# FHNW APSI Lab 1 : Wichtigkeit von $2^n$

Roland Hediger, Jonas Schwammberger

4. November 2013

**Datenstruktur:**

- Statische Teile des Briefes bleiben dasselbe, jeder dynamische Teil hat ein Platzhalter, `{ # 1}`
- 32 Plathalter, 2 Möglichkeiten : Insgesamt $2^{32}$ mögliche Briefe **(mit gleicher Konto Nr)**
- Datenstruktur : `HashMap<Integer,ArrayList<String>`
- Alle möglichkeiten für Platzhalter sind hardcodiert.

```
1  private int createVariation(int combination,
       String file) {

   String tmpfile = new String(file);
   for (int i = 0; i < 32; i++) {
   int bit = combination & 1;
6  String placeHolderString = "{#" + Integer.
       toString(i) + "}";
   ArrayList<String> combinationsForPlaceHolder =
       map.get(i);
   tmpfile = tmpfile.replace(placeHolderString,
   combinationsForPlaceHolder.get(bit));
   combination >>>= 1;
11 }

   return hash.createHash(tmpfile.getBytes());
   }
```

- `Integer.MAX` = $2^{32}$
- Jeder Brief Variante als Integer zwischen 0 und Integer.MAX
- Möglichkeit für Platzhalter gegeben durch 1 oder 0 in **Binärdarstellung des Integers**
- Nächste Möglichkeit für Plathalter des Briefes wird mittels **vorzeichenloses Bitshift** erreicht

1. Preprocessing : Nehme Byte Array und füge Padding hinzu damit der Länge durch 8 teilbar ist.
2. Hasherzeugung :
3. IV definiert als 8 Byte Array mit 101010...

**Für jeder Block**

1. BufferedBlockCipher initializieren mit IV
2. Output Array für DES (16 Byte) initializieren mithilfe von BufferedBlockCipher.
3. Bytes verarbeiten für diese Iteration.(Processbytes
4. XOR "linke und rechte" Blöcke (i, und i+8 im DES Output Array) und dann XOR mit IV.
5. Swap verfahren : Erzeugte hash wird zu neue IV. (Alles in `byte[]` Format bis jetzt)
6. Postprocessing : Hash um 32 bits schieben, und XOR mit Integer.reverse von hash. (`int` Format).

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

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

```
    try {
16  cipher.doFinal(desOut, 0);

    // xor magix
    for (int j = 0; j < hash.length; j++)
    hash[j] = (byte) ((desOut[j] ^ desOut[j + 8]) ^
        previousHash[j]);
21
    // swap
    byte[] tmp = hash;
    hash = previousHash;
    previousHash = tmp;
26
    } catch (CryptoException ce) {
    System.err.println(ce);
    }
```

```
31 }

   ByteBuffer buffer = ByteBuffer.wrap(previousHash
       );
   buffer.order(ByteOrder.LITTLE_ENDIAN);
   return buffer.getLong();
36 }
```

- 2 Mengen gebildet : 1 mit alle Variationen basierend auf originalem KontoNr, und eine genau so aber mit gefälschte KontoNr.

Statt variationen für alle Kombinationen zu machen, wählen wir 2 beliebige Kombinationen.

- Hashen von den Briefvarianten , prüfen auf gleicheit.
- Ausgabe der Variantennummern und den dazugehörigen Hashes.

Falls keine Gleichheit gefunden wird, wählen wir weitere beliebige Kombinationen in Blocken von 1024 aus der ganzen Menge.

# Danke für Ihre Aufmerksamkeit.