Algoritmi di Crittografia

Corso di Laurea Magistrale in Informatica

A.A. 2018/2019

Algoritmi di Crittografia

Algoritmi di Crittografia

What is a block cipher?

- A *block cipher* is an encryption/decryption function designed to work on blocks of data of fixed size
- Current *state-of-the-art* block ciphers have blocks of 128 bits
- Whatever the actual size B, this immediately raises a question: what if the length L of a message to be encrypted is different from B?
- For $L < B$ the obvious solution is *padding*
- The answer seems easy also when $L > B$: break the message into $\lceil L/B \rceil$ blocks (with the last possibly padded) and then separately encrypt each block
- Well, not so trivial, as we shall see later

About the block size B

- B must not be too large in order:
  - to minimize the overhead when processing small size messages
  - to increase efficiency (a small-size block fits in the CPU's registers) and to facilitate implementations in dedicated hardware
- B must not be too small either since short blocks may incur in a serious loss of security
- In particular, if B is too small, under the chosen plaintext attack model, the attacker might pre-compile a table with all the plaintexts (2^B entries) indexed by the corresponding ciphertext
- To decrypt a ciphertext would then simply amount to a table lookup
- This is known as the *codebook* attack
- Examples:
  - For $B = 32$, the table would have size $2^{32} \times \frac{32}{8} = 16G$ bytes
  - For $B = 64$, the table would have size 2^{67} bytes, which is unfeasible

A model for encryption

- Encryption and decryption with block ciphers require secret key K
- Typical key sizes are 128 or 256 bits
- Without K, and in light of the Kerkhoff's principle, no secrecy would be possible
- Note that, since plaintext and ciphertext have the same size, from a mathematical standpoint encryption is a *permutation*
- To each possible B bits plaintext must correspond a unique B bits ciphertext, and viceversa, for otherwise the process would not be reversible
- It might be illuminating to see encryption as a two stage process:
  1. given K, select a specific lookup table T_k, with 2^B entries, storing a particular B bits pemutation;
  2. use the plaintext P as an index into T_k and retrieve the corresponding ciphertext

A model for encryption

- As a simple example, with $B = 3$ we could have (among the others) the following two tables, corresponding to as many different keys

000	1	0	1
001	0	1	1
010	0	0	0
011	1	1	0
100	0	1	0
101	1	1	1
110	1	0	0
111	0	0	1

000	1	0	0
001	0	0	1
010	0	0	0
011	1	1	1
100	1	1	0
101	0	1	1
110	0	1	0
111	1	0	1

- In general, the number of different permutations/tables is $2^B!$.
- For $B = 3$ that already means 40320 possible (different) tables
- ... which calls for a key size of at least 16 bits for each table be possibly used

A model for encryption

- For real ciphers, $2^B! >> 2^s$, where s is the key size
- This means that the overwhelming majority of permutations will never be used
- This is not bad, after all, since <u>many</u> permutations are not suitable
- For instance, the second permutation of the previus slides is not suitable for cryptographic purposes since the last bit of the ciphertext and that of the plaintext always coincide
- Good permutations must exhibits the properties of *confusion* and *diffusion*

Confusion and diffusion

- We recall these two fundamental properties required to all secure ciphers
- Diffusion requires that any small, localized change in the plaintext affects all parts of the ciphertext
- Confusion means that any pattern (e.g., two consecutive identical letters) in the plaintext must be hidden in the ciphertext
- One-to-one transformations (e.g., permutations) that exhibit confusion and diffusion properties are not easy to devise

Complexity issues

- For one thing, the two stage model we have seen cannot be implemented in practice
- For instance, for B as small as 48 bits the size of the lookup table would be larger than 10^3 terabytes
- So we need an algorithm that implements the key dependent permutations
- However, a single ("huge") algorithm would be not easy to implement and analyze
- Block ciphers essentially breaks the complexity into simpler pieces
- The bottom line of a block cipher is to apply a relatively simple algorithm a substantial number of times
- Each such application is known as a *round*

Structure of a block cipher

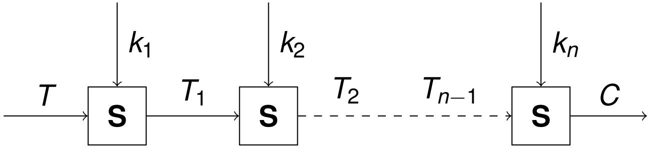
- Let T be our plaintext and let $\mathbf{S}$ be an invertible transformation that apply to plaintexts
- Also, let T_i be the result of i applications to T
- That is, $T_0 = T$, and $T_{i+1} = \mathbf{S}(T_i)$, for $i = 0, 1, \ldots$
- Each transformation also uses a piece of additional data, different for each transformation, which we call the *round key*
- k_i will be the key used in the transformation $T_{i+1} = \mathbf{S}(T_i)$
- Each round key is derived from a (usually longer) *general key* through and algorithm known as the *key schedule*

The general encryption process

- Using the keys, block encryption can then be described as follows

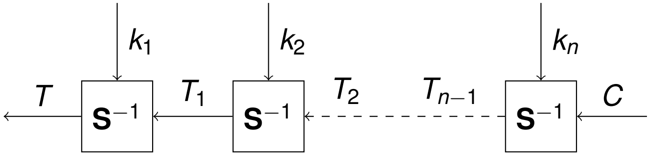
$$\begin{aligned} T_0 &= T \\ T_1 &= \mathbf{S}(T_0, k_1) \\ &\cdots \\ C &= T_n = \mathbf{S}(T_{n-1}, k_n) \end{aligned}$$

  where n is (somewhat) arbitrary and C is the ciphertext
- ... or, using a schematic diagram,



The decryption process

- The transformation **S** is (relatively) simple and easy to invert
- If $\mathbf{S}^{-1}$ denotes the inverse, then we have $T_{i-1} = \mathbf{S}^{-1}(T_i, k_i)$
- This means that the schematic diagram for the decryption process can be obtained by simply “reversing the arrows” in the encryption schema

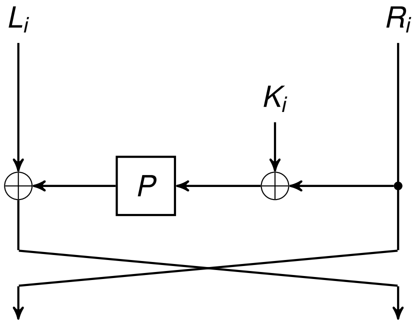


- Indeed, as we shall see, there is not even need for an inverse transformation $\mathbf{S}^{-1}$ ...

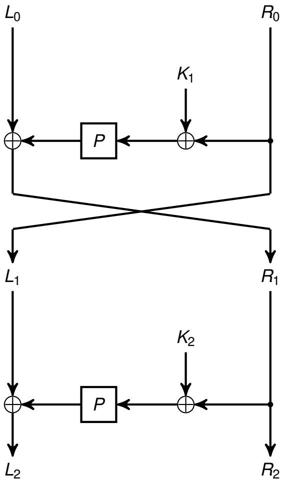
Algoritmi di Crittografia

A closer look into **S**

- The following is the structure of a much simplified block (to begin with)
- Note that the input T_i (as well as the output) is split into two halves, L_i and R_i



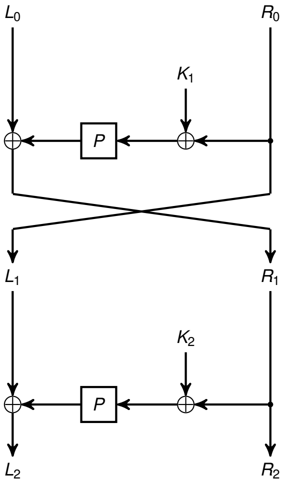
A complete (though exceedingly simple) block cipher



- The cipher is made of just two rounds
- The outputs of the last round “do not cross”
- For simplicity, define $F_K(X) = P(X \oplus K)$. Then:

$$\begin{aligned} L_1 &= R_0 \\ R_1 &= L_0 \oplus F_{K_1}(R_0) \\ L_2 &= L_1 \oplus F_{K_2}(R_1) \\ R_2 &= R_1 \end{aligned}$$

A complete (though exceedingly simple) block cipher



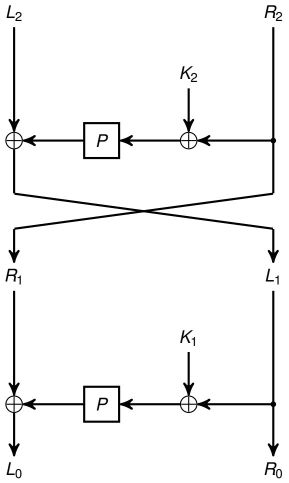
- We can easily "invert" the equations:

$$\begin{aligned} L_1 &= R_0 \\ R_1 \oplus F_{K_1}(R_0) &= L_0 \oplus F_{K_1}(R_0) \oplus F_{K_1}(R_0) \\ L_2 \oplus F_{K_2}(R_1) &= L_1 \oplus F_{K_2}(R_1) \oplus F_{K_2}(R_1) \\ R_2 &= R_1 \end{aligned}$$

- Simplifying and rearranging:

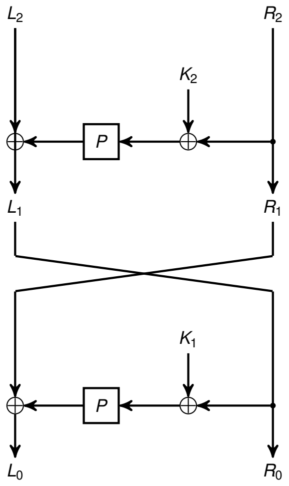
$$\begin{aligned} R_1 &= R_2 \\ L_1 &= L_2 \oplus F_{K_2}(R_2) \\ R_0 &= L_1 \\ L_0 &= R_1 \oplus F_{K_1}(L_1) \end{aligned}$$

A complete (though exceedingly simple) block cipher



$$\begin{aligned} R_1 &= R_2 \\ L_1 &= L_2 \oplus F_{K_2}(R_2) \\ R_0 &= L_1 \\ L_0 &= R_1 \oplus F_{K_1}(L_1) \end{aligned}$$

A complete (though exceedingly simple) block cipher



- If we regard the intermediate values (i.e., here just L_1 and R_1) as the output of the previous block, rather the input of the current block, ther this set of equations

$$\begin{aligned} R_1 &= R_2 \\ L_1 &= L_2 \oplus F_{K_2}(R_2) \\ R_0 &= L_1 \\ L_0 &= R_1 \oplus F_{K_1}(L_1) \end{aligned}$$

can be computed exactly by the same circuit (see left)

- The only difference is the order of the round keys, which must be reversed

The bitarray structure in Python

- Makes it easy to manipulate array of bits/booleans
- In particular, supports bitwise operations
- For details, see project page
  https://pypi.org/project/bitarray/
- Examples:

```
>>> from bitarray import bitarray
>>> a = bitarray(endian='big')
>>> a.append(1)
>>> a.extend([0,1,1])
>>> a
bitarray('1011')
>>> b = a^bitarray('1111',endian='big')
>>> b
bitarray('0100')
```

Python code for a single round

```
def BCRound(L,R,rk,P):
'''Implements a single round of our
block cipher. Output swap must
be done by the caller
'''
from copy import copy
T = copy(R)^rk # xor of R and round key
T = perm(T,P)
L ^= T
return L, R
```

Python code for the cipher

```
def cipher(block, blockkey, P):
    '''Implements our block cipher, both
       encryption and decryption. blockkey
       is typically a generator (or an
       iterable) that returns the round keys
    '''
    from copy import copy
    lbk = len(block)
    assert lbk%2==0
    L = copy(block[:int(lbk/2)])
    R = copy(block[int(lbk/2):])
    for rk in blockkey:
        R, L = BCRound(L,R,rk,P)
    R.extend(L)
    return R
```

Other implementation issues

- Our cipher may work on bitarrays (the input blocks) of even length 2ℓ
- A general key K is required of length 2ℓ
- Given K, we implements two functions that return generators for the encryption and decryption round keys
- Round keys have length ℓ
- The number of round keys generated determines the number of rounds of the block cipher
- Round keys are obtained by rotating the general key and returning its first half bits
- The last component is a function that reorders bitarrays according to a specified permutation

Generator for the encryption keys

```
def enc_keys(blockkey,span=3):
    lbk = len(blockkey)
    assert lbk%2==0 and lbk>=span
    from copy import copy
    bk = copy(blockkey)
    # gcd is the greatest common divisor function
    for j in range(int(lbk/gcd(lbk,span))):
        # Perform a right rotation of span bits
        bk[:span],bk[span:] = bk[lbk-span:],\
                              bk[:lbk-span]
        yield bk[:int(lbk/2)]
```

- The generator for the decryption keys simply returns the same keys but in reverse order

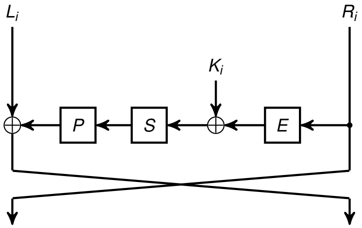
A sanity check for our block cipher

```
def sanity_test(blklen=32,span=3):
    P = derangement(16) # P is a derangement
    plaintext = bitarray.bitarray(endian='big')
    plaintext.frombytes(urandom(2))
    key = bitarray.bitarray(endian='big')
    key.frombytes(urandom(4))
    f = enc_keys(key,span)
    ciphertext = cipher(plaintext,f,P)
    g = dec_keys(key,span)
    cleartext = cipher(ciphertext,g,D)
    if plaintext != cleartext:
        print(plaintext, ciphertext, \
              cleartext, key, D)
```

Algoritmi di Crittografia

Feistel construction

- This refers to the general block structure (and block combination) we have also adopted for our simplified block cipher
- Different ciphers based on Feistel's blocks differ in the function F
- Our example cipher is insecure because our F function is really weak: $F_K = P(R \oplus K)$
- The following is the structure of the famous (and now obsolete) *Data Encryption Standard*, or simply *DES*



- DES is a 64 block cipher with $|L| = |R| = 32$.
- E is an *expansion* block (from 32 to 48 bits)
- S is *substitution* box

DES

- The encryption/decryption key K used by DES is “only” 56 bit long
- From K, the 16 rounds keys are obtained which are 48 bit long
- Round keys are xor-ed with the expanded (by the E block) right-half of the input
- The S-box brings the size back to 32 and this is the most important transformation of the whole Feistel block
- The S-box is actually a set of eight small boxes that map 6 bits to 4 bits each
- The mapping is implemented as a table lookup, with the table actually a 4×16 matrix with 4-bit entries
- The first and last input bit determine the index the row i, while the other bits (bits 2 to 5) pin down the column index j
- Entry i, j is then returned as the output of the small box

DES

- Besides the simplicity of design (thanks to its structure, as we have already seen), the cryptographic properties of a Feistel construction depends on various aspects:
  1. Inverting the *L* and *R* semiblocks let the various parts of the message mix together
  2. The S-box are responsible to ensure non-linear dependence between plaintext and ciphertext (see later)
  3. The xor operations that occurs during encryption ensure that the key is mixed with the plaintext, which is of course crucial to secrecy
  4. Exapnsion (i.e., the replication of some bits) and contraction of the plaintext being encrypted, together with the permutation and the number of rounds, ensure the diffusion of small changes (even of one single bit) in the plaintext

About linearity

- Suppose that the funcion F is composed of just a permutation (i.e., no expansion and contraction)
- Also, to make things easy, suppose that the block size is 4 bits and (hence) that each round key is 2 bit long
- In this case, we can express the output (L', R') of block as follows (also taking inversion into account):

$$\begin{aligned} \ell'_1 &= r_1 \\ \ell'_2 &= r_2 \\ r'_1 &= \ell_1 \oplus k_{i2} \oplus r_2 \\ r'_2 &= \ell_2 \oplus k_{i1} \oplus r_1 \end{aligned}$$

and this is simply a set of linear equations over the field $\mathbf{Z}_2$

Problems with DES

- The key size is small to resist to the power of modern computers; successful brute force attacks have been reported
- 3DES is an attempt to cope with this problem, using 112 or 168 bit keys
- The block size (64 bits) is too small as well, which leads to inefficient implementations
- DES exhibits a “funny” behavior on some inputs that make it easy to distinguish from any ideal block cipher
- Just to mention one, the so called *complementation* property, it holds that:

$$\overline{C} = \overline{E(P, K)} = E(\overline{P}, \overline{K})$$