2021-03-16 theory

March 2021

Summary

Summary of operations introduced on first week. The operators were introduced using baby examples of symmetric cipher, in which we used strings of *5 bits*, i.e. $element \in [0,1]^5$:

- $\otimes$ Xor $\longleftrightarrow$ The Vernam cipher
- $\boxplus$ Addition $\longleftrightarrow$ The Caesar cipher
- <<< Rotation $\longleftrightarrow$ The rot cipher
- $\boxtimes$ Multiplication
- $\otimes$ Tensor product $\longleftrightarrow$ Galois cipher

This operation can be generalized to n *bits*, i.e. $element \in [0,1]^n$:

- $\otimes$ Xor $\surd$ (it's a bitwise operation)
- Addition $\Rightarrow$ mod(2^n): after the sum operation, we only consider the the n *LSB* of the binary representation. (When we take the first n *bits*, it's equivalent to take the reminder of mod(2^n))
- <<< Rotation $\surd$
- $\boxtimes$ Multiplication $\Rightarrow$ mod(2^n): same as for the addition, after the multiplication we have to take only the first n *bits*
- $\otimes$ Tensor product(Galois) $\Rightarrow$ we need additional information about the G(x) polynomial function.

1 ARX cipher

ARX which stands for Addition/Rotation/XOR, is a class of symmetric-key algorithms designed using only the following simple operations: modular addition, bitwise rotation and exclusive-OR. In academia and industry alike, ARX has gained an enormous amount of interest because of its small size and simple operations. ARX ciphers are block ciphers with very interesting advantages such as:

- **fast performance** on PCs;
- compact implementation;
- easy algorithms;
- no timing attacks: in many other ciphers, analyzing the time taken to execute cryptographic algorithms gives useful informations to the attacker in order to work backwards to the input, since the time of execution can differ based on the input;
- functionally completeness (assuming constants included): every possible logic gate can be realized as a network of gates using ARX operations and constants;
- it's a trade of speed and mathematical security.

1.1 RC4

In cryptography, RC4 is a stream cipher designed by Ron Rivest (the "R" of RSA) in 1987. While it is remarkable for its simplicity and speed in software, multiple vulnerabilities have been discovered in RC4, rendering it insecure. RC4 became part of some commonly used encryption protocols and standards, such as WEP in 1997 and WPA in 2003/2004 for wireless cards; and SSL in 1995 and its successor TLS in 1999, until it was prohibited for all versions of TLS by RFC 7465 in 2015, due to the RC4 attacks weakening or breaking RC4 used in SSL/TLS. The main factors in RC4's success over such a wide range of applications have been its speed and simplicity:boxtimes efficient implementations in both software and hardware were very easy to develop.

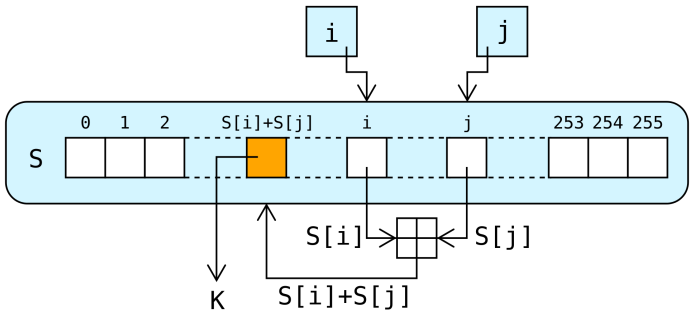
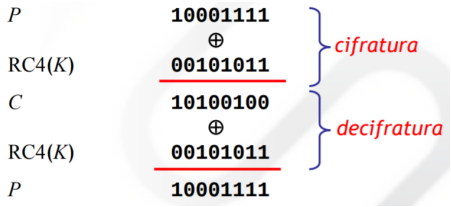


Figure 1: The lookup stage of RC4. The output byte is selected by looking up the values of S[i] and S[j], adding them together modulo 256, and then using the sum as an index into S; S(S[i] + S[j]) is used as a byte of the key stream, K.

1.1.1 Behaviour

RC4 is a function that, starting with a key (1 to 256 octets long), generates a sequence pseudorandom (keystream) used to encrypt and decrypt (by XOR) a data stream. The key (1 to 256 octets long) comes used only in the initialization



phase of the state. Vector register contains one at all times permutation of values from 0 to 255. The register vector is used for subsequent generation of pseudo-random bytes and then to generate a pseudo-random stream which is XORed with the plaintext to give the ciphertext. Each element in the register vector is swapped at least once. The algorithm works in two phases, key setup(Figure 2) and ciphering(Figure 3). Key setup is the first and most difficult phase of this encryption algorithm. During a N-bit key setup (N being your key length), the encryption key is used to generate an encrypting variable using two arrays, register and key, and N-number of mixing operations. These mixing operations consist of swapping bytes, modulo operations, and other formulas. Once the encrypting variable is produced from the key setup, it enters the ciphering phase, where it is XORed with the plain text message to create an encrypted message. XOR is the logical operation of comparing two binary bits. If the bits are different, the result is 1. If the bits are the same, the result is 0. Once the receiver gets the encrypted message, he decrypts it by XORing the encrypted message with the same encrypting variable.

3.1.2 RC4: Init(k)

```
#
# Initizialization of RC4 or Key Schedule Algorithm (KSA)
#     input: a key = [k1,k2,....]  k_i numbers mod 256
#     output: register = [ , , ... , ] array of length 265
#             with numbers mod 256
#


from swap import swap

def Init(key):
    register = [i for i in range(0,256)]
    j=0
    l = len(key)
    for i in range(0,256):
        j = (j + register[i] + key[i%l])%256
        swap(register,i,j)
    return register
```

Figure 2: **Initialization**: In register enter the values from 0 to 255: register[n] = n; Inside key there is a vector (of 256 octets) with the key value(repeating it if the key is shorter); You go through register by exchanging the current element-ith with another determined one using the key.

3.1.4 RC4 ciphering

```
# RC4:
#    input :
#           key = 'ASCII' string
#           Plaintext = 'ASCII' string
#
#    output:
#           ciphertext = array of hexadecimal

from swap import swap
from RC4KSA import Init

def RC4(key,Plaintext):
    register = Init(key)
    i=0
    j=0
    ciphertext=[]
    for r in range(0,len(Plaintext)):
        i = (i+1)%256
        j = (j + register[i])%256
        register = swap(register,i,j)
        cr = Plaintext[r]^(register[(register[i]+register[j])%256])
        ciphertext.append(cr)
    return ciphertext
```

Figure 3: **Generation of the keystream**: The vector register is followed, exchanging the element current (i-th) with another determined by current state of register and j.

RC4 Algorithm Strengths:

- The difficulty of knowing where any value is in the table
- The difficulty of knowing which location in the table is used to select each value in the sequence
- A particular RC4 Algorithm key can be used only once
- Encryption is about 10 times faster than DES (Data Encryption Standard)

RC4 Algorithm Weakness:

- The algorithm is vulnerable to analytic attacks of the register vector
- One in every 256 keys can be a weak key. These keys are identified by cryptanalysis that is able to find circumstances under which one of more generated bytes are strongly correlated with a few bytes of the key.
- WEAK KEYS: these are keys identified by cryptanalysis that is able to find circumstances under which one or more generated bytes are strongly correlated with small subset of the key bytes. These keys can happen in one out of 256 keys generated