1. Erzeugen eines Schlüsselpaares

Schreiben Sie ein JAVA-Programm **`RSAKeyCreation`**, welches ein RSA-Schlüsselpaar erzeugt (Schlüssellänge: 2048 Bit) und beide Schlüssel jeweils in einer Datei speichert. Der Name des Inhabers soll als Argument in der Kommandozeile übergeben werden.

Der öffentliche Schlüssel soll einer einer Datei <Inhabername>.pub gespeichert werden, deren Struktur wie folgt aussieht:

1. Länge des Inhaber-Namens (integer)
2. Inhaber-Name (Bytefolge)
3. Länge des öffentlichen Schlüssels (integer)
4. Öffentlicher Schlüssel (Bytefolge) [X.509-Format]

Der private Schlüssel soll einer einer Datei <Inhabername>.prv gespeichert werden, deren Struktur wie folgt aussieht:

1. Länge des Inhaber-Namens (integer)
2. Inhaber-Name (Bytefolge)
3. Länge des privaten Schlüssels (integer)
4. Privater Schlüssel (Bytefolge) [PKCS8-Format]

Beispiel: `java RSAKeyCreation KMueller`
erzeugt die Ausgabedateien `KMueller.pub` *und* `KMueller.prv`

2. Erzeugen, Signieren und Verschlüsseln eines geheimen Sitzungsschlüssels und Verschlüsseln einer Dokumentendatei (Sender)

Schreiben Sie ein JAVA-Programm **SSF** („SendSecureFile“) mit folgender Funktionalität:

a) Einlesen eines privaten RSA-Schlüssels (.prv) aus einer Datei gemäß Aufgabenteil 1.
b) Einlesen eines öffentlichen RSA-Schlüssels (.pub) aus einer Datei gemäß Aufgabenteil 1.
c) Erzeugen eines geheimen Schlüssels für den AES-Algorithmus mit der Schlüssellänge 128 Bit
d) Erzeugung einer Signatur für den geheimen Schlüssel aus c) mit dem privaten RSA-Schlüssel (Algorithmus: „SHA1withRSA“)
e) Verschlüsselung des geheimen Schlüssel aus c) mit dem öffentlichen RSA-Schlüssel (Algorithmus: „RSA“)
f) Einlesen einer Dokumentendatei und Verschlüsseln der Datei mit dem symmetrischen AES-Algorithmus (geheimer Schlüssel aus c).
g) Erzeugung einer Ausgabedatei mit folgender Struktur:
    1. Länge des verschlüsselten geheimen Schlüssels (integer)
    2. Verschlüsselter geheimer Schlüssel (Bytefolge)
    3. Länge der Signatur des geheimen Schlüssels (integer)
    4. Signatur des geheimen Schlüssels (Bytefolge)
    5. Verschlüsselte Dateidaten (Ergebnis von f) (Bytefolge)

Die Dateinamen sollen als Argument in der Kommandozeile übergeben werden.
Als privater Schlüssel ist derjenige des Senders zu verwenden, als öffentlicher Schlüssel derjenige des Empfängers.

Beispiel (K. Müller sendet an F. Meier):
`java SSF KMueller.prv FMeier.pub Brief.pdf Brief.ssf`

3. Entschlüsseln und Verifizieren eines geheimen Sitzungsschlüssels und Entschlüsseln eines Dokuments (Empfänger)

Schreiben Sie ein JAVA-Programm **RSF** („ReceiveSecureFile“) mit folgender Funktionalität:

a) Einlesen eines öffentlichen RSA-Schlüssels aus einer Datei gemäß Aufgabenteil 1.
b) Einlesen eines privaten RSA-Schlüssels aus einer Datei gemäß Aufgabenteil 1.
c) Einlesen einer .ssf-Datei gemäß Aufgabenteil 2, Entschlüsselung des geheimen Schlüssels mit dem privaten RSA-Schlüssel, Entschlüsselung der Dateidaten mit dem geheimen Schlüssel (AES) sowie Erzeugung einer Klartext-Ausgabedatei.
d) Überprüfung der Signatur für den geheimen Schlüssel aus c) mit dem öffentlichen RSA-Schlüssel (Algorithmus: „SHA1withRSA“)

Die Dateinamen sollen als Argument in der Kommandozeile übergeben werden.
Als öffentlicher Schlüssel ist der Schlüssel des Senders zu verwenden, als privater Schlüssel derjenige des Empfängers.

Beispiel (F. Meier empfängt von K. Müller):
```
java RSF FMeier.prv KMueller.pub Brief.ssf Brief.pdf
```

4. Testverfahren

Testen Sie Ihre Programme aus Aufgabenteil 1.- 3. auf folgende Weise:

a) Erzeugen Sie sich ein eigenes RSA-Schlüsselpaar (mit `RSAKeyCreation).`
b) Ver- und Entschlüsseln Sie die Dokumentendatei `ITSAufgabe3.pdf` mit Ihren Programmen SSF und RSF (inkl. Signierung und Verifikation).
c) Benutzen Sie Ihr eigenes SSF-Programm zur Verschlüsselung (mit `MHuebner.pub` als öff. RSA-Schlüssel) und Signierung sowie das Programm `RSFTest` (mit `MHuebner.prv` als privatem RSA-Schlüssel) aus dem Pub-Verzeichnis zur Entschlüsselung und Verifikation.

Verwenden Sie die kryptographischen Default-Parameter des Providers SUN (insbesondere zum Test mit `RSFTest`), falls in der Aufgabenstellung nichts anderes angegeben ist!

Tipps und Infos:

- Orientieren Sie sich an den Beispielprogrammen der JCA_JCE-Einfuehrung!
- Benutzen Sie die JAVA-Packages `java.security`, `java.security.spec`, `javax.crypto`, `javax.crypto.spec` und `java.io` !
- Verwenden Sie die Klassen `DataInputstream` und `DataOutputStream` !
- Verarbeiten Sie die verschlüsselten (Massen-)Daten beim Dateilesen und -schreiben über einen Puffer vom Type `byte[]` (wie in der 2. Praktikumsaufgabe)
- In allen Programmen sind geeignete Fehlerbehandlungsmaßnahmen vorzusehen.

Viel Spaß!