University of Applied Sciences and Arts Northwestern Switzerland
School of Engineering

APSI Lab 1

4. November 2013

put abstract here

Inhaltsverzeichnis

1 Aufgabenstellung

Ihre Aufgabe besteht darin, sogenannte Kollisionen im Hash-Verfahren zu suchen, d.h. nderungen im Originaltext, die den gleichen Hashwert liefern: $h(m_{orig}) = h(m_{fake})$. Wie Sie vielleicht bereits bemerkt haben, handelt es sich um eine praktische Anwendung des bekannten Geburtstagsparadoxons, das Sie in der Mathematik bzw. in der Kryptologie kennengelernt haben.

2 Softwareaufbau

Die Aufgabe wurde mit zwei Klassen implementiert. Die Klasse S̈implifiedHashb̈einhaltet die beschriebene Hashfunktion und in der Klasse C̈ollisionGeneratorẅerden die verschiedenen Kombinationen der Mails generiert und nach einer Hashkollision berprft.

2.1 Hashfunktion

Die Hashfunktion wurde nach der Spezifikation der Aufgabenstellung implementiert. Jedoch hat der DES Cipher eine Output-Lnge von 128 Bit, wie man die 128 Bit verkrzen soll auf 64 Bit ist nicht spezifiziert. Deshalb wird der DES-Output in zwei 64-Bit-Blcke aufgeteilt und mit XOR auf einen 64-Bit-Block eingestampft.

2.2 Kollisionsgenerator

2.2.1 Variationserzeugung

Lauf der Aufgabenstellung haben wir 2^{32} verschiedene Kombinationsmglichkeiten pro Mail. Diese Kombinationen haben wir in einem Integer codiert, dabei reprsentiert ein Bit einen Platzhalter. Zum Beispiel: Das zweite Bit steht auf 0, dann wird das Wort ”vom Herzenëingesetzt. So knnen wir jede Variation eindeutig bestimmen.

2.3 Kollisionsdetektion

2.3.1 Strategien

Wir knnen zwischen zwei Strateien unterscheiden. Entweder, wir generieren alle Original und Fake Variationen linear (beginnend bei 0), oder wir je einen Random Integer fr die Original und die Fake Mail. Mit der Random Strategie erhoffen wir uns eine bessere Laufzeit und kleineren Memory Footprint falls man nach mehreren Kollisionen sucht.

2.3.2 Datenstrukturen

Wir haben zwei Hashmaps, eine fr alle Variationen der Original-Mail und eine fr die Fake-Mail. Fr den Key benutzen wir jeweils den Hashwert und als Value wird der Variations-Integer dieser Mail gesetzt. So knnen wir bei der Kollisionsdetektion durch die Schlssel der Original-Mail-Hashmap

3 Anhang

```
1  package ch.fhnw.apsi.lab1;

   import java.nio.ByteBuffer;
   import java.nio.ByteOrder;

6  import org.bouncycastle.crypto.BlockCipher;
   import org.bouncycastle.crypto.BufferedBlockCipher;
   import org.bouncycastle.crypto.CryptoException;
   import org.bouncycastle.crypto.engines.DESEngine;
   import org.bouncycastle.crypto.modes.PaddedBlockCipher;
11 import org.bouncycastle.crypto.params.KeyParameter;

   public class SimplifiedHash {
     private final static byte[] iv = { (byte) 0b10101010, (byte) 0
         b10101010, (byte) 0b10101010, (byte) 0b10101010, (byte) 0b10101010
         , (byte) 0b10101010, (byte) 0b10101010,
         (byte) 0b10101010 };
16
     private byte[] preprocess(byte[] input) {
       int mLength = input.length;
       byte[] length = ByteBuffer.allocate(8).putLong(mLength).array();
       int r = 8 - mLength % 8; // calculate the rest you need to make the
21                     // input divisible by 8
       byte[] out = new byte[mLength + r + 8];

       // Copy
       for (int i = 0; i < mLength; i++)
26       out[i] = input[i];
       // padding
       if (r > 0)
         out[mLength] = -128; // 0b1000 0000

31     for (int i = 1; i < r; i++)
         out[mLength + i] = 0;
```

```
    // add message length
    for (int i = 0; i < 8; i++)
      out[out.length - 8 + i] = length[i];

    return out;
  }

  private long create(byte[] input) {
    BlockCipher engine = new DESEngine();
    @SuppressWarnings("deprecation")
    BufferedBlockCipher cipher = new PaddedBlockCipher(engine);
    byte[] desOut = new byte[16];
    byte[] hash = new byte[8];
    byte[] previousHash = iv.clone();

    for (int i = 0; i < input.length; i += 8) {
      KeyParameter p = new KeyParameter(previousHash);
      cipher.init(true, p);
      desOut = new byte[cipher.getOutputSize(8)];
      int outputLen = cipher.processBytes(input, i, 8, desOut, 0);

      try {
        cipher.doFinal(desOut, 0);

        // xor magix
        for (int j = 0; j < hash.length; j++)
          hash[j] = (byte) (desOut[j] ^ desOut[j + 8] ^ previousHash[j
              ]);

        // swap
        byte[] tmp = hash;
        hash = previousHash;
        previousHash = tmp;

      } catch (CryptoException ce) {
        System.err.println(ce);
      }

    }

    ByteBuffer buffer = ByteBuffer.wrap(previousHash);
    buffer.order(ByteOrder.LITTLE_ENDIAN);
    return buffer.getLong();
  }

  public int createHash(byte[] input) {
```

```
        long hash = this.create(this.preprocess(input));
        int h1 = (int) (hash >>> 32);
81      int h2 = Integer.reverse((int) hash);

        return h1 ^ h2;
    }

86 }
```

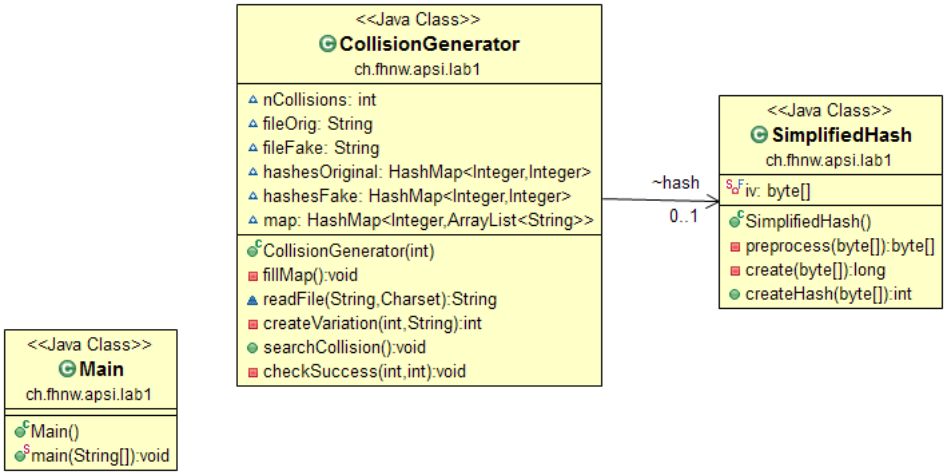

Abbildung 1: Klassendiagramm