WPP	Praktikum IT-Sicherheit	UGUS
SS 13	Aufgabe 2 – Symmetrische Verschlüsselungstechniken	07.05.13

1. Klassische Kryptographie

a. Vigenère-Chiffre
Nehmen wir an, es gäbe eine Sprache, die nur aus den Buchstaben **a** und **b** besteht. Nehmen wir weiter an, dass der Buchstabe **a** mit Häufigkeit von **10%** in der Sprache auftritt.

Gegeben sei der Chiffretext **ABABABBBAB**.

- Berechnen Sie die Länge des Schlüsselwortes mit dem in der Vorlesung besprochenem Verfahren (n = ?).
- Berechnen Sie das Schlüsselwort und entschlüsseln Sie den Chiffretext.

b. Hill-Chiffre

- Gegeben sei der Schlüssel **k =** $\begin{pmatrix} b & c \\ d & e \end{pmatrix}$. Finden Sie zwei Nachrichten, die dieselbe Chiffre erzeugen.
- Gegeben sei der Chiffretext **GEZXDS**, der mit einer **2x2 Matrix** verschlüsselt wurde. Der Klartext ist ***solved***. Berechnen Sie den Schlüssel.

➢ *Hinweis:* (a,b) * $\begin{pmatrix} c & d \\ e & f \end{pmatrix}$ = ((a*c + b*e), (a*d + b*f))

2. Pseudozufallszahlengenerierung

Implementieren Sie in JAVA einen Pseudozufallszahlengenerator nach der **Linearen Kongruenzmethode**!

Stellen Sie eine Klasse `LCG` mit einer Methode `nextValue()` zur Verfügung. Diese Methode soll nach der linearen Kongruenzmethode einen „Zufallswert“ zwischen 0 und 1 liefern. Der Startwert des Pseudozufallszahlengenerators (X_0) soll dem Konstruktor der `LCG` – Klasse als Parameter übergeben werden.

➢ *Verwenden Sie eine Parameterkombination für a, b und N aus der Datei „LinearerKongruenzgenerator-Infos.pdf“!*
➢ *Achten Sie grundsätzlich auf die Verwendung eines geeigneten Datentyps (z.B. „long“), um mögliche Überläufe zu vermeiden!*

3. Stromchiffre

a. Schreiben Sie unter Verwendung der `LCG` – Klasse aus Teil 1 ein JAVA-Programm **`HC1`** („HAW-Chiffre 1“), welches als Eingabeparameter von der Standardeingabe einen numerischen Schlüssel (Startwert) sowie den Pfad für eine zu verschlüsselnde / entschlüsselnde Datei erhält.
Ihr Programm soll jedes Byte der Datei mit einem – ausgehend vom übergebenen Startwert – „zufällig“ erzeugten Schlüsselbyte mittels XOR verknüpfen und das Ergebnis in eine neue Chiffredatei ausgeben.

b. Testen Sie Ihre Stromchiffre **`HC1`**, indem Sie eine Klartextdatei verschlüsseln und die erzeugte Chiffredatei anschließend durch einen nochmaligen Aufruf von HC1 wieder entschlüsseln. Verifizieren Sie (z.B. mittels „diff“), dass beide Dateien identische Inhalte besitzen.

- *Arbeiten Sie mit Input/Outputstreams und vermeiden Sie die Verwendung von „Buffered Reader" oder „Buffered Writer" – Klassen!*

4. TripleDES als Blockchiffre im CFB-Modus

a. Implementieren Sie in JAVA ein Programm **`HC2`** mit der Aufgabe, alle Bytes einer beliebigen Datei mit Hilfe des TripleDES-Verfahrens zu verschlüsseln oder zu entschlüsseln. Ihr Programm soll zur Verschlüsselung mit drei verschiedenen DES-Schlüsseln arbeiten und in der Reihenfolge EDE (encrypt-decrypt-encrypt) das DES-Verfahren dreimal hintereinander durchlaufen. Implementieren Sie als Blockchiffre-Betriebsart[1] den CFB-Modus (*Folie zum CFB-Modus liegt im pub*).

Kommandozeilen-Parameter:

1. Dateiname einer zu verschlüsselnden/entschlüsselnden Datei
2. Dateiname einer Datei mit folgendem Inhalt:
   Byte 1-24: 24 Schlüsselbytes (3 DES-Schlüssel à 8 Byte, wobei von jedem Byte jeweils 7 Bit verwendet werden)
   Byte 25-32: 8 Bytes für den Initialisierungsvektor zum Betrieb im CFB - Modus
3. Dateiname der Ausgabedatei
4. Status-String zur Angabe der gewünschten Operation:
   `encrypt` – Verschlüsselung der Datei
   `decrypt` – Entschlüsselung der Datei

b. Testen Sie Ihre Implementierung, indem Sie die Datei „3DESTest.enc" entschlüsseln (Tipp: Ergebnis ist eine PDF-Datei). Schlüssel und Initialisierungsvektor finden Sie in der Datei „3DESTest.key".

Tipps:

- *Benutzen Sie als DES-Implementierung die mitgelieferte Datei `DES.java` !*

- *Überlegen Sie sich, ob bei Anwendung des CFB-Modus die Entschlüsselungsfunktion des TripleDES-Algorithmus überhaupt benötigt wird.*

- *Beispielcode zum blockweisen Kopieren einer Datei:*

```
FileInputStream in;
FileOutputStream out;
byte[] buffer = new byte[8];
int len;
while ((len = in.read(buffer)) > 0) {
     out.write(buffer, 0, len);
}
```

Viel Spaß!

[1] Die Betriebsart bezieht sich hier auf das gesamte TripleDES-Verfahren, nicht auf die einzelnen DES-Aufrufe!