Exercise 1

Fredrik Kortetjarvi & Rohullah Khorami

January 26, 2021

1

Citoday breach this contains 226,883,414 accounts. The breach used socail engineering to get the information they used mailinator to mail fake mails to users in a guitar forum there it could send out. This kind of breaches cant be stoped becasue this is using tricks to trick the brain of people so they give out the information for free. this can only be improved but not fixed.[1]

2

The program generate a picture with a name IOCCC in raytracing. We used cmake to make this into a ppm file then display it in gimp to see the picture that was created see figure 1. We found the makefile on the internet which was a programming contest. The program use functions to make a program to draw graphics into a ppm file that is a picture.[2]

Figure 1: Raytracing.

3

The Key that Eve guessed indeed decrypts the cipher text to "LATER". we checked the result in java program and on pen and paper. The program explaind in Task 4.
$11 = (x - 84) mod 26 \Leftrightarrow T = 11 \rightarrow L, newkey = 19 \rightarrow T$
$0 = (x - 82) mod 26 \Leftrightarrow T = 25 \rightarrow Z, newkey = 16 \rightarrow Q$
$19 = (x - 84) mod 26 \Leftrightarrow T = 21 \rightarrow V, newkey = 20 \rightarrow U$
$4 = (x - 83) mod 26 \Leftrightarrow T = 3 \rightarrow D, newkey = 17 \rightarrow R$
$17 = (x - 72) mod 26 \Leftrightarrow T = 18 \rightarrow S, newkey = 8 \rightarrow I$

First attempt
Plain Text = LZVDS - 11 25 21 3 18
Key = TRTSH - 84 82 84 83 72
Cipher Text = EQNVZ - 4 16 13 21 25

Take out the key
Plain text = LATER - 11 0 19 4 17
Key = TQURI - 19 16 20 17 8
Cipher Text = EQNVZ - 4 16 13 21 25

$T = (x - k) mod 26 + 65$
$78 - 65 = 13 = (x - 84) mod 26 + 65 \Leftrightarrow T = 78 \rightarrow N, x = 71 \rightarrow G$
$69 - 65 = 4 = (x - 82) mod 26 + 65 \Leftrightarrow T = 69 \rightarrow E, x = 86 \rightarrow V$
$86 - 65 = 21 = (x - 84) mod 26 + 65 \Leftrightarrow T = 86 \rightarrow V, x = 79 \rightarrow O$
$69 - 65 = 4 = (x - 83) mod 26 + 65 \Leftrightarrow T = 69 \rightarrow E, x = 87 \rightarrow W$
$82 - 65 = 17 = (x - 72) mod 26 + 65 \Leftrightarrow T = 82 \rightarrow R, x = 89 \rightarrow Y$

Plain text = NEVER - 78 69 86 69 82
Key = TRTSH - 84 82 84 83 72
Cipher text = GVOWY - 71 86 79 87 89

4

```
import java.io.*;
import java.util.Scanner;

public class OtpInputStream extends java.io.InputStream {
        final int first_letter = 65; // 65 = A
        char[] text;
        char[] newtext;
        int method = 1;
        char[] key;// key to encrypt and decrypt
```

```java
int pos = 0;

/**
 * This method will encryption/decryption
 * your messages in OTP or XOR encryption/
     decryption
 *
 * @param text send in a text to decrypt or
     encrypt
 * @param key send in the key to decrypt or
     encrypt
 * @param method select the encryption/decryption
      method
 */
public OtpInputStream (char[] text, char[] key,
    int method) {
        this.text=text;
        this.key=key;
        this.method=method;
        transform(this.method);

}

@Override
/**
 * Reads byte of data from this Input stream
 * @return the next byte of data, or -1 if end of
      the line.
 */
public int read() throws IOException {
        if(pos<newtext.length) {
                return newtext[pos++];
        }else {
                return -1;
        }
}

public char[] getText() {
        return text;
}

public void setText(char[] text) {
        this.text = text;
}

public char[] getNewtext() {
```

```
        return newtext;
}

public void setNewtext(char[] newtext) {
        this.newtext = newtext;
}

public char[] getKey() {
        return key;
}

public void setKey(char[] key) {
        this.key = key;
}
public void reset() {
        pos=0;
}
/**
 * This method take the text to choose to
    encryption/decryption
 *  with XOR or OTP
 * @param method select the method to encryption/
    decryption
 */
public void transform(int method) {
        switch (method) {
        case 1:
                newtext = Encrypt_char(text, key)
                    ;
                break;
        case 2:
                newtext = Decrypt_char(text, key)
                    ;
                break;
        default:
                newtext = Encr_Decr_xor(text, key
                    );
                break;
        }
}
/**
 *
 * @param n is a number of character that should
 * be encrypted/decryption
 * @return this program going to return random
 * character which we use for encryption and
```

```
     decryption
 */

public char[] random_char(int n) {
        char[] character = new char[n];
        for (int i = 0; i < n; i++) {
                character[i] = (char) ((int) (
                    Math.random() * 25) +
                    first_letter);// 65-90

        }
        return character;
}

/**
 *
 * @param text it is plain text that we want to
     encrypt it
 * @param key  is the random character key value
 * @return it is going to return a cipher text
 */
public char[] Encrypt_char(char[] text, char[]
    key) {
        char[] cipher = new char[text.length]; //
            cipher text at the end

        for (int i = 0; i < cipher.length; i++) {
                cipher[i] = (char) (((text[i] +
                    key[i]) % 26) + first_letter);
        }

        return cipher;
}

/**
 *
 * @param cipher it is an encrypted value from
     encryption function
 * @param key     is the same key we used when we
     decrypted the plain text.
 * @return value is going to the message or the
     plain text.
 */

public char[] Decrypt_char(char[] cipher, char[]
    key) {
```

```
        char[] text = new char[cipher.length]; //
            cipher text at the end
        for (int i = 0; i < cipher.length; i++) {
                int num1 = cipher[i];
                int num2 = key[i];
                num2 = num1 - num2;
                if (num2 < 0) {
                        num2 = num2 + 26;
                }
                text[i] = (char) (((num2) % 26) +
                    first_letter);
        }

        return text;
}

/**
 * This function change char to binary
 *
 * @param text is an array of chars that can be
    plain text or cipher text and
 *             even a key if user want to see the
     key
 * @return a string which show 1 and 0
 */
public String[] char_to_binary(char[] text) {
        String[] binary = new String[text.length
            ];

        for (int i = 0; i < binary.length; i++) {
                binary[i] = String.format("%8s",
                    Integer.toBinaryString(text[i
                    ])).replace(" ", "0");
        }
        return binary;
}

/**
 * This function does xor operation by taking to
    char and does xor byte wise.
 *
 * @param value is the plain text or cipher text
    that we want to do the xor
 *              operation on them
 * @param key   is the key value
```

```
     * @return an array of character which can be a
        cipher text or a plaint text.
     */
    public char[] Encr_Decr_xor(char[] value, char[]
      key) {
          char[] result = new char[value.length];

          for (int i = 0; i < result.length; i++) {
                result[i] = (char) (value[i] ^
                  key[i]);
          }

          return result;
    }

}
```

5

All steps to find public key and private key

1.
$p = 7, q = 11$

2.
$N = P * q = 77$

3.
$W = (p - 1)(q - 1) = 60$

4.
To decide an E value thoug we should know that E must be a prime number and GCD(E,W) = 1 and 1 < E <W we asume that E = 53 and GCD(53,60) = 1

5.
D = 1/E mod W => ED = 1 Mod W => D = ((W*i)+1)/E
We check i value step by step or we count the number of prime numbers from 1 to 53. The i Value must be an Integer. In this situation there are 15 prime number before 53 than the i value become 15.
i = 15 and D = ((60*15)+1)/53 = 17. D = 17

6.
public key = E,N = 53,77
private key = D,N = 17,77

7.

Exemple we want to encrypt a message "M" there M<N and M = 10.

Encryption:
$C = T^{E} mod N$
C = cipher text
T= Message = 10
E = Exponent = 53
N= p*q = 77
$C = 10^{53} mod 77 = 54$
cipher text = 54

Decryption:
$T = C^{D} mod N$
$T = 54^{17} mod 77 = 10$
T = message = 10.

6

Euclidean algorithm is an efficient method for computing the Greatest Common Divisor (GCD) of two integer. The largest number that divides them both without a remainder.
Euclidean alogorithm is used in RSA cipher to find an exponent E so that the E Should not be a factor of $\phi(n)$, in other word $GCD(E, \phi(n)) = 1$ which means that we use Euclidean algorithm to find a prime E that the GCD between Exponent and $\phi(n)$ (in our case it is W in task 5) should be equal to one.

7

References

[1] Tony Hunt. “Inside the Cit0Day Breach Collection”. In: (2020). URL: https://www.troyhunt.com/inside-the-cit0day-breach-collection/.

[2] Matt Zucker. “Most shiny”. In: (2011). URL: https://www.ioccc.org/2011/zucker/hint.html.