MysteryTwister C3

THE CRYPTO CHALLENGE CONTEST

The LoRa Cipher
LOGIC RANDOMNESS – Part 1

Author: George Theofanidis

February 2020

Introduction (1/7)

"LoRa", an abbreviation of “Logic Randomness”, is a new cipher that calculates the ciphertext by using a logic expression as a key and a number of binary textfiles as operands for the logic expression.

For a better understanding about logic expressions please have a look at these websites [1], [2], [3], and [4].

In order to implement the LoRa cipher, the following steps must be accomplished.

Introduction (2/7)

Encryption:

1. Convert the plaintext to binary (the result is still a text file but it contains only 0s and 1s, maybe separated with whitespaces which can be ignored).
2. Create $\mathbf{n}$-1 text files, where $\mathbf{n}$ is at least 2. Each file is a group of random bits, having the same length as the plaintext, expressed in binary, as in the previous step.
3. Generate a random logic expression which uses $\mathbf{n}$ different operands where the operator applied to the plaintext is always XOR.

Remark: In mathematics an expression is described using operands and operators. So if we have a XOR b, then a and b are operands, and XOR is the operator.

Introduction (3/7)

4. Calculate the resulting file by applying the logic expression to each bit of all $\mathbf{n}$ binary textfiles (so the plaintext and the $\mathbf{n-1}$ randomly chosen files are the input to this encryption function).
5. The ciphertext is a concatenation of the values of the $\mathbf{n-1}$ randomly chosen files and of the calculated resulting file (so all except the plaintext), separated with a "/" between the files. Send the ciphertext to the receiver, but not the plaintext and not the logic expression.
6. Agree out-of-the-band with the receiver on the key (the logic expression).

Introduction (4/7)

Decryption:

1. Apply the logic expression to all files received within the ciphertext and find the binary plaintext.
2. Convert the binary plaintext to a textfile.

Just with any symmetric cipher, the key has to be exchanged out-of-the-band. So if the encrypted data and the binary textfiles are transmitted with a certain transport medium, the key (here the logic expression) must be transmitted with another transport medium, i.e. via encrypted email or via a messenger like WhatsApp or Wire.

All $\mathbf{n}$ files – the plaintext and the $\mathbf{n}$-1 randomly chosen files – must be involved at the logic expression, either as they are, or inverted.

Introduction (5/7)

For simplicity purposes:

- The logic operator applied to the calculated resulting file is always XOR (eXclusive OR).

The logic expression could be transmitted in either one of two ways: in a mnemonic form or in an explicit, detailed form.

The mnemonic form is a simplified expression where operators are replaced by a certain character. For example, the NOT logic operator is symbolized as the mnemonic symbol ‘.

The logic expression is calculated from left to right, in sequential order, not following the order of operator precedence.

Introduction (6/7)

Assuming the number of the given files is n, then

- The files can be used as they are, or inverted with the NOT operator, so there are already 2^n operator choices.
- There are on top $n-1$ logic operators in two modes. In the simple mode, an operator can be either AND, OR, or XOR. In the extended mode, an operator can take one of the following six forms: AND, OR, NAND, NOR, XNOR, or XOR.

Introduction (7/7)

- To sum up the last two bullets, the set of possible operators is either NOT, AND, OR, XOR
  or NOT, AND, OR, NAND, NOR, XNOR, XOR.
- So, the total combinations to produce a logic expression in simple mode are $2^{(n-1)} \times 3^{(n-2)}$, in the extended mode there are $2^{(n-1)} \times 6^{(n-2)}$ possible expressions. For 26 files, this leads to 9,476,762,676,643,233,792 combinations in simple mode and 158,993,694,406,781,688,266,883,072 combinations in extended mode.

Example – Encryption (1/7)

The example plaintext is “The quick brown fox,jumps over the lazy dog!”.

Written in binary form [5], this is:

```
01010100011010000110010100100000011100010111010101101
00101100011011010110010000001100010011100100110111101
11011101101110001000000110011001101111011110000010110
00110101001110101011011010111000001110011001000000110
11110111011001100101011100100010000001110100011010000
11001010010000001101100011000001011110100111100100100
00011001000110111101100111001000001
```

Example – Encryption (2/7)

We want 3 files a, b, c to be involved.
(So the total number of combinations are $2^2 \times 3 = 12$):

a =

```
10011111100110110101101001110001101001100100111001100
01101001010101011011011011101101110101101010101101000
01111100000111110001101010100100000100100011001000001
01100001111100111101110111111011101111111011000000100 1
11011100100011100110010010001001001111110110010110111
10001011100010101011100011011100010000011110101111010
01010001110111111101001111001011 11
```

Example – Encryption (3/7)

b =

```
11011110011011011010010001010111000101101100110110111
11100100100110100110000111110110000111011011001100010
11010001110000110010111010001011001101011100011010011
10110101000001101010001101111001100011101000011100110
01100001001010111011101011111000000100111110011101010
01100110110001101011111001110000101101100011000000001
10110000101100100011101000000
```

We can choose a random logic expression in three forms:

- in a mnemonic (simplified expression) form as: a'@b'#c
- in a detailed form as: not(a) OR not(b) XOR c
- in choices form as: 22231

Example – Encryption (4/7)

Standardization of the choices form:

- The odd indices of this form deal with the state of the files:
  - If an index is 1, the file is not inverted and stays as it is; if the index is 2, the file is inverted with the not_to_string function
- The even indices deal with the choice of logic operators between the files:
  - in the simple mode of the cipher: 1 for AND, 2 for OR, and 3 for XOR
  - in the extended mode of the cipher: 1 for AND, 2 for OR, 3 for NAND, 4 for NOR, 5 for XNOR, and 6 for XOR

Example – Encryption (5/7)

- In the mnemonic form, logical operators are represented by the sequence of characters on the standard QWERTY keyboard. In simple mode, ! stands for AND, @ for OR, # for XOR; and in advanced mode ! stands for AND, @ for OR, # for NAND, $ for NOR, % for XNOR and ^ for XOR. As already mentioned, in the mnemonic form the symbol ' stands for the NOT operator.
- Security could be raised if a transposition key is added. A value of i.e. 4-3-1-2 would mean to transpose the initial files from starting positions 1-2-3-4 to 4-3-1-2 and then follow the rest of the procedure.

Example – Encryption (6/7)

In order to find the last file, i.e. c, we "solve" the logic expression, so

not(a) OR not(b) XOR c == plaintext,

is converted as

c = ((not(a) OR not(b)) XOR (plaintext)

We submit 3 files (a,b,c) with a certain transmission medium.

We submit the mnemonic logic expression with another transmission medium.

Example – Encryption (7/7)

Here are all combinations of the operators for this example:

No	Mnemonic	Detailed	Choices	Status
1	a!b#c	a AND b XOR c	11131	Wrong
2	a!b'#c	a AND not(b) XOR c	11231	Wrong
3	a@b#c	a OR b XOR c	12131	Wrong
4	a@b'#c	a OR not(b) XOR c	12231	Wrong
5	a#b#c	a XOR b XOR c	13131	Wrong
6	a#b'#c	a XOR not(b) XOR c	13231	Wrong
7	a'!b#c	not(a) AND b XOR c	21131	Wrong
8	a'!b'#c	not(a) AND not(b) XOR c	21231	Wrong
9	a'@b#c	not(a) OR b XOR c	22131	Wrong
10	a'@b'#c	not(a) OR not(b) XOR c	22231	Correct
11	a'#b#c	not(a) XOR b XOR c	23131	Wrong
12	a'#b'#c	not(a) XOR not(b) XOR c	23231	Wrong

Example – Decryption

1. We receive the 3 files and the logic expression.
2. We use the logic expression and the 3 files to calculate the plaintext (in binary).
3. We convert the binary textfile of the plaintext into readable text (ASCII).

Note: All needed calculations can be done, step by step by isolated functions, using the submitted Python script. The syntax for decoding is described in the following slide, reading the ciphertext and the key in choices form.

Note: The syntax for the example and the challenge is different, due to different names of the files involved.

Syntax of the Python Command Line Tool

In order to encrypt / decrypt, keep the program and all needed files in the same folder. The program is written in Python 3.

Decryption for the example:

1. Reading the key from a file:
   python LoRa-01.py -kf key_example.txt -df ciphertext_example.txt
2. Reading the key from a string at the command line:
   python LoRa-01.py –ks 22231 -df ciphertext_example.txt

Remark: If there are n files included in the ciphertext, i.e. $n-1$ binary text files plus the resulting text file, then the lengths of the key in choices form is always $2n-1$.

Challenge (1/2)

Use the 5 binary textfiles (a,b,c,d,e).

The 5 files are contained in a separate file “ciphertext_challenge.txt”. The files are separated by “/”.

The logic expression is unknown.

The total number of combinations to produce this logic expression is:

$$2^4 \times 3^3 = 16 \times 27 = 432$$

The plaintext is in English, it contains spaces and exclamations marks. In this part of the challenge, neither extended mode, nor a transposition key (see page 13) was applied.

Challenge (2/2)

The solution should be the logic expression in mnemonic form.

So if the output of the program is for example

- mnemonic form: a!b@c'#d!e
- detailed form: a AND b OR not(c) XOR d AND e
- choices form: 111223111

send a!b@c'#d!e as the solution.

References

1. https://en.wikipedia.org/wiki/Boolean_expression
2. https://realpython.com/python-operators-expressions/
3. http://openbookproject.net/thinkcs/python/english3e/conditionals.html
4. https://www.geeksforgeeks.org/python-logical-operators-with-examples-improvement-needed/
5. https://www.rapidtables.com/convert/number/ascii-to-binary.html

Additional Files

- ciphertext_example.txt
  ➥ the ciphertext example
- key_example.txt
  ➥ the key example in mnemonic form
- LoRa-01.py
  ➥ Python 3 script to decrypt and encrypt with this cipher. The script is called with either `python LoRa-01.py` or `python3 LoRa-01.py` depending on the system environment.
- ciphertext_challenge.txt
  ➥ the ciphertext of this challenge

The MTC3 team wishes you success!