Report

Encryption

- Practical work one

Author: Michael Johansson & Peter Danielsson
Supervisor: Ola Flygt
Semester: VT 2016
Subject: 1DV700

Contents

1 Task one

1.1 A

- Symmetric encryption VS Asymmetric encryption:
  Symmetric encryption uses a shared key that is used for both encryption and decryption. Both parts (sender and receiver) must therefore know the shared key to be able to communicate. A potential challenge with Symmetric encryption is the secure distribution of the shared key to the involved parties. Asymmetric encryption on the other hand, uses a private and public key for carrying out it's work. The sender uses the receivers public key to encrypt the information. Decryption of the information is only possible using the receivers private key which is (hopefully) only known by the receiver. Assymetric encryption has the advantange that it doesn't require a secret exchange of encryption keys to the involved parties.

- Encryption algorithms VS Hash algorithms:
  Hash algorithms takes a text and generates a fixed length of numbers, called hash code. Hash algorithms is designed in a way to make it impossible to reverse the hash code back to it's original text. An example where hash functions comes to good use is secure storage of user credentials in an Operating system. In that way the passwords are protected, even if the Operating system is compromised. Encryption algorithm takes a text and generates a number of characters which is not of a fixed length. The big difference between hash objects and encrypted objects is that encrypted objects can be decrypted back to their original state.

- Compression VS Hashing:
  Compression is when you reduce a file to a smaller size by removing regularities, for example five bit zeroes can be shortened down to one zero. Compression can be reversed as lesslossy or lossy. Lesslossy means no information are lost during the reverse process while lossy can result in some information loss. Hashing is a compression type as well, however it is almost impossible to reverse the process and get back to the original file. Hashing compression is often used for "checksums", if you hash a file that file will always get the same hash as long as the file is not changed in any way.

1.2 B

Steganography is a method of hiding secret information in plain sight. One example could be embedding text files into image files, by changing some bits of information in the image file. Another example is constructing a secret message using the last word in every sentence in a text, as demonstrated in exercise 2. Digital Watermarking means that you embed something on to a file to act like a signature rather than a secret message. It can be used to tag files, for example image files, with ownership information so there's an ability to claim copyrights if a file is being used without permission. Encryption involves hiding information by distorting the contents to an unreadable state , while the other two techniques don't change the appearance of the original content. Instead they hide the information by embedding it to the contents of other files.

2 Task two

The hidden message in the text is "George your package ready friday 21st. Room three please destroy this immediately." We took the last word in each sentence and put them together.

3 Task three

3.1 A

The message is "encrypted message".

3.2 B

If someone gets a hold of the cipher text but does not have the encryption key, there's still a possibility of cracking the encryption by exploiting the regularities of the language. First there has to be an assumption that the message is written in a specific language, for example english. Then the regularities of the language can be used to try to decipher the message. For example, the "H" in the cipher text can be assumed to equal "e" in plain text since that's the most used letter in the cipher text.

4 Task four

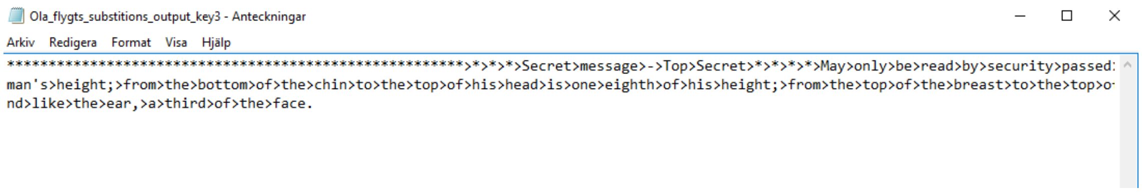
We included the source code for our Java program in the zip file.

5 Task five

We added a file to the "Cipher texts" folder by the name MichaelJ_PeterD.txt.

6 Task six

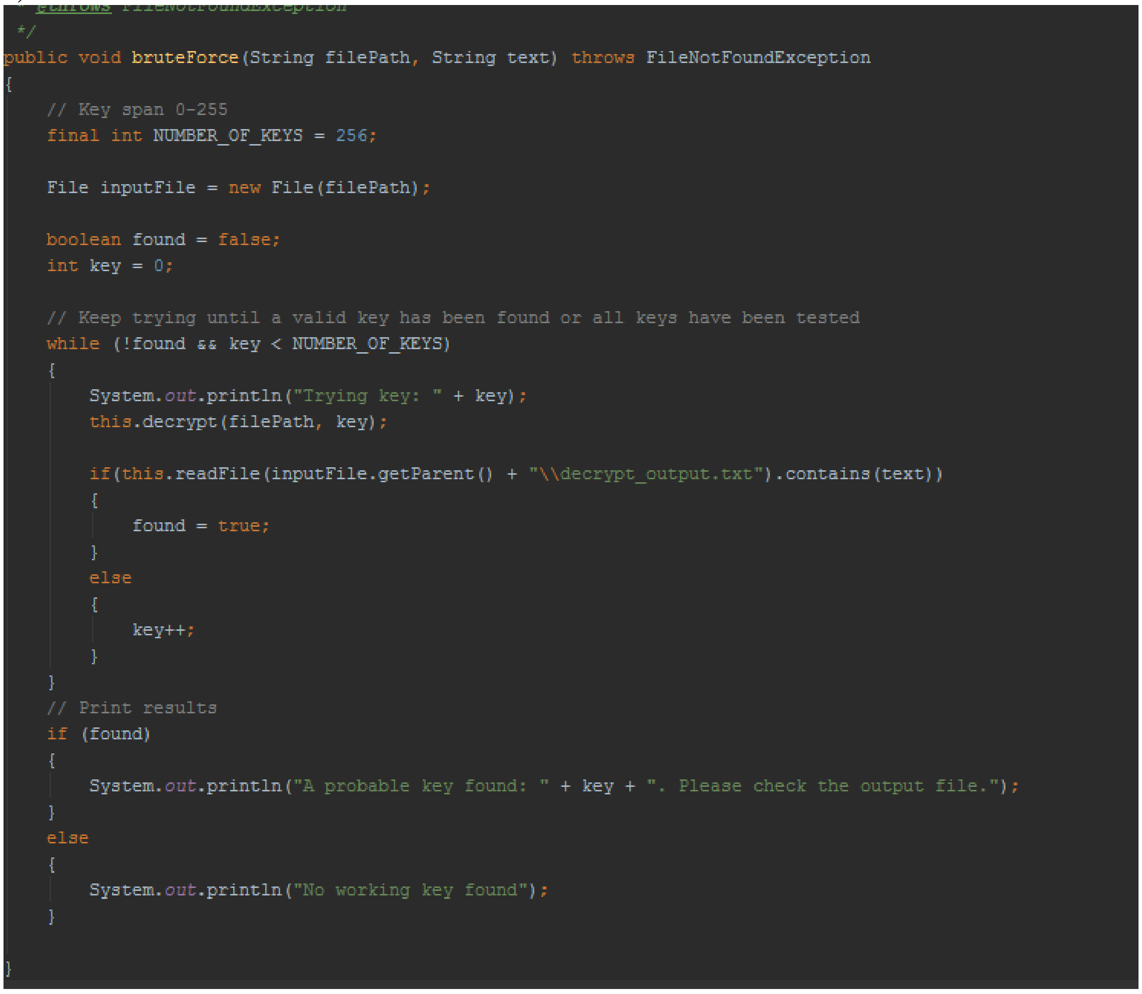
We cracked Ola's substitution cipher. With some manual testing, we could soon see that he had shifted all the characters 3 steps in the ASCII table.



Ola_flygts_substitions_output_key3 - Anteckningar

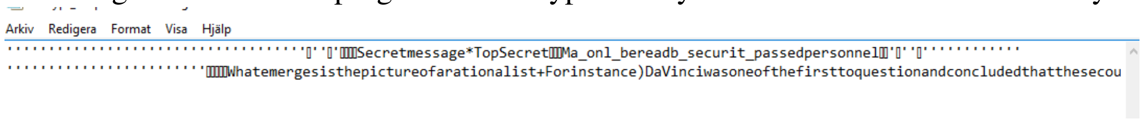
```
**************************************************>*>*>*>Secret>message>->Top>Secret>*>*>*>*>May>only>be>read>by>security>passed:
man's>height;>from>the>bottom>of>the>chin>to>the>top>of>his>head>is>one>eighth>of>his>height;>from>the>top>of>the>breast>to>the>top>o-
nd>like>the>ear,>a>third>of>the>face.
```

We also created our own brute force program in java (that is also included in our source code).



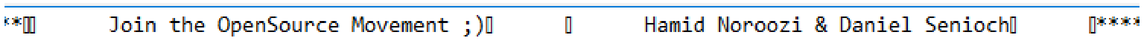
```
 * @throws FileNotFoundException
 */
public void bruteForce(String filePath, String text) throws FileNotFoundException
{
    // Key span 0-255
    final int NUMBER_OF_KEYS = 256;

    File inputFile = new File(filePath);

    boolean found = false;
    int key = 0;

    // Keep trying until a valid key has been found or all keys have been tested
    while (!found && key < NUMBER_OF_KEYS)
    {
        System.out.println("Trying key: " + key);
        this.decrypt(filePath, key);

        if(this.readFile(inputFile.getParent() + "\\decrypt_output.txt").contains(text))
        {
            found = true;
        }
        else
        {
            key++;
        }
    }
    // Print results
    if (found)
    {
        System.out.println("A probable key found: " + key + ". Please check the output file.");
    }
    else
    {
        System.out.println("No working key found");
    }

}
```

Using this brute force program we decrypted Pouya Khast substitution with the key 3.



```
''''''''''''''''''''''''''''''''''''▯''▯'▯▯▯Secretmessage*TopSecret▯▯▯Ma_onl_bereadb_securit_passedpersonnel▯▯'▯''▯'''''''''''''
''''''''''''''''''''''''▯▯▯▯Whatemergesisthepictureofarationalist+Forinstance)DaVinciwasoneofthefirsttoquestionandconcludedthatthesecou
```

We also decrypted Hamid Noroozi and Daniel Senioch subsititution cipher with the key 246.

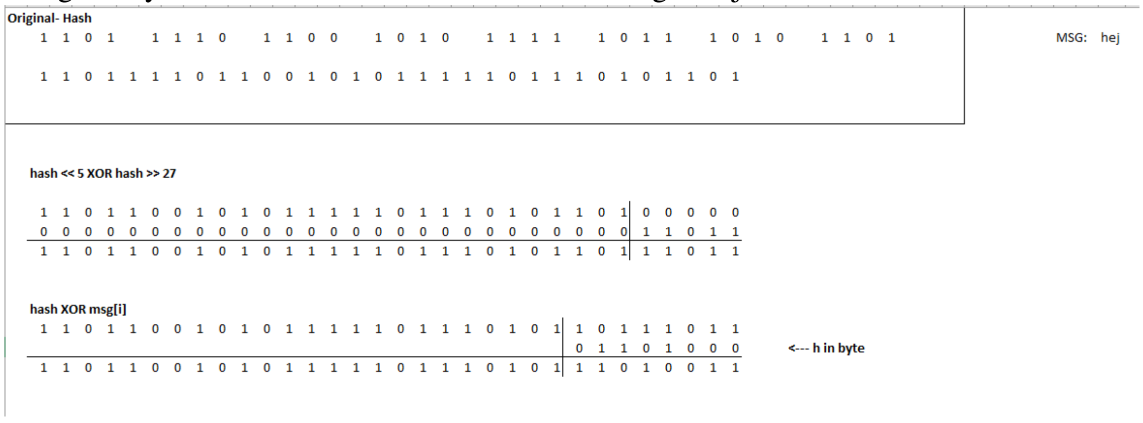


```
**▯▯      Join the OpenSource Movement ;)▯      ▯      Hamid Noroozi & Daniel Senioch▯      ▯****
```

7 Task seven

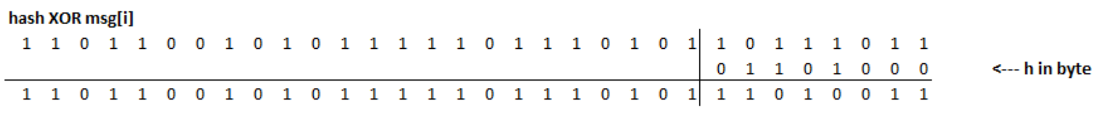
7.1 A

We start with the original hash and do the first 5 left and XOR 27 right and also XOR using the byte from the first character in our message "hej".

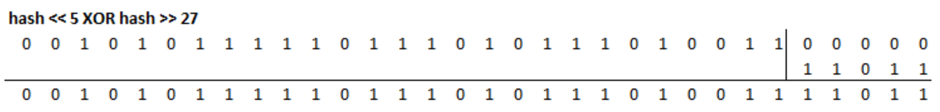


```
Original- Hash
 1 1 0 1   1 1 1 0   1 1 0 0   1 0 1 0   1 1 1 1   1 0 1 1   1 0 1 0   1 1 0 1          MSG:  hej

 1 1 0 1 1 1 1 0 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 0 1


hash << 5 XOR hash >> 27

 1 1 0 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 0 1| 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0| 1 1 0 1 1
 1 1 0 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 0 1| 1 1 0 1 1

hash XOR msg[i]
 1 1 0 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1| 1 0 1 1 1 0 1 1
                                                 | 0 1 1 0 1 0 0 0     <--- h in byte
 1 1 0 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1| 1 1 0 1 0 0 1 1
```

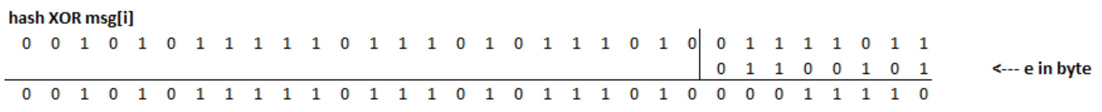
Then we take the updated hash and do the same thing to it.



```
hash XOR msg[i]
 1 1 0 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1| 1 0 1 1 1 0 1 1
                                                 | 0 1 1 0 1 0 0 0     <--- h in byte
 1 1 0 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1| 1 1 0 1 0 0 1 1
```

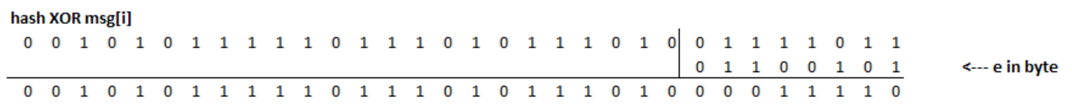


```
hash << 5 XOR hash >> 27
 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0 0 1 1| 0 0 0 0 0
                                                       | 1 1 0 1 1
 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0 0 1 1 1 1 0 1 1
```

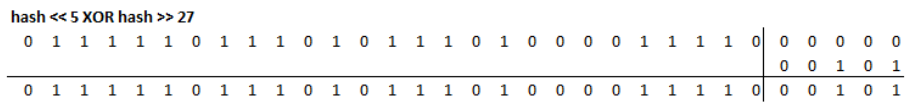


```
hash XOR msg[i]
 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0| 0 1 1 1 1 0 1 1
                                                 | 0 1 1 0 0 1 0 1     <--- e in byte
 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0 0 0 0 1 1 1 1 0
```

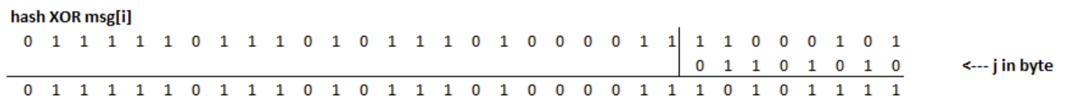
And also for our last character.



```
hash XOR msg[i]
 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0| 0 1 1 1 1 0 1 1
                                                 | 0 1 1 0 0 1 0 1     <--- e in byte
 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0 0 0 0 1 1 1 1 0
```



```
hash << 5 XOR hash >> 27
 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0 0 0 0 1 1 1 1 0| 0 0 0 0 0
                                                       | 0 0 1 0 1
 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0 0 0 0 1 1 1 1 0| 0 0 1 0 1
```



```
hash XOR msg[i]
 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0 0 0 0 1 1| 1 1 0 0 0 1 0 1
                                                 | 0 1 1 0 1 0 1 0     <--- j in byte
 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0 0 0 0 1 1 1 0 1 0 1 1 1 1
```

Finally we return the hash bitwise and adding 0x7FFFFFFF in bits (0111 1111 1111 1111 1111 1111 1111 1111).

7.2 B

The Hash function has some serious limitations. The first XOR adds the first 5 bits to the last part of the hash-code, which results in a repeating pattern. The second XOR only change the last 8 bits of the hash-code. So in conclusion, our opinion is that the generated hash-codes will not be random enough and will result in many collisions over time.