University of Applied Sciences and Arts Northwestern Switzerland
School of Engineering

Warum der Wert von n in 2^n wichtig ist

4. November 2013

Fabio Oesch & Jan Fässler

Inhaltsverzeichnis

Institute of computer science,
www.fhnw.ch

1 Aufgabenstellung

Konstruieren Sie zwei Briefe an Alice, einen (orginal) für Bob und einen (gefälscht) für Alice, die aber den gleichen Hashwert haben. Da die Fälschung etwas für Sie einbringen soll, ersetzen Sie im Brief an Alice die Kontonummer 222-1101.461.12 durch Ihre eigene: 202-1201.262.10. Sie haben freilich bei der gleichen Bank ein Konto mit dieser Nummer eröffnet.

2 Software Aufbau

Das Projekt wurde in 2 Klassen unterteilt. Die Klasse HashingMachine ist für das Hashing zuständig. Die Klasse LabOne ist für das einlesen der Daten und das Durchprobieren der Hashingmethoden zuständig.

2.1 HashingMachine

Diese Klasse hasht den Vorgegebenen Text mit der Hashfunktion die wir von der Aufgabe erhalten haben.

2.1.1 preprocess

Die Methode preprocess(byte[] input) strukturiert den Input wie er nachher benötigt wird.

Listing 1: Structure the input

```
1 for (int i = 0; i < input.length; i++) out[i] = input[i];
2 if (r > 0)                              out[input.length] = -128;
3 for (int i = 1; i < r; i++)             out[input.length + i] = 0;
4 for (int i = 0; i < 8; i++)             out[out.length - 8 + i] = length
      [i];
```

Die erste Linie kopiert den input in den Ouput. Linien 2 und 3 sind für das Padding verantwortlich. Die Linie 4 kopiert die Länge in den Output.

2.1.2 create

Listing 2: Hashing

```
1
2 byte[] hash = iv.clone();
3
4 for (int i = 0; i < input.length; i += 8) {
```

```
5   try {
6     desOut = new byte[16];
7     cipher.init(true, new KeyParameter(hash));
8     cipher.processBytes(input, i, 8, desOut, 0);
9     cipher.doFinal(desOut, 0);
10    for (int j = 0; j < hash.length; j++)
11      tempState[j] = (byte) ((desOut[j] ^ desOut[j + 8]) ^ hash[j]);
12  } catch (CryptoException ce) { System.err.println(ce); }
13
14  // swap
15  byte[] tmp = tempState;
16  tempState = hash;
17  hash = tmp;
18 }
19
20 ByteBuffer buffer = ByteBuffer.wrap(hash);
21 buffer.order(ByteOrder.LITTLE_ENDIAN);
22 long result = buffer.getLong();
23 return (int) (result >>> 32) ^ Integer.reverse((int) result);
```

Auf der Zeile .. wird der cipher mit dem Initialvektor initialisiert.
Xor Magic von altem und neuem Zeugs
Da dies ein Blockcipher ist wird nun ein neuer hash benutzt mit der der Text verschlüsselt wird.

2.2 LabOne

Main Klasse in der, die Hashfunktion geknackt wird.

2.2.1 main

In dieser Klasse können wir Einstellungen für unsere Hashfunktion vornehmen.

2.2.2 resolveCombination

Je nachdem wie der Integer ist von combination wird ein String generiert, mit dem Template, für die er die verschiedenen Optionen von options.ini einsetzt.

2.2.3 createVariation

Macht das selbe wie resolveCombination jedoch wird stattdessen der Hashwert als Integer zurückgegeben.

2.2.4 generateHashes

Listing 3: Hashing Knacken

```
1 for (int i = 0; i < count; i++) {
2   if (useRandom) {
3     do { originalCombination = rand.nextInt(); } while (hashesOriginal.
          containsValue(originalCombination));
4     do { fakeCombination     = rand.nextInt(); } while (hashesFake.
          containsValue(fakeCombination));
5   } else {
6     originalCombination++;
7     fakeCombination++;
8   }
9   hash = createVariation(originalCombination, templateOriginal);
10  hashesOriginal.put(hash, originalCombination);
11  hash = createVariation(fakeCombination, templateFake);
12  hashesFake.put(hash, fakeCombination);
13 }
```

Methode welche überprüft, ob man den gleichen Hashwert erhalten hat. Man überprüft, ob man die generierten fakeCombination's eine Kollision mit den dem erstellten orginalCombination hat.