Documentation for the Project 1 of the Cryptography course

Michal Chomo, xchomo01

1 Assignment

The goal of this project is to break an unknown synchronous stream cipher and find the key in the form:

$$KRY\{< 24\ unknown\ ASCII\ characters >\}$$

The assignment includes a directory *in* with one plaintext file and three files encrypted by the unknown cipher:

- bis.txt
- bis.txt.enc
- hint.gif.enc
- super_cipher.py.enc

2 Obtaining the keystream

It can be presumed that the cipher works like the Vernam cipher[2], therefore it is possible to obtain the keystream by XOR-ing the plaintext file *bis.txt* with its encrypted version *bis.txt.enc*. To make sure the keystream is correct it needs to be XOR-ed with the encrypted file *bis.txt.enc* and if the output equals the *bis.txt* file, the keystream is correct.

3 Examining the files

After obtaining the keystream, it can be used to decrypt the files *hint.gif.enc* and *super_cipher.py.enc*, but since it is only 512 bytes long, it has to be prolonged somehow. When the keystream is XOR-ed with *super_cipher.py.enc* the cipher algorithm is revealed and it can be used to prolong the keystream and decrypt all content of both files.

```
for file in *.*
do ./super_cipher.py "KRY(                    )" < $file > $file.enc
done
```

Figure 1: hint.gif

```
SUB = [0, 1, 1, 0, 1, 0, 1, 0]
N_B = 32
N = 8 * N_B

def step(x):
  x = (x & 1) << N+1 | x << 1 | x >> N-1
  y = 0
  for i in range(N):
    y |= SUB[(x >> i) & 7] << i
  return y
```

Figure 2: Keystream generation step

4 Reversing the algorithm

It is evident from the algorithm that each bit of y depends on three bits of x which determine if the bit is 0 or 1 by choosing from SUB array. The zeros and ones in SUB array are always uniquely identifiable by the first two bits of their index in the array e.g. for bits 110 it is certain that it indexes the 1 on the 6th position and not any other 1 in the array. Therefore if the first two bits of x are known then it is possible to determine the next bit by looking at the first bit of y and finding the index of this bit which corresponds with the first two bits of x. Because the first two bits of x are not known, all four combinations must be tried. Each combination is then shifted one bit to the right and trimmed to N bits by applying logical AND to reverse the first part of the algorithm. The correct combination is determined by applying the original step function on it and checking if it equals to y.

5 Using SAT solver for reversion

When using a SAT solver[1], a similar approach of exploiting the aforementioned attribute is used. The difference is that the bits are used as variables and the relations between them are written in conjunctive normal form (CNF). This CNF is given to SAT solver which solves it and determines which variables are true (bit is 1) and which are false (bit is 0). Since the first two bits of x are not known and all 4 combinations have to be tried, there are 4 CNFs and the correct result is checked analogously to the normal reversion.

6 Finding the key

The keystream generation in the file *super_cipher.py* works with 32-byte integer value that is passed to the step function. The initialization lies in converting the key (given as a command line argument) to integer and applying the step function to it $N/2$ times. Therefore to get the original key, the first 32-bytes of the keystream obtained from *bis.txt* and *bis.txt.enc* files have to be converted to integer and passed to reversing step function $N/2$ times.

References

[1] SAT.
https://en.wikipedia.org/wiki/Boolean_satisfiability_problem.

[2] The Vernam Cipher.
http://www.cryptomuseum.com/crypto/vernam.htm.