Applikationssicherheit Laboraufgabe 1

# Ressourcen

Die Files wurden in ISO-8859-1 kodiert, damit die Umlaute richtig dargestellt und ausgelesen werden können.

**original.txt**

Die originale Textnachricht von Bob, plaintext

**Patterns.txt**

Patterns, die benutzt werden, um eine Nachricht zu generieren, welche der Originalnachricht ähnlich sehen.

Die Form dazu ist die folgende:

* Je Pattern existiert eine Linie im File
  + Gesamthaft also 32 Linien => Bits in einem Integer
* Jede Linie ist durch eine Trennlinie, in diesem Fall „|“, unterteilt

Diese Patterns werden in der Klasse TextGenerator in ein 2D Array in der folgenden Form abgefüllt:

* Erster Index: Linie im File
* Zweiter Index: Position relativ zur Trennlinie => Links = 0, Rechts = 1

# Klassen

**DESHash (hash)**

Allgemein: Durch diese Klasse kann ein Hashwert mittels DES generiert werden.

Detail:

1. Generiert den Initialisierungsvektor g0
2. Holt die Bytes vom übergebenen String
3. Speichert die Länge der aktuellen Nachricht in einer Variable zwischen
4. Überprüft, ob ein Padding nötig ist und fügt sie hinzu
   1. Länge wird nach der folgenden Formel berechnet: ((x/8)+1)\*8
   2. Padding in Form 1000….
5. Speichert die Länge der aktuellen Nachricht in dessen Byte Array
6. Erzeugt ein Objekt der Klasse PaddedBufferedBlockCipher
   1. Als Parameter wird ein Objekt der Klasse DESEngine übergeben
7. Iteration (von i bis Ende Nachricht; Schritt in BlockSize des Ciphers:
   1. Initialisiert den Cipher mit dem Schlüssel
   2. Erstellt ein Byte Array
      1. Als Grösse wird der OutputSize des Ciphers bzw. des DES genommen
   3. Cipher bearbeitet einen Block
   4. Cipher gibt den Output durch doFinal aus und speichert es im Block
   5. Erstellen des neuen Schlüssels
      1. Durch Formel
         1. Block muss in sich xor-en, da DES einen Output der Länge 2\*64 Bit liefert
8. Führt den letzten Schritt im DES aus. Berechnet also H in der Formel.
9. Wrapt das Ergebnis in ein ByteBuffer und form ein Integer daraus
10. Gibt diesen Integer als Resultat zurück

**Start (main)**

Allgemein: Diese Klasse ist ein Startpunkt für die Applikation.

Ansatz: Festes originales Dokument. Generiertes gefälschtes Dokument (mit n^32 Permutationen).

Detail:

1. Erzeugt Instanzen der Klassen TextGenerator und DESHash.
2. Ladet das File Patterns.txt
3. Erzeugt den Hashwert des originalen Textes
4. Schleife:
   1. Erzeugt den Hashwert der Kopie
   2. Prüft, ob die Hashwerte übereinstimmen
   3. Bricht ab, falls ein Hashwert gefunden wurde
5. Gibt den Informationen zur Kollision auf der Konsole aus

**StartPermutation**

Allgemein: Diese Klasse ist ein Startpunkt für die Applikation.

Ansatz: Beide Dokumente, sowohl gefälscht wie original, werden generiert (n^32 Permutationen je).

1. Festlegen, wie viele Kollisionen der Benutzer finden will
2. Instanzieren von je einem TextGenerator und einer HashMap für die Files
3. Instanzieren einer HashMap für di gefundenen Keys
4. Instanzieren der Klasse DESHash
5. Schleife; so lange bis der Zähler die gewünschte Anzahl Kollisionen erreicht hat:
   1. Generieren von je 2048 neuen Texten pro File und speichern dieser Texte inkl. deren Hashes in den zugeordneten HashMaps.
   2. Iterieren über die Keys der HashMap für die gefälschten Files
      1. Falls der Hash ebenfalls in der Map für die originales Files vorkommt
         1. Invariante: Falls der Hash bereits bekannt ist, ist mindestens ein originaler Text bekannt.
         2. Falls dieser Hash bereits bekannt ist
            1. Es soll überprüft werden, ob der Text zum Hash bereits abgespeichert wurde. Falls nicht, soll der abgespeichert werden.
            2. Falls der abgespeicherte Text einer der gefälschten Texte war, soll der Counter für gefundene Kollisionen hochgezählt werden.
         3. Falls dieser Hash nicht bekannt ist
            1. Es soll eine neue Liste kreiert und die Texte (sowohl gefälscht wie original) sollen mit dem dazu gehörenden Hash in der Map für gefundene Hashes abgespeichert werden.
6. Ausgeben der Kollisionen

**TextGenerator**

Allgemein: Diese Klasse generiert Kopien des Originals mit Permutationen.

Methodendetails:

loadTextFile: Diese Methode soll dazu dienen, die Patterns zu laden

1. Liest das File ein
   1. Dabei wird jede Linie einzeln eingelesen
   2. Jede Linie wird am Trennpunkt gesplitet und in einem 2D Array gespeichert
      1. 2D Array erster Index: Nummer der Linie
      2. 2D Array zweiter Index: Nummer des Patterns

getNextText: Diese Methode soll dazu dienen den nächsten Text aus dem 2D Array zu holen.

1. Überprüft, ob randomisiert werden soll
   1. Falls nicht, dann soll der nächste Text gemäss der Laufnummer ausgegeben werden
   2. Falls ja
      1. Den nächsten Integer aus dem Randomisierer holen.
      2. Überprüfen, ob diese bereits genommen wurde
         1. Falls ja, den nächsten Text ausgeben (sich selber aufrufen)
         2. Falls nicht, die Nummer in die Liste hinzufügen und den aktuellen Text ausgeben

getText: Diese methode soll den Text anhand eines Integers generieren

1. Den Integer in dessen binärer Form in einen String konvertieren.
2. StringBuilder instanzieren
3. Durch diesen String iterieren:
   1. Der Index im String bestimmt die Linie bzw das Pattern
   2. Der Character (0,1) bestimmt, ob in einem Pattern die erste oder zweite Position genommen wird
   3. Append im StringBuilder
4. Rückgabe des Strings vom StringBuilder