

    try {

        cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, symetricalKeyFromClient);
```

```
            return cipher.doFinal(this.message.getBytes());

        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (NoSuchPaddingException e) {
            e.printStackTrace();
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } catch (BadPaddingException e) {
            e.printStackTrace();
        } catch (IllegalBlockSizeException e) {
            e.printStackTrace();
        }

        return null;
    }
```

Listing 8: Verschlüsseln der Nachricht

2.2.3 Der Socket

Schlussendlich wird die verschlüsselte Nachricht, wie schon angekündigt über einen Socket übertragen. Dies werde ich nicht genauren erläutern, da Sockets relative selbsterklärend sind.

```
    private void startServerSocket()
{
    try {
        ServerSocket serverSocket = new ServerSocket(3000);

        while(true)
        {
            Socket clientSocket = serverSocket.accept();
            System.out.println("A new client has connected to the server"+
            "\n"+"Starting transmission...");

            DataOutputStream out = new DataOutputStream(clientSocket.getOutputStream());
            DataInputStream in = new DataInputStream(clientSocket.getInputStream());

            byte data[];
            if((data= toByteArray(in.readUTF())).length > 0)
            {
                // Decrypt secret key
                this.wrappedKEyFromClient = data;
                unwrapSymetricalKeyFromClient();

                // Encrypt and send message
                encryptMessageSymetrical();
                out.write(this.encryptedMessageSymetrical);
            }

            clientSocket.close();
            out.close();
            in.close();
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Listing 9: Die Kommunikationsschnittstelle

2.3 Der Client

Die Funktionalität des Client ist im Prinzip genau die umgekehrte des Servers. Er verschlüsselt asymmetrisch und entschlüsselt symmetrisch. Daher sind auch die Vorgänge bei der Ver- und Entschlüsselung ähnlich, wenn nicht schon fast ident (bis auf die Konstanten bei Cipher zur Auswahl des Verschlüsselungsmodus dienen).

2.3.1 Der Public Key

Als erstes holt sich der Client den Public-Key für die asymmetrische Verschlüsselung über das LDAP System. Dazu gibt man dem LDAPConnetor einfach nur die DN und den CN und bekommt den Public-Key aus dem Attribut description. getPublicKey ist hierbei eine Methode aus dem vorgegebenen Code.

```
    private String getPublicKeyFromLDAP()
{
    String ldapKey = null;
    LDAPConnector ldapConnector = new LDAPConnector();
    NamingEnumeration listName = null;
    try {
        listName = ldapConnector. search( LDAP_DOMAIN_NAME, LDAP_COMMON_NAME_SEARCH);

        while( listName.hasMore() )
        {
            SearchResult searchResult = (SearchResult) listName.next();
            ldapKey = getPublicKey(searchResult.getAttributes()).toString();
        }
        return ldapKey;
    } catch (NamingException e) {
        e.printStackTrace();
    }
    return null;
}
```

Listing 10: Public Key aus dem LDAP

Nun muss der Public-Key noch 'unwrappt' werden, das heißt es wird aus dem String von LDAP mittels einer Key-Specification, in diesem Fall X509EncodedKeySpec, sowie einer Key-Factory der Public-Key generiert mit dem verschlüsselt werden kann.

```
    private void unwrapPublicKey()
{
    String ldapKey = getPublicKeyFromLDAP();

    byte[] key = toByteArray(ldapKey);
    X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec( key );

    try {

        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        this.pubKey = keyFactory.generatePublic(pubKeySpec);

    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (InvalidKeySpecException e) {
        e.printStackTrace();
    }
}
```

Listing 11: Unwrapping des Public Keys

2.3.2 Der symmetrische Secret-Key

Der Client muss ja, wie vorher schon angesprochen, dem Server einen verschlüsselten Secret-Key geben. Mit diesem kann der Server dann die Nachricht symmetrisch verschlüsseln und zurücksenden.
Um dies zu erreichen, generiert man als erstes mit einem Key-Generator einen Secret-Key mit dem AES Algorithmus und 256 Bit Länge.

```
        private void generateSecretKeySymectrical()
    {
        KeyGenerator keyGenerator= null;

        try {
            keyGenerator =  KeyGenerator.getInstance("AES");
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        keyGenerator.init(256);
        this.secretKey = keyGenerator.generateKey();
    }
```

Listing 12: Generieren eines Secret-Keys

Nun 'wrappt' man diesen Secret-Key mit dem Public-Key aus dem LDAP System, um diesen zu verschlüsseln. Dazu wird wieder Cipher im WRAP_MODE verwendet.

```
        private void wrapSecretSymetricalKeyInPubKey()
    {
        Cipher cipher;

        try {
            cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.WRAP_MODE, this.pubKey);
            this.secretKeyAsymEncrypted = cipher.wrap(this.secretKey);

        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (NoSuchPaddingException e) {
            e.printStackTrace();
        } catch (IllegalBlockSizeException e) {
            e.printStackTrace();
        }
    }
```

Listing 13: Wrappen des Secert-Keys mit dem Public-Key

Hat man nun die symmetrisch verschlüsselt Nachricht vom Server bekommen, kann man diese mit Cipher im DECRYPT_MODE und seinem Secret-Key wieder entschlüsseln.

```
        private byte[] decryptSymetrical(byte[] data)
    {
        Cipher cipher;

        try {
            cipher = Cipher.getInstance("AES");
            cipher.init(Cipher.DECRYPT_MODE, this.secretKey);
            return cipher.doFinal(data);

        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } catch (BadPaddingException e) {
            e.printStackTrace();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
```

```
16          } catch (IllegalBlockSizeException e) {
                e.printStackTrace();
            } catch (NoSuchPaddingException e) {
                e.printStackTrace();
            }
21          return null;
        }
```

Listing 14: Entschlüsseln der symmetrischen Nachricht

2.3.3 Der Socket

Nun muss sich der Client nur über einen Socket zu einem existierenden Server Socket verbinden und von dort die verschlüsselte Nachricht empfangen. Diese wird anschließend auf die Konsole ausgegeben.

```
    private void connectToService()
    {
3       try {

            Socket socket = new Socket("localhost", 3000);
            DataOutputStream out = new DataOutputStream(socket.getOutputStream());
            DataInputStream in = new DataInputStream(socket.getInputStream());
8
            // Send secret key
            out.write(this.secretKeyAsymEncrypted);

            // Reveice message
13          byte[] data;
            if((data = toByteArray(in.readUTF())).length>0)
            {
                System.out.println(new String(this.decryptSymetrical(data)));
            }
18
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Listing 15: Der Socket des Clients

2.4 Änderungen

Ich habe im Endeffekt noch die jeweiligen Domain Names und die Common Names auf die virtuelle Maschine aus dem Beispiel angepasst, da die jetzigen logischerweiße nur bei meinem Server funktioniert hätten.

Tabellenverzeichnis

Listings

Abbildungsverzeichnis

