Algoritmi di Crittografia

Corso di Laurea Magistrale in Informatica

A.A. 2018/2019

Algoritmi di Crittografia

1. Block Ciphers (2)
   - Slide attacks
   - The current block cipher standard: AES
   - Modes of operation for block ciphers
   - Two powerful attacks

Algoritmi di Crittografia

Why not to use the same key at each block?

- The key schedule algorithm must be known, according to Kerkhoff's principle
- The round keys do not add additional randomness, with respect to that of the general key
- So, why not to use the same secret key at each iteration?
- Among other reasons, ... because that would expose the block cipher to a particular type of attack, known as *slide attack*

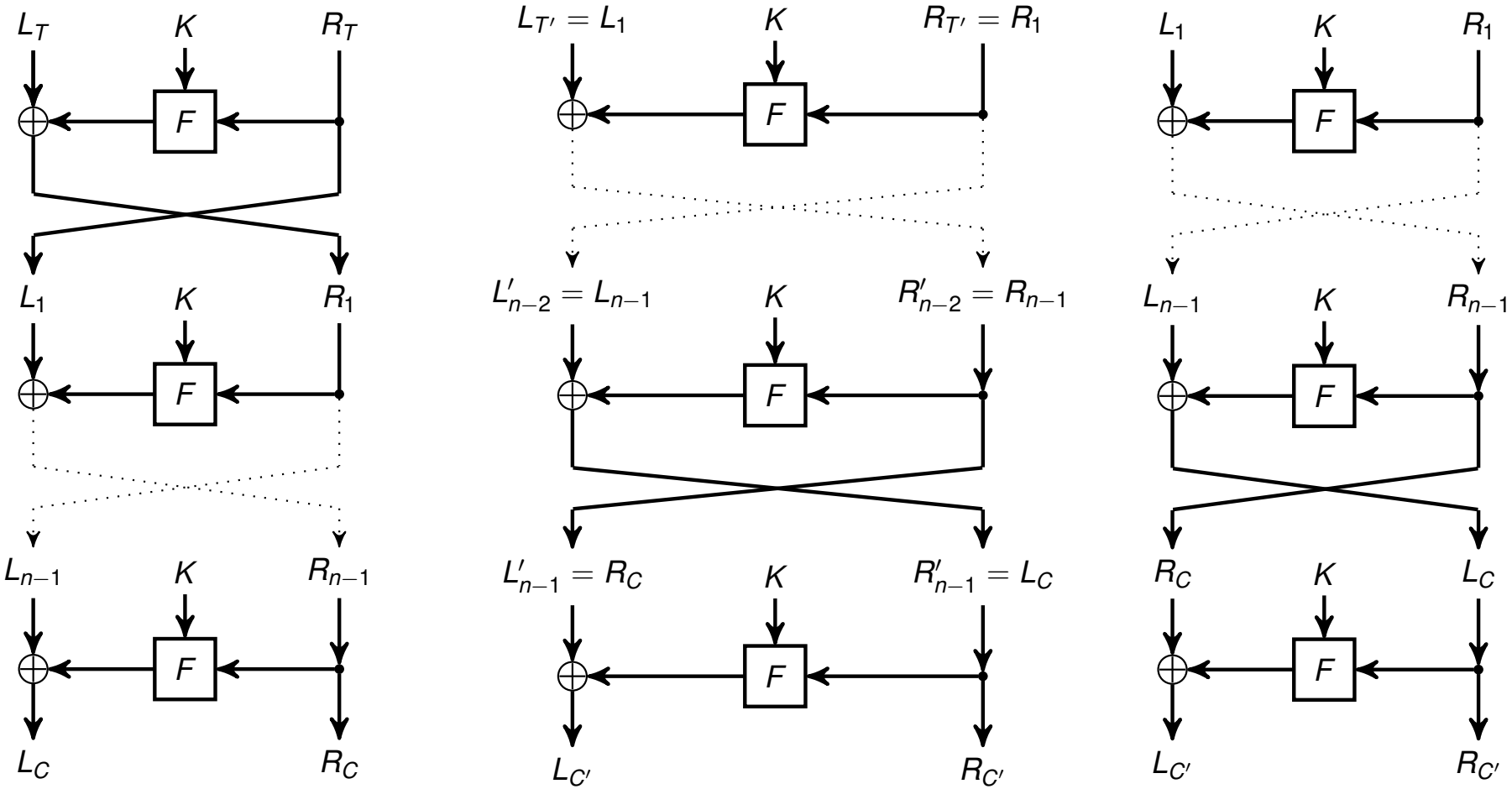
Slide attacks

- Let $T_{i+1} = \mathbf{S}(T_i, K_{i+1})$ denote the transformation performed by a single round of a generic block cipher, $i = 0, 1, \ldots$
- For block ciphers based on the Feistel's construction we know that $\mathbf{S}(T_i, K_{i+1}) = \mathbf{S}(L_{T_i}||R_{T_i}, , K_{i+1}) = R_i||F(L_{T_i}, R_{T_i}, K_{i+1})$, for some specific function F
- Here, and in the following slides, we make the following positions:
  1. if X is any block of n bytes (with n even), then L_X and R_X denote the first and last half bytes of X
  2. the symbol $||$ denote block concatenation, hence $X = L_X||R_X$

Slide attacks

- Suppose now that $K_i = K$, i.e., the same secret key is used at each round
- A *slid pair* is a pair of "plaintexts", T and T', such that $T' = \mathbf{S}(T, K)$, i.e., T' coincides with the result of one round applied to T
- Under these circumstances, it is easy to see that, for the corresponding ciphertexts $C = L_C||R_C$ and C', it also hold that $C' = \mathbf{S}(R_C||L_C, K)$ (see next slide).

Sliding a copy of the block cipher



Weak permutations

- The round transformation **S** is said a *weak permutation* if, given two equations:

$$\begin{aligned} Y_1 &= \mathbf{S}(X_1, K) \\ Y_2 &= \mathbf{S}(X_2, K) \end{aligned}$$

  it easy to compute K
- But then, if T and T' are slid pairs, two possible equations are:

$$\begin{aligned} T' &= \mathbf{S}(T, K) \\ C' &= \mathbf{S}(R_C||L_C, K) \end{aligned}$$

Recognizing slid pairs

- To mount a successful attack, a CPA-able attacker must obtain a slid pair and then, if the round permutation is easy, extract the key
- Now, recall that $T = L_T||R_T$ and, similarly, $T' = L_{T'}||R_{T'}$, $C = L_C||R_C$, $C' = L_{C'}||R_{C'}$
- Now, in case of Feistel's construction, slid pairs are relatively easy to detect, since $L_{T'} = R_T$ and $R_{C'} = L_C$
- The above can be used as a "filtering condition", in the following sense: if T and T' form (supposedly) a slid pair, then we first check that:

$$\begin{aligned} L_{T'} &= R_T \\ R_{C'} &= L_C \end{aligned}$$

and, if the above does not hold, T and T' can be immediately "rejected"

Recognizing slid pairs (cont.d)

- If the above check is “passed”, then we proceed to solve:

$$\begin{aligned} R_{T'} &= L_T \oplus F(K, R_T) \\ L_{C'} &= R_C \oplus F(K, L_C) \end{aligned}$$

- Under the CPA model, the attacker can then generate N plaintext/ciphertext pairs and check whether, among these, there is indeed a slid pair
- To issues arise immediately: choose a “good” value of N and avoid $O(N^2)$ attempts to detect a slid pair
- To solve these issues we use ... both mathematics and algorithmics!

The slide attack

- Let $\mathcal{P} = (P_1, C_1), \ldots, (P_N, C_N)$ be a set of N plaintext/ciphertext pairs, where $P_i = L_{P_i} || R_{P_i}$ and, similarly, $C_i = L_{C_i} || R_{C_i}$, $i = 1, \ldots, N$
- The idea is to store $\mathcal{P}$ as a lookup table indexed (and kept sorted) by keys $R_{P_i} || L_{C_i}$ and values $L_{P_i} || R_{C_i}$.
- Now, to check whether given plaintext/ciphertext pair (P_j, C_j) has a slid mate in $\mathcal{P}$ amounts to performing a lookup with key $L_{P_j} || R_{C_j}$
- If $L_{P_j} || R_{C_j} = R_{P_i} || L_{C_i}$, for some i, then proceed to check whether (P_i, C_i) and (P_j, C_j) form a slid pair by solving:

$$\begin{aligned} R_{P_j} &= F(R_{P_i}, K) \oplus L_{P_i} \\ L_{P_j} &= F(L_{C_i}, K) \oplus R_{P_i} \end{aligned} \tag{1}$$

The slide attack (cont.d)

- Let us “visualize” the search:

keys	values
...	...
$R_{P_i}\|\|L_{C_i}$	$L_{P_i}\|\|R_{C_i}$
...	...
$R_{P_j}\|\|L_{C_j}$	$L_{P_j}\|\|R_{C_j}$
...	...

- For each value $L_{P_j}||R_{C_j}$ in the second column, a plaintext/ciphertext pair that satisfies the filtering condition can be found by searching a key $R_{P_i}||L_{C_i}$ in the first column such that $R_{P_i}||L_{C_i} = L_{P_j}||R_{C_j}$
- Since first column is sorted, each search has logarithmic cost in the table size

The slide attack (cont.d)

- Now, how large must N be chosen?
- Suppose the block size is 2^n bits, which implies there are as many plaintext/ciphertext pairs
- By the birthday paradox, if we generate $N = \theta(\sqrt{2^n}) = \theta(2^{n/2})$ plaintext/ciphertext pairs uniformly at random, then we expect to find about one slid pair
- More precisely, we expect that, among the pairs that pass the table lookup check, about one admits a solution K to equations (1) and that about one does equations

Algoritmi di Crittografia

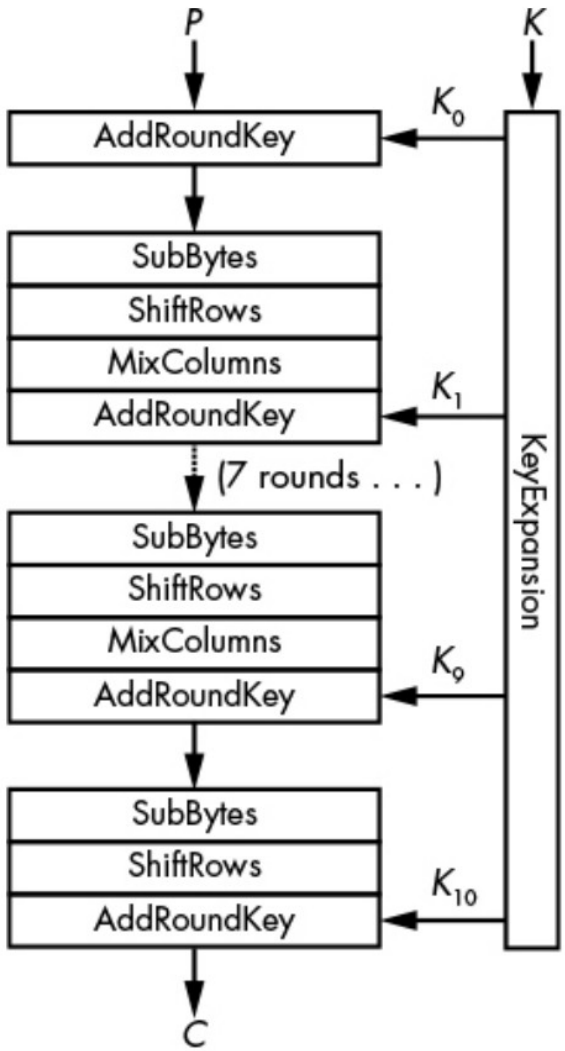
AES in brief

- *American Encryption Standard* is ... a Belgian cipher!
- Designer are two Belgian cryptographers, Rijmen and Daemen, who gave it the name of *Rijndael cipher*
- Became a standard as a winner of a competition launched by *NIST* (National Institute of Standards and Technology)
- Original Rijndael cipher works with blocks and keys made of $32i$ bits, for $i \in \{4, 5, 6, 7, 8\}$
- Standard AES has fixed block size 128 and key size 128, 192, or 256

AES: general structure

- Like DES (and virtually all modern block ciphers), AES works in rounds
- Based on substitution-permutation network design instead of Feistel's construction
- Number of rounds is either 10 (for 128 bit keys), 12 (192 bit keys) of 14 (256 bit keys)
- Each round (but the last) is made of four operations that transform the *internal state*
- Initial internal state is the plaintext, final internal state is the ciphertext
- 128 bits of the internal state are organized as a 16×16 matrix, stored in column major order
- A further component is (clearly) the key schedule algortihm

AES: high level architecture



Source: J.P. Aumasson, Serious Cryptography, No Starch Press (2018)

AES: high level architecture

- The *key schedule* provides different round keys (to avoid slide attacks), each of 128 bits
- The *AddRoundKey* operation performs bitwise $\oplus$ between the key and the internal state
- The *ShiftRows* rotates row i exactly i positions to left (recall each entry is a byte)

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

$\Rightarrow$

b_0	b_4	b_8	b_{12}
b_5	b_9	b_{13}	b_1
b_{10}	b_{14}	b_2	b_6
b_{15}	b_3	b_7	b_{11}

ShiftRows avoids possible codebook attacks since with no shift AES would behave as four “parallel” block cipher with 32 bit blocks

AES: high level architecture (cont.d)

- The *Mixcolumn* step amounts to performing the same linear transformation to the four columns separately (and, potentially, in parallel)
- These are designed so that each byte of the column affects all the bytes in the result
- MixColumns (like ShiftRows) helps to enforce diffusion and, also, guards against codebook attacks to each byte separately
- The most interesting step is however *SubBytes*
- SubBytes (i.e. byte substitution) replace each byte with a different byte using (for all the 16 bytes separately) the same procedure and the same data
- SubBytes provides (high) nonlinearity to AES

A focus on SubBytes

- This is a 2-step process applied to each byte separately
- A byte $b = b_7 b_6 \ldots b_0$ is regarded as a polynomial in $GF(2^8)$, i.e. the unique (up to isomorphisms) Galois Field with 256 elements
- In other words, b represents the polynomial $b(x) = b_7 x^7 b_6 x^6 \ldots b_0$
- Operations in $GF(2^8)$ are done modulo an irreducible polynomial $q(x)$ of degree 8 with coefficients in $\mathbf{Z}_2$:

$$q(x) = x^8 + x^4 + x^3 + x + 1$$

- Each element of $GF(2^8)$ has a multiplicative inverse mod $q(x)$. For instance, the inverse of $p(x) = x^7 + x^5 + x^2 + 1$ is $p^{-1}(x) = x^7 + x^5 + x^4 + x^3$
- As the first step of SubBytes, the coefficients of $p^{-1}(x)$ replace those of $p(x)$, i.e., $B8_{16}$ replaces $A5_{16}$

A focus on SubBytes (cont.d)

- Clearly, in any real AES implementation, the inverse of each element in $GF(2^8)$ is pre-computed and stored in a lookup table
- There is still one serious problem, though: $b = 0$ (i.e., the zero polynomial) does not have multiplicative inverse
- To cope with this problem, an affine transformation is applied a second step of SubBytes
- Given b and the corresponding polynomial $b(x)$, let c denote the coefficients of $b^{-1}(x) \pmod{q(x)}$
- Since $b = 0$ does not have an inverse, we assume $c = 0$ is the result of first step applied to $b = 0$
- The second step is then an affine trasformation applied to c:

$$d = Ac \oplus e$$

and d is final result of SubBytes

A focus on SubBytes (cont.d)

- The matrix A in AES is a Toeplitz matrix obtained by repeatedly rotating the row vector $(1,1,1,1,1,0,0,0)$, i.e.,

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- The translation vector e is $(0,1,1,0,0,0,1,1)^T$
- Everything is chosen so that $b \neq d$ and $b \oplus d \neq FF_{16}$

Summing up

- Is AES secure? For the future, who knows? However, no practical attack has been discovered yet. Only “theoretical” attacks that are still unfeasible, even on modern high-performance computing platforms
- Among the pros, the possibility of highly-efficient implementations, both in hardware and in software. In particular, many logical operations performed bt AES are amenable to parallel implementation
- As for the cons, differently from cipher based on Feistel’s construction, decryption in AES is quite different from encryption
- To decrypt, AES needs an inverse lookup substitution table and inverse matrix; it also needs inverse linear tranformations for the inverse MixColumns step

Example usage (Python)

```
from Crypto.Cipher import AES
from Crypto import Random

key = 'not a clever key'.encode('utf-8')
prng = Random.new()
iv = prng.read(AES.block_size)
cipher = AES.new(key, AES.MODE_CFB, iv)
cipher.encrypt(b'short secret message')
```

Example usage (OpenSSL)

```
> echo ``short secret message'' > plaintext.txt
> openssl enc -aes-128-cbc -in plaintext.txt
-out ciphertext.bin
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:

> openssl enc -aes-128-cbc -d -in ciphertext.bin
enter aes-128-cbc decryption password:
short secret message
```

Algoritmi di Crittografia

1. Block Ciphers (2)
   - Slide attacks
   - The current block cipher standard: AES
   - Modes of operation for block ciphers
   - Two powerful attacks

What's an operation mode (remind)

- In the context of block ciphers, the mode of operation is "simply" the algorithm used to orchestrate encryption and decryption of messages that do not precisely fit a single block
- Most modes requires that the length of the plaintext being encrypted is a multiple of the block size of the (block) cipher used
- Thus, in general, some *padding* is required
- Padding must be reversible: this means that, after padding, it must still be possible to recover the original message (i.e., removing the padding data)
- If padding is often required, it is indeed better to assume that some padding is always present
- This means that, when the length of the original plaintext P is already a multiple of the block length, then a full block of padding data is enqueued to the P

Some invertible padding scheme

- Suppose b is the number of bytes in a block: $b = 8$ for DES and $b = 16$ for AES are two possibilities
- For definiteness, here we assume $b = 16$
- One very simple padding scheme is the following:

  Append 80_{16} to P, followed by as many zero bytes (i.e., 00_{16} as required to arrive at a multiple of 16

- Note that if $|P|$ is already a multiple of 16, then a full block of padded data is appended: $80_{16} \underbrace{00_{16}00_{16} \ldots 00_{16}}_{15}$
- It is easy to see that the padding is reversible, just because padding is always inserted

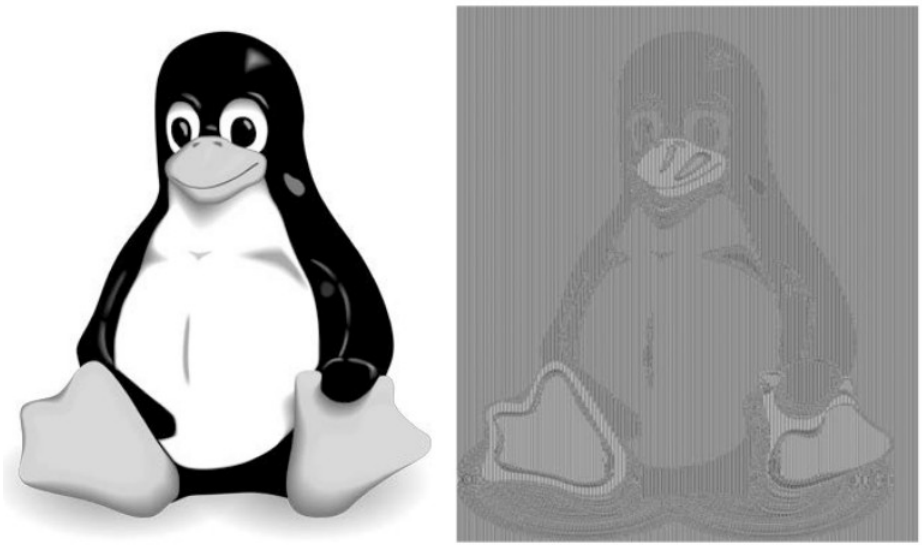
Some invertible padding scheme (cont.d)

- Another reversible padding scheme appends different bytes depending on the amount of padding
  - If $|P| = 16k$, for some $k > 0$, append $\underbrace{10_{16}10_{16}\ldots10_{16}}_{16}$ to P
  - If $|P| = 16k + 1$, $k \geq 0$, then append $\underbrace{0f_{16}0f_{16}\ldots0f_{16}}_{15}$
  - If $|P| = 16k + 2$, $k \geq 0$, then append $\underbrace{0e_{16}0e_{16}\ldots0e_{16}}_{14}$
  - ...
  - If $|P| = 16k + 15$, $k \geq 0$, then append 01_{16}

The ECB mode

- The simplest (but insecure) mode of operation is the *Electronic Codebook Mode* (*ECB*)
- Under ECB each block of the plaintext is encrypted separately, using the same key
- Formally, let $P = P_1||P_2||\ldots||P_n$ be the (already padded) plaintext, and let **E** denote the block cipher's encryption function
- Then: $C_1 = \mathbf{E}(K, P_1), C_2 = \mathbf{E}(K, P_2), \ldots, C_n = \mathbf{E}(K, P_n)$, where K is (obviously) the key
- The big problem with ECB is that identical plaintext blocks get encrypted to identical ciphertexts
- Like in the classic Vigenère cipher, if two identical parts of a message align to the start of block, then such blocks will be encrypted the same
- Encrypting picture or graphic files can be really problematic, as is well known

The Linux penguin encrypted with ECB



The Linux penguin: original image and the image encrypted under ECB mode

Source: J.P. Aumasson, Serious Cryptography, No Starch Press (2018)

The CBC mode

- There is more than one solution to the problems raised by ECB
- One possibility is somehow to “chain” the different blocks
- In the *Cipher Block Chaining* (*CBC*) this is done by encrypting $P_i \oplus C_{i-1}$ instead of P_i
- Since there is no previous encrypted text to use with P_1, for the first encryption a special *initialization vector* (*IV*) is used
- The whole scheme is then the following:

$$\begin{aligned} C_1 &= \mathbf{E}(K, P_1 \oplus IV) \\ C_i &= \mathbf{E}(K, P_i \oplus C_{i-1}), \quad i = 2, 3, \ldots \end{aligned}$$

- As for decryption:

$$\begin{aligned} P_1 &= \mathbf{D}(K, C_1) \oplus IV \\ P_i &= \mathbf{D}(K, C_i) \oplus C_{i-1}, \quad i = 2, 3, \ldots \end{aligned}$$

- The IV must clearly be inserted into the encrypted message

Possible choices for the IV

Fixed IV This is the simplest choice but one that introduces the same ECB problem for the first block. Often messages start with a common header ...

Random IV Since the very idea of CBC is to "randomize" each block of the plaintext with the ciphertext of the previous block, then why not to randomize also the first block?

Use a Nonce A *nonce* is a "number used once" (and then discarded). The protocol for using a nonce as an IV is described in the following slide

Protocol for using nonces

- Assign a unique number to each message. This is typically done by keeping a counter that starts at 0 and gets incremented by one at each new message (not using modular arithmetic)
- Prepare the the nonce as a block of data (i.e., a block as adopted by the block cipher being used). The nonce might in principle be just the message number, but in general other information is added that ensure its uniqueness with the particular key
- Encrypt the nonce using the block cipher to obtain the IV
- Encrypt the whole message using the generated IV (and the CBC mode)
- Add information to the ciphertext that allows the receiver to reconstruct to nonce (i.e., the message number)
- Note that the IV is not inserted into the encrypted message

The CTR mode

- *Counter mode* (*CTR*) is quite different from other modes since it does not use the block cipher to encrypt the blocks of the message
- The block cipher is used to produce different block keys that are xor-ed with the blocks of the message
- More precisely, CTR mode uses a counter i and a nonce n.
- The counter start at 1 and gets incremented for each message block. It must be unique within the message
- The nonce must be the same for blocks of the same message but different messages must use different nonces
- The protocol is the following:

$$\begin{aligned} K_i &= \mathbf{E}(K, n||i), \qquad i = 1, 2, \ldots \\ C_i &= P_i \oplus K_i \end{aligned}$$

Algoritmi di Crittografia

Doubling DES (as any other "weak" cipher)

- To strenghten a block cipher, one possible (and reasonable) idea is to perform double (or triple) encryption
- That is, compute $\mathbf{E}'(K_1, K_2, P) = \mathbf{E}(K_2, \mathbf{E}(K_1, P))$, where $\mathbf{E}$ is the encryption function of the cipher at hand, with decryption done in reverse order
- This sounds a good idea, at first, since it doubles the secret bits used, which is crucial in case of ciphers like DES
- However, there is a preliminary condition that must be satisfied, for the whole construction to make sense, namely that the block cipher is not *idempotent*
- What does that mean? Simply that the composition of the two permutations corresponding to K_1 and K_2 (recall our "ideal" model of a block cipher) does not turn out to be the permutation corresponding to a third possible key K_3

Doubling DES (as any other "weak" cipher)

- Under our model, it is very unlikely that this happens, fortunately
- The argument runs as follows, where (for definiteness) we use key and block length of DES
- There are 2^{56} possible keys, hence there are as many different permutations computable by DES (precisely those that correspond to a key)
- However, the number of different permutations is incredibly larger, namely $2^{64}!$
- But then the probability that the composition of two permutations computable by DES is still a permutation computable by DES is of the order of $\frac{2^{56}}{2^{64}!}$ which is negligible

A trivial example

- Suppose the block size is 3 and suppose we have 1-bit keys (hence just 2 different keys...)
- Say the keys 0 and 1 correspond to the linear permutations described by:

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

- If we compose the above permutations in all the (four) possible ways, we obtain four different permutations that can be indexed by 2-bit keys

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix} \text{ and } \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$$

Meet-in-middle attack

- So at least duobling DES might have sense
- There is another serious problem, though, and this is the *meet-in-the-middle* attack
- Meet in the middle means precisely "look for possible intermediate values" generated by the encryption process
- The strategy, for a CPA-capable attacker, is the following:
  - Choose a plaintext P and obtain the corresponding ciphertext C
  - For each possible key K_1, compute $\mathbf{E}(K_1, P)$ (note here you don't need CPA capabilities; you can do it your own)
  - Store the results in a lookup table with the $\mathbf{E}(K_1, P)'s$ as keys and the $K_1's$ as values
  - For each possible value of K_2 compute $\mathbf{D}(K_2, C)$ and look if the value appears in the table, corresponding to some key K_1
  - In the affirmative case you have found a pair such that $\mathbf{E}'(K_1, K_2, P) = C$.

Meet-in-middle attack (cont.d)

- Note, however, that K_1 and K_2 need not be the right keys!
- Consider the two pairs of permutations:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 4 & 1 & 3 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 3 & 4 & 1 & 2 \end{pmatrix}$$

and

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 5 & 2 & 3 & 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 5 & 3 & 4 \end{pmatrix}$$

- The permutations are all different, but the composition of the first pair and of the second pair is the same, namely

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 1 & 5 & 2 \end{pmatrix}$$

Meet-in-middle attack (cont.d)

- This “fix” is quite easy, though
- Ask for some other pairs plaintext/ciphertext and compare them with the pairs you can compute yourself with your candidate keys
- The question is: how many other pairs are enough to be (reasonably) sure to have picked the right keys?
- Here’s the figures for DES: there are 2^{112} different pairs of keys and there are “only” 2^{64} ciphertexts
- Under the (reasonable) assumption that plaintexts are mapped to ciphertexts uniformly, there must be $2^{112}/2^{64} = 2^{48}$ pairs of keys under which the first chosen plaintext P is mapped precisely to C

Meet-in-middle attack (cont.d)

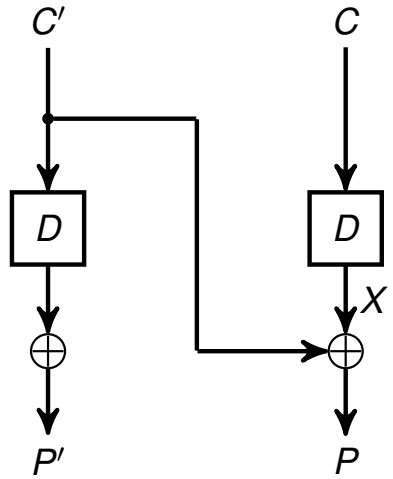
- We now pick another plaintext P' and obtain the corresponding ciphertext C', then we encrypt P' using the candidate keys
- If the candidate keys are good, we clearly obtain the same result
- Otherwise, given our uniformity assumption, for any false pair we have only one chance over 2^{64} (i.e., the number of different ciphertexts) of computing the same value C'
- Thus, with just one "check" we have a probability $2^{48}/2^{64} = 2^{-16}$ of being fooled
- As an exercise, argument that with an additional check the probability drops to 2^{-80} (and not to 2^{-32})

What is the cost of the m-i-t-m attack?

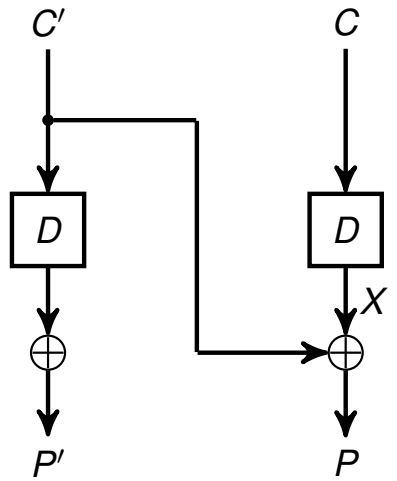
- While the obvious goal of doubling the encryption is to bring the security to $2|K|$ bits (112 for DES), the achieved results is much weaker
- Building the table has a cost proportional to $2^{|K|}$ (actually, a bit more, since the table must be kept sorted)
- Also, decrypting C with all possible keys has cost proportional to $2^{|K|}$
- There is also the considerable cost in terms of storage: a table with $2^{|K|}$ entries of $b \cdot |K|$ bits (here b is the block size in bits). In case of DES this mean $15 \cdot 2^{56} \approx 10^{18}$ bytes
- However, the point is that the security of "double DES" is essentially that of DES
- The same attack proves that the security of 3DES is 112 bits, rather than the theoretical 168 bits

Padding oracle attack

- Works for any block cipher in CBC mode
- Requires that the attacker be “informed” about whether the plaintext decrypted has well-formed padding (say, by the remote server throwing some exception, thus acting as an “oracle”)
- Hence, to counter this attack, padding errors must be trapped
- How does padding oracle attack work?
- Let us schematize the last two rounds of a CBC decryption

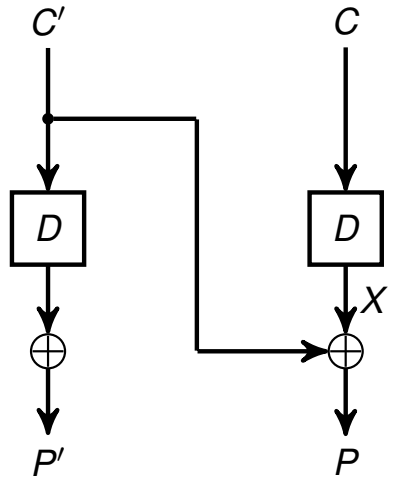


Padding oracle attack (cont.d)



- If we randomly change C', with high probability the decrypted plaintext will be malformed and he oracle will throw an exception
- Our strategy is thus to pick $C'[0:14]$ at random (note, differently from Python, here we assume that in the slice notation $x:y$ the last element is included)
- As for $C'[15]$, we initially set it to 00_{16} and increment by 1 (base 16) in each subsequent query

Padding oracle attack (cont.d)



- Suppose for a given value $C'[15] = xy_{16}$ the oracle does not throw the exception
- In this case, we can be reasonably certain that the last byte of the decrypted cypertext is 01_{16}, which in turn means that $X[15] = xy_{16} \oplus 01_{16}$
- Exercise: show that the probability of falsely assuming 01_{16} is bounded by $\sum_{i=1}^{15} \frac{1}{2^{8i}}$

Padding oracle attack (cont.d)

- Once we have determined $X[15]$, we set $C'[15] = X[15] + 02_{16}$ and repeat the strategy: i.e., we fix $C'[0:13]$ to random bytes and let $C'[14]$ vary
- By the same arguments, when the oracle accepts, we can be almost sure that the second last byte of the decryped cyphertext is 02_{16} and so we can recover $X[14]$.
- Proceeding in this way we can completely recover X
- The expected number of queries is approximately 128×16, i.e., slightly more than 2000
- Having recovered X and known the "true" C' we can decrypt P
- This process can be reapeated for all the blocks except for the first one, unless we also know the IV