Prof. Dr. Tobias Urban
Vorlesung: IT-Sicherheit

Technology
Arts Sciences
TH Köln

Praktikum 1:
Ver- und Entschlüsselung in Java

In dieser Praktikumsaufgabe steht die Nutzung des Pakets `javax.crypto`[1] im Mittelpunkt. Dieses Paket bietet eine Reihe von Klassen und Methoden, die es ermöglichen, Daten zu verschlüsseln, zu entschlüsseln und kryptographische Hashwerte zu berechnen.

Insgesamt bietet das Paket `javax.crypto` eine leistungsfähige und flexible Plattform für die Implementierung von kryptographischen Funktionen in Java-Anwendungen. Durch die sorgfältige Planung, Implementierung und Überprüfung können Entwickler sicherstellen, dass ihre Anwendungen die erforderlichen Sicherheitsstandards erfüllen und Daten sicher vor unbefugtem Zugriff schützen.

In dieser Praktikumsaufgabe sollen sie mit Hilfe des Paketes eigenständig Code entwickeln, der zum Ver- und Entschlüsseln von Dateien bzw. zum Austausch von Nachrichten genutzt werden kann. Ebenfalls sollen Sie, mit Hilfe des Pakets, Code schreiben, der zu beliebige Daten einen kryptographischen Hashwert berechnet.

Hinweis: Generell dürfen (und müssen) Sie den bereitgestellten Code erweitern (z.B. neue Methoden oder ggf. sogar Klassen erstellen).

Aufgabe 1: Symmetrische Ver- und Entschlüsselung einer Datei

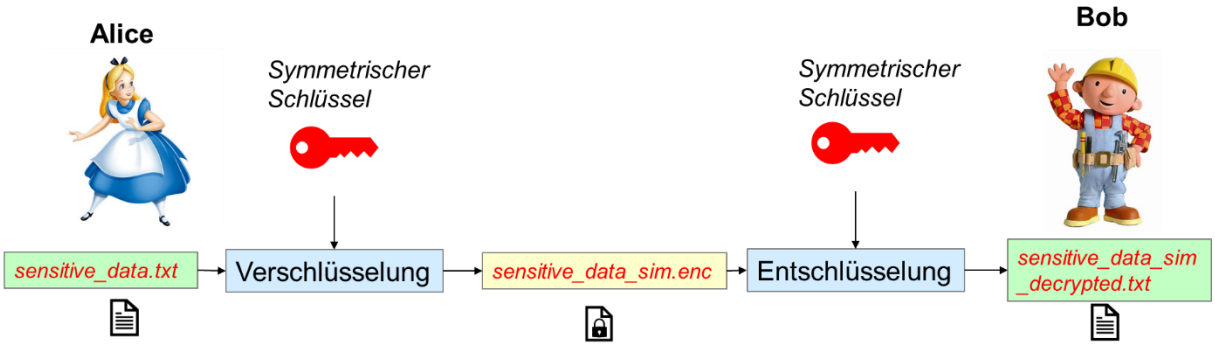

Abbildung 1: Schematische Darstellung der Aufgabe 1.

In dieser Teilaufgabe geht es darum Programmcode zu entwickeln, der die Datei `resources/sensitive_data.txt` mittels symmetrischer Verschlüsselung verschlüsselt,

[1] https://docs.oracle.com/en/java/javase/21/docs/api/java.base/javax/crypto/package-summary.html

in einer neuen Datei den Chiphertext ablegt und anschließend diese Datei wieder entschlüsselt und den entschlüsselten Plaintext in einer Datei ablegt (siehe Abbildung 1).

Aufgabe 1.1: Implementierung in Java

In der Aufgabe soll der *Advanced Encryption Standard* (AES) und eine vom Bundesamt für Sicherheit in der Informationstechnik (BSI) als sicher eingestufte Schlüssellänge verwendet werden. Ebenfalls sollt kein vom BSI als unsicher eingestufter Betriebsmodus (z.B. ECB) genutzt werden. Allerdings muss der sichere *Austausch* von Geheimnissen (z.B. des symmetrischen Schlüssels) nicht gesondert betrachtet werden.

Als Vorlage für die Umsetzung der Aufgabe steht ein Java-Projekt zur Verfügung. Die benötigten Klassen liegen in dem Packet `edu.thkoeln.itsec.verschluesselung.symmetrisch`. Für diesen Teil der Aufgabe sind vier der Verfügbaren Dateien von Bedeutung:

- Die Datei `PerformSymmetricEncryption.java` enthält die main-Method des Projekts. Bis auf die Zeile, in welcher der Betriebsmodus definiert wird `(private static final String AES_CHIPER_MODE = "CHANGE ME";)` muss diese Datei nicht angepasst werden. Wenn die Datei ohne Probleme kompiliert und ohne Fehler (Exceptiones) ausgeführt werden kann, funktioniert die Verschlüsselung *wahrscheinlich*.
- Die Datei `AbstractSymmetricEncryptor.java` enthält eine abstrakte Klasse und muss für die Aufgabe nicht angepasst werden.
- Die tatsächliche AES-basierte Implementierung muss in der Datei: `SymmetricEncryptor.java` umgesetzt werden. Hier müssen zum einen die Methoden `public void encryptFile(File inputFile, File outputFile) throws Exception` und `public void decryptFile(File inputFile, File outputFile) throws Exception` implementiert werden und zum anderen die für den AES-Algorithmus wichtigen Parameter generiert werden (z.B. der Symmetrische Schlüssel). Obwohl in der Praktikumsaufgabe kein Austausch der Schlüssel und anderen Materialien vorgesehen ist, müssen sie diese zwischen speichern, um zumindest einen Austausch zu simulieren bzw. zu ermöglichen.
- Letztlich wird eine Datei (`resources\sensitive_data.txt`) zur Verfügung gestellt, welche für die Verschlüsselung genutzt werden soll.

Aufgabe 1.2: Herausforderungen bei der Speicherung von Schlüsseln

In der Aufgabe ist der Austausch der Schlüssel nicht nötig. Diese sollen aber auf der Festplatte gespeichert werden. Welche Probleme sehen Sie in diesem Vorgehen?

Aufgabe 2: Asymmetrische Verschlüsselung der Kommunikation zwischen zwei Parteien

Ähnlich zu Teilaufgabe 1 des Arbeitsblattes soll in dieser Teilaufgabe eine Textnachricht (String) mittels asymmetrischer Verschlüsselung ver- und entschlüsselt werden.

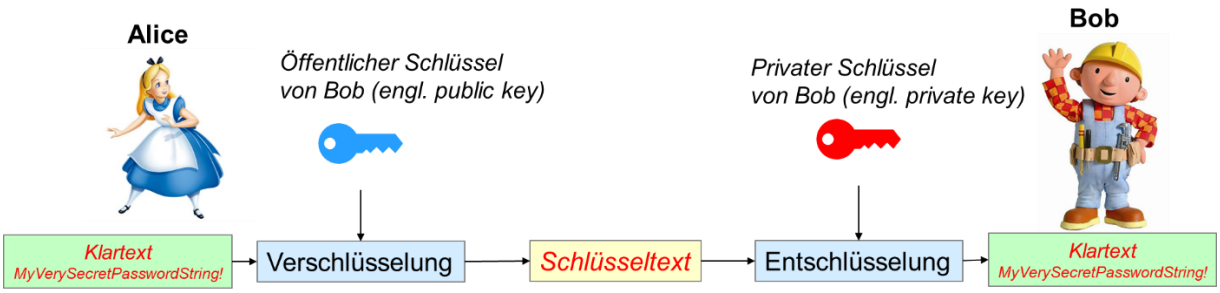

Abbildung 2: Schematische Darstellung der Aufgabe 2.

Aufgabe 2.1: Implementierung in Java

Für die Verschlüsselung soll das RSA (*Rivest–Shamir–Adleman*) Verfahren genutzt werden. Dabei soll eine vom Bundesamt für Sicherheit in der Informationstechnik (BSI) als sicher eingestufte Schlüssellänge verwendet werden. Als Vorlage für die Umsetzung der Aufgabe steht das gleiche Java-Projekt wie in Aufgabe 1 zur Verfügung. Die benötigen Kalssen liegen in dem Packet `edu.thkoeln.itsec.verschluesselung.asymmetrisch`. Für diesen Teil der Aufgabe sind vier der Verfügbaren Dateien von Bedeutung:

- In der Datei `SimpleRsaCommunication.java` befindet sich die main-Methode. Diese muss so angepasst werden, dass die beiden Kommunikationspartner (Alice und Bob) über das passende Schlüsselmaterial zum Verschlüsseln einer Nachricht verfügen. Auch hier gilt wieder: Sollte die `main` Methode ohne Fehler (Exceptions) durchlaufen, hat die Ver- und Entschlüsselung *wahrscheinlich* funktioniert.
- In der Datei `AbstractRsaCommunicationPartner.java` muss der Konstruktor so angepasst werden, dass für einen Kommunikationsteilnehmer das passende schlüsselmaterial generiert wird.
- In der Datei `RsaCommunicationPartner.java` müssen die Funktionen zum verschlüsseln (`public byte[] encryptMessage(String message) throws`

`Exception`) und entschlüsseln (`public String decryptMessage(byte[] ciphertext) throws Exception`) mittels RSA Verfahren umgesetzt werden sowie einfach GET- und SET-Methoden für das Schlüsselmaterial implementiert werden.

Aufgabe 2.2: Lange Nachrichten

Was passiert, wenn die zu übertragende Nachricht sehr lang wird? Zum Beispiel: `String plainText = "MyVerySecretPasswordString!".repeat(999);` Was könnte man tun, um diese Herausforderung zu lösen?

Aufgabe 3: Kryptographischen Hashwerte und Digitale Signaturen

In der letzten Teilaufgabe soll zum einen kryptographischer Hashwert und zum anderen eine digitale Signatur des Logos der TH Köln berechnet werden. Zur Umsetzung soll, die für Aufgabe 2 entwickelte Klasse erweitert werden.

Aufgabe 3.1: Berechnung eines Hashwertes in Java

Für die Berechnung des kryptographischen Hashwertes, soll ein Verfahren genutzt werden, dass von dem BSI als sicher eingestuft wird. Als Vorlage für die Umsetzung dient die in Teilaufgabe 2 entwickelte Klasse `RsaCommunicationPartner` sowie die entsprechende abstrakte Oberklasse `AbstractRsaCommunicationPartner`. Die benötigte Klasse liegt in dem Packet `edu.thkoeln.itsec.verschluesselung.hashing`.

- Dabei sind in den Klassen besonders die Methoden `public byte[] comupteCryptographicHash(byte[] bytes)`, sowie die entsprechenden Attribute von Interesse. Hier muss der Code zur Erstellung der Signatur ergänzt werden.
- Die Datei `HashesAndSignatures.java` enthält die main-Methode und muss für diese Teilaufgabe nicht angepasst werden.
- Es wird eine Datei, zu der ein Hashwert berechnet werden soll, bereitgestellt (`TH-resources/Koeln-logo.png`).

Aufgabe 3.2: Signieren von Daten und prüfen von digitalen Signaturen

In dieser Teilaufgabe soll die Klasse `RsaCommunicationPartner` so erweitert werden, dass diese zur Erstellung und Verifikation von digitalen Signaturen genutzt werden kann.

Dazu muss die Klasse `RsaCommunicationPartner` und die dazugehörige Oberklasse angepasst werden.

- Dabei stehen die Methoden `public byte[] signData(byte[] bytes)` zum Signieren beliebiger Daten und `public boolean verifySignature(byte[] signatureBytes, byte[] messageBytes)` zur Verifikation von digitalen Signaturen im Fokus. In diesen Methoden muss der Code zur Erstellung und Verifiaktion der Daten eingefügt werden.
- Die Datei `HashesAndSignatures.java` enthält die main-Methode. Wie in der vorherigen Aufgabe muss hier der „Austausch“ der Schlüssel zwischen den Objekten `alice` und `bob` vorgenommen werden. Ansonsten müssen keine Änderungen vorgenommen werden.
- Es wird eine Datei, zu die Signatur erstellt werden soll, bereitgestellt (`TH-resources/Koeln-logo.png`).