Name: Nicholas Low Jun Han

Matric No: A0110574N

CS4236: Project 1 (Report)

This report consists of 2 main portions:

1. Table containing the secret keys that were found
2. Methods employed

**Part A: Table of secret keys**

**19 out of 60 keys found**

|  |  |
| --- | --- |
| **Data Set Number** | **Secret Key Found** |
| A00 | 63 63 88 73 12 |
| A01 | 16 80 1 89 86 |
| A02 | 30 56 14 22 53 |
| A03 | 87 18 61 71 1 |
| A04 | 30 3 76 6 37 |
| A05 |  |
| A06 | 12 12 11 47 8 |
| A07 |  |
| A08 |  |
| A09 |  |
| A10 |  |
| A11 | 21 80 32 79 56 16 54 |
| A12 |  |
| A13 |  |
| A14 | 20 10 34 33 30 65 89 |
| A15 |  |
| A16 | 5 61 52 12 59 76 15 |
| A17 |  |
| A18 |  |
| A19 |  |
| A20 |  |
| A21 |  |
| A22 |  |
| A23 |  |
| A24 |  |
| A25 |  |
| A26 | 58 78 53 44 73 41 10 14 75 |
| A27 | 30 13 20 35 49 64 51 64 88 |
| A28 |  |
| A29 |  |
| A30 | 54 71 11 34 71 28 15 49 32 9 1 |
| A31 |  |
| A32 |  |
| A33 |  |
| A34 | 26 45 22 40 76 59 74 23 73 57 80 |
| A35 |  |
| A36 |  |
| A37 |  |
| A38 |  |
| A39 |  |
| A40 | 6 12 49 66 37 78 66 57 11 84 70 5 44 |
| A41 | 5 62 42 12 42 5 21 72 48 7 17 55 17 |
| A42 |  |
| A43 |  |
| A44 |  |
| A45 | 34 5 47 23 6 37 23 58 32 11 82 49 51 |
| A46 |  |
| A47 |  |
| A48 |  |
| A49 |  |
| A50 |  |
| A51 |  |
| A52 |  |
| A53 | 4 72 39 88 37 32 19 2 24 35 65 15 52 57 84 |
| A54 | 41 25 37 72 54 28 4 41 27 74 10 39 27 81 30 |
| A55 |  |
| A56 |  |
| A57 | 38 35 59 31 74 43 29 74 72 79 6 62 76 59 62 |
| A58 |  |
| A59 |  |

**Part B: Method Employed**

**Fluhrer, Martin and Shamir (FMS) Attack combined with backtracking**

Description:

In the zip file, there are 2 code files that accompany this report, namely:

1. RC4Breaker.java
2. PermutationGen.java

The general FMS attack algorithm (as described in the lecture notes, “Project 1: Background on RC4”, slide 33) is implemented within the RC4Breaker.java file.

To simulate the backtracking property that is essential for the attack to work, the RC4Breaker code takes a text file that contains the various permutations of which key byte to take at each level of the key generation stage. This text file is generated by the PermutationGen.java file, where the attacker can specify the “depth” of the possible keys to generate.

For example: (Suppose secret key of key length 5 is being investigated)

A text file generated by PermutationGen.java (where depth is set as 2):

1 1 1 1 1

1 1 1 1 2

1 1 1 2 1

. . . . .

2 2 2 2 2

Hence, after taking in the above text file, RC4Breaker will then generate keys (of length 5) by reading the file as follows:

For the **first** generated key, RC4Breaker will choose the **most frequent appearing** byte during frequency analysis portion of the algorithm for each of the five bytes that make up the key, since the current permutation is “1 1 1 1 1”.

For the **second** generated key, RC4Breaker will choose the **most frequent appearing** bytes for the 1st, 2nd, 3rd and 4th byte. However, as per the current permutation of “1 1 1 1 2”, the 5th byte of the secret key will be the **2nd most frequent appearing byte**.

Following the process described above, RC4Breaker will then generate the remaining keys based on the current permutation of that iteration of generation process.

Subsequently, for each of the generated keys in the list, the attacker will then run rc4.exe (compiled from the given rc4.C) and match the output data with the given data. (For efficiency purposes, the process of checking is made easier through a custom shell script)

**If any of the keys match**, the attacker has found the secret key.

**Otherwise**, the attacker will then carry out the process of generating keys and checking said list of generated keys again, except with a higher depth specified.