CS4236 Project 2 Report

Machine configuration: Laptop, CPU: i7-3630QM @2.40 GHz, Memory: 8GB DDR3 **Java**
C = 47.9%, S = 382 KB, T = 2.922 s, t = 7.446 s, F = 391

Background

As described in the Project Description, the aim of this project is to use Rainbow Table to find the plaintext given a secure hash digests.

For simplicity, a word in the dictionary is a 24-bit string. Hence, there are a total of 2^{24} possible words. We will be using SHA1 for our hashing algorithm.

Algorithm

```
generateTable() {
  int i = 0;
  while (table.size() < RAINBOWTABLE_SIZE) {
    wordStart = intToBytes(i);
    wordEnd = generateChain(wordStart, i);
    key = bytesToHex(wordEnd);
    if (!table.containsKey(key)) {
      table.put(key, wordStart);
      success++;
    } else {
      collisions++;
    }
    i++;
  }
}

generateChain(byte[] wordStart, int index) {
  byte[] digest = new byte[20];
  byte[] word = wordStart;
  for (int i = 0; i < CHAIN_LENGTH; i++) {
    digest = hash(word);
    word = reduce(digest, i);
  }
  return word;
}
```

The main highlight for this algorithm is to compress the rainbow table file size. In a normal rainbow table, the head will be the message word/plaintext and the ending the ending will be the digest. However, here we will reduce the digest again to get the word. Thus, the rainbow table will store the starting word and ending word. This will reduce the size of the rainbow table by a large margin as the word size is only 24 bits (compared to digest size of 160 bits). Due to this, during the invert function, the program will do an extra hashing function to get the ending word stored in the rainbow table.

Furthermore, the starting word will start from 0 an increase by 1 for each iteration. I do not use random starting word and prefer the ordered starting word. Using random starting word will not improve the accuracy of the cracking but will increase the rainbow table size by a large margin instead. This is because using ordered starting word will result in a better compression of the rainbow table (more similar bytes).

```
reduce(byte[] digest, int i) {
  byte[] word = new byte[3];
  byte[] i_byte = intToBytes(i);
  word[0] = (byte)(digest[0] + i_byte[0]);
  word[1] = (byte)(digest[1] + i_byte[1]);
  word[2] = (byte)(digest[2] + i_byte[2]);
  return word;
}
```

The reduce function is by taking the last 3 bytes of the digest and add it with an iterator depending on the chain length. Some precision loss occurs here when converting int to bytes as int is supposed to be 4 bytes long.

Data

Rainbow Table Row = 47500, Chain Length = 210, Words Found: 479

t, the time taken by `invert`	t = 7.446 s
C, the percentage of words correctly found by `invert`	C = 47.9%
S, the size of Rainbow Table in bytes	S = 382 KB
T, the time taken to compute 2^{23} SHA1 operations	T = 2.922 s
F, speedup factor of the program,	F = 1000 T / t = 391

There are 2 test ran for this program:

1. Using the tail words as the known ending message digest.
This will result in about 99.5% accuracy. The accuracy lost is due to loss in precision when converting int to bytes used in reduce function.

2. Using random words as the known ending message digest
This will result in about 45% accuracy which is the threshold for this project and time taken to by invert function in about 7.5 s. This will still result in F > 380.

Conclusion

For rainbow table, the time, accuracy and space are trade-offs between one another. As the chain length is increased, the time taken will obviously increase as it now has to search the words in a longer chain. However, accuracy will increase as it technically storing more message and digest pairs in the chain. As the table row increase, accuracy will also increase as it can now store more message and digest pairs. However, increasing table row does not have significant effect on the time as searching can be done very fast.