Symmetric Key Cryptography

Gianluigi Me
me@disp.uniroma2.it
Anno Accademico 2008/09

Outline

- Feistel networks
- Keeloq algorithm (RFID)
  - Attacks on Keeloq
- Data Encryption Standard
  - Attacks on DES
- Advanced Encryption Standard
- RC4
  - RC4 attacks on WEP

Design principles

For software ciphers, the basic principle of diffusion and confusion is supplemented with the following mechanisms:

- The dependence of S-boxes on the secret key
- An increase in the size of S-boxes
- An increase in the size of the transformed data block, up to 512 bytes
- The dependence of the encryption algorithm on the secret key (for example, adjusting the transformation operations by the secret key)
- Data-dependence of the transformation operations (i.e., controlling the selection of a modification of the current operation depending on the input message)
- Data-dependence of the subkey selection
- Various combinations of the mechanisms listed in the first five items (for example, specifying the dependence of the subsets of possible modifications of the controlled operation on the secret key; in other words, a combination of the third and fourth items)

Feistel Networks

- Most block encryption algorithms use this general structure, due to Horst Feistel (1973)
- Inputs: Plaintext (halved) , Key, Round function F
- Uses *n* rounds, in each
  - Inputs: L_i and R_i
  - $L_{i+1} = R_i$
  - $R_{i+1} = L_i \oplus F(R_i,K_i)$
  - F( ) is a function that selects certain bits, duplicates some, and permutes them. K_i is derived from K
- Final ciphertext is combination of L_n and R_n
- At IBM, Feistel built *Lucifer*, the first such system

Feistel networks

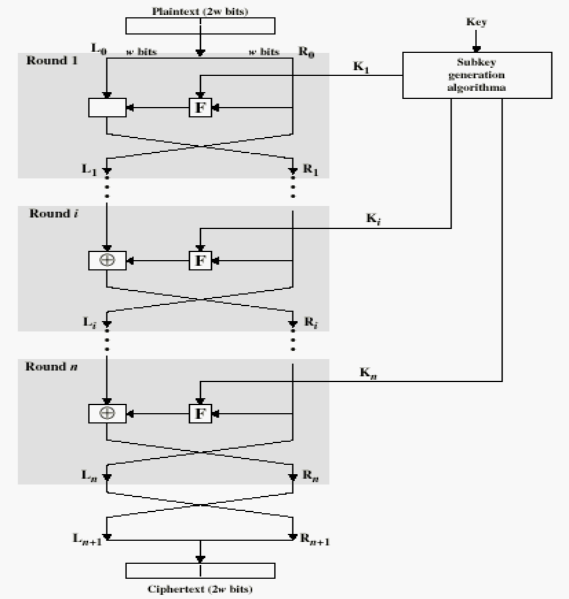


Figure 2.2 Classical Feistel Network

Feistel networks (definitions)

- *The F-function of a conventional Feistel network can be expressed as:*

$$F : \{0,1\}^{n/2} \times \{0,1\}^{k} \rightarrow \{0,1\}^{n/2}$$

- *One round of a conventional Feistel network (also called a balanced Feistel network) is:*

$$X_{i+1} = (F_{k_i}(msb_{n/2}(X_i)) \oplus lsb_{n/2}(X_i))||msb_{n/2}(X_i)$$

- *A balanced Feistel network consists of j rounds, where*

$$X_{i+1} = (F_{k_i}(msb_{n/2}(X_i)) \oplus lsb_{n/2}(X_i))||msb_{n/2}(X_i)$$

Feistel networks (definitions)

- *An Unbalanced Feistel Network (UFN) is a Feistel network where the "left half" and the "right half" are not of equal size.*
- *One round of a s-on-t, or s:t, UFN is*

$$X_{i+1} = (F(msb_s(X_i), k_i) \oplus lsb_t(X_i))||msb_s(X_i)$$

- *The msbs(Xi) is called the **source block**. The lsbt(Xi) is called the **target block**.*
- *UFNs where s > t are called source heavy, and UFNs where s < t are called target heavy. A sub-block, b, is the gcd of s, t, and n. The F-function is a collection of 2^k mappings of the s-bit numbers onto the t-bit numbers, or s/b sub-blocks onto t/b sub-blocks.*

Feistel networks (definitions)

- *A UFN is homogenous when the F-function function is identical in each round, except for the round keys. A UFN is heterogeneous when the F-function for different rounds is not always identical except for the round keys.*
- *A UFN is complete when s + t = n, that is, when in each round every bit in the block is either part of the source block or part of the target block.*

Feistel networks (definitions)

- *A UFN is incomplete when s + t < n. In an incomplete UFN, the n - s - t = z bits that are not part of the source block or part of the target block are called the null block.*
- *A UFN is consistent when s, t, z, and n remain constant for the entire cipher. A UFN in which the sizes of the source and target block change during encryption is called inconsistent.*

Notes on Feistel Cipher Structure

- Process is reversible
  - $R_{i-1} = L_i$
  - $L_{i-1} = R_i \oplus F(R_{i-1},K_{i-1})$
  - Same algorithm can be used but with keys reversed
- Security Considerations
  - Larger block size means fewer blocks and greater security
  - Larger key size means greater security
  - More rounds considered to offer better security (?)
  - Greater complexity of subkey generation may help security
  - Greater complexity of round function may increase security

Block Cipher Design Issues

- Easy to design a secure block cipher
  - By increasing the complexity of F (e.g., more complex S-boxes)
  - By iterating 1000 rounds
- Goals
  - Fast – few rounds, use simple operations
    - Low communication overheads
    - Low battery consumption in hand-helds
  - Easy to implement in hardware
    - Simple, ubiquitous operations
  - Efficient in memory usage
    - Can run on a smart card
  - Does not require too much secret material (keys, boxes)
    - Sometimes put on expensive tamper-proof memory

Data Encryption Standard (DES)

- Without a standard, software and hardware cannot interoperate, or at least it is very expensive
- In 1973, National Institute for Standards and Technology (NIST) issued RFP for Data Encryption Algorithm (DEA)
  - provide high level of security
  - completely specified and easy to understand
  - the security must reside in the key
  - available to all users
  - adaptable to diverse applications
  - economically implementable in hardware
  - efficient to use
  - validated
  - exportable

Data Encryption Standard (DES)

- NIST (NBS) issued a Request For Proposal (RFC)
- IBM had only serious proposal
  - Patented and based on Lucifer (Feistel et al)
- NIST issued a Request For Comments (RFC)
  - Quite a few were concerned about NSA backdoor
  - NSA reduced the key size from 112 to 56 bits
    - Diffie and Hellman presented a $20MM 1-day DES cracking machine
  - NSA had also changed the original S-boxes design
    - There were some claims of linearity in the new design
- DES was adopted in 1977
- In 1987, under NSA pressure, DES almost not recertified
- Until 1994, only hardware implementations of DES were permitted

Data Encryption Standard (DES)

- A Feistel block cipher structure
  - 64-bit blocks
  - 56-bit keys
  - 16 rounds
  - Adds initial and final permutation of the text (irrelevant to security)
  - Key shifted circularly for next round, and 48 bits are selected for K_i

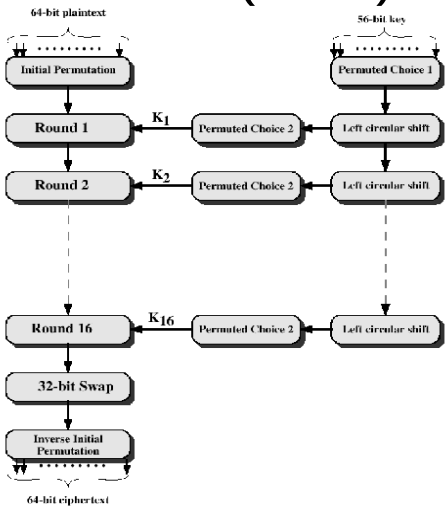


Figure 2.3 General Depiction of DES Encryption Algorithm

One Round of DES

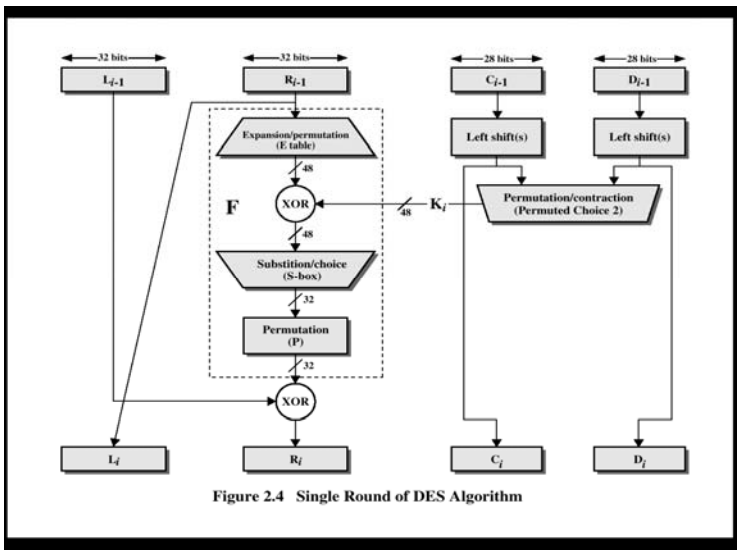


Figure 2.4 Single Round of DES Algorithm

One Round of DES

- Key Transformation
  - Each key-half is shifted 1 or 2 bits in each round (per given table)
  - The 56 key bits are permuted and 48 bits are chosen (per table)
- Text transformations
  - Expansion of R_i from 32 to 48 bits (size of key)
    - Avalanche effect – some bits are duplicated
  - 48 bits are XORed with K_i
  - Substitution, using 8 S-Boxes with 6-bit input and 4-bit output
    - S-boxes are well chosen to introduce non-linearity

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

  - 32 bits are permuted according to specified P-Box
  - 32 bits are XORed with L_i to create R_{i+1}
- There were claims that S-boxes weakened by NSA

DES

- Initial software implementations were slow
  - On IBM Mainframe 32,000 blocks / second
  - 125 MHz about 200,000 blocks per second (about 1.6 MByte/s)
- Hardware implementations are very fast
  - VLSI Technology 6868 (“Gatekeeper”) DESes in 8 clock cycles
  - DEC built GaAs gate array that DESes 16.8 million blocks/s
  - Many DES VLSI chips: e.g., 32MHz with data rate of 200 MByte/s
- Weak keys
  - All 0’s, or all 1’s in each half would result in same subkeys
  - Note: if K’ = complement of K, then $E_{k'}(P')$ = complement of $E_k(P)$

Example

Attacks to DES

Brute force:

- Attacker must try all possible keys (2^{56}) to match given plain-ciphertext pair.
- How expensive (time and money)?
- In 1977, Diffie & Hellman claimed could build machine for $ 20,000,000 to recover DES key in 10 hours
- Extrapolated that in 1987 could build a similar machine for $ 200,000
- In 1996 Wiener designed $ 10.50 DES chip with 5,000,000 encryption/sec. Used 5760 chips in parallel (~$ 60,000 worth of chips) to recover DES key in 1.5 days
- Claimed could break key in 3 hours with $ 1,000,000 machine

Data Encryption Standard (DES)

Differential Cryptanalysis: Eli Biham & Adi Shamir 1990

- A chosen-plaintext attack that uses two plaintexts with specific difference. Then, based on the difference in the ciphertext (and also internal rounds), one can update the a priori probability of keys
- Best attack against full 16-round DES requires 2^{47} chosen plaintexts
- I.e., attacker must be able to choose 2^{47} different plaintexts and submit them to DES!
- Largely unpractical: to get requisite data, need to feed 1.5 Mbits/s data stream of chosen plaintext for almost 3 years to DES under attack
- Converting chosen plaintext to known plaintext attack requires 2^{55} plaintext blocks (similar to brute force)
- S-boxes (rounds) designed so as to make DES amazingly resilient to this attack

Data Encryption Standard (DES)

Linear Cryptanalysis attack Mitsuru Matsui 1993

- Use linear approximations to describe the action of a block cipher
- Certain XORs of plaintext and ciphertext bits will result in a certain XOR of key bits with some probability $p \neq 1/2$
- Heavily dependent on structure of S-Boxes
- S-Boxes not optimized against this type of attack
- Against full 16-round DES, can recover key with average of 2^{43} known plaintexts
- In 1994 a software implementation using 12 HP9735 workstations recovered a DES key in 50 days

Back to Brute Force

- DES Challenge I (1997) 96 days
- DES Challenge II (1998) 56 hours with special hardware (64 x 28 dedicated chips)
- DES Challenge III (1999) 22 hours with 100,000 PCs on Internet
- Time to break DES keys decreases as CPU speed increases
- (approx. every 18 months CPU speed doubles up)
- About 15 doublings singe 1977
- To stay at same level of security as 1977, DES needs ~71-bit keys
- Number of bits should increases as CPU speed goes up!

Triple DES (3DES)

- 3DES uses 3 56-bit keys

$C = E_{k1}(D_{k2}(E_{k1}(P)))$

$P = D_{k1}(E_{k2}(D_{k1}(P)))$

- Effective as ~ 90-bit key
- Note: if K1 = K2 then 3DES = DES
- 2DES doesn't work
  - meet-in-the-middle attack requires only 2^{56}

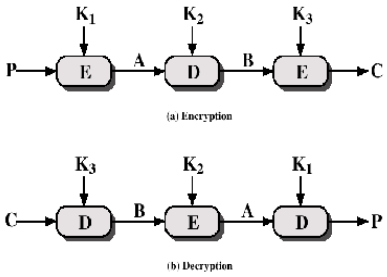


Figure 2.6 Triple DEA

Keeloq cipher

- KeeLoq is a block cipher with a 64-bit key which operates on 32-bit words
- Its design is based on a nonlinear feedback shift register (NLFSR) of length 32 bits with a nonlinear feedback function of 5 variables. The feedback depends linearly on two other register bits and on the next key bit taken from the rotated key register of length 64 bits.

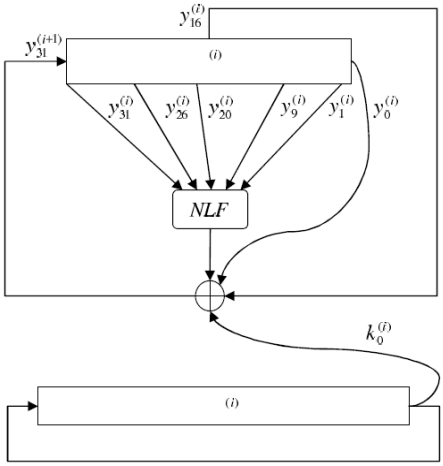
Keeloq cipher

•Let $V_n = GF(2)^n$ be the set of all n-bit words and
•$Y(i) = (y(i)_{31}, \ldots, y(i)_0) \in V_{32}$, $y(i)_j \in GF(2)$, describe the state of the text register in cycle i for j = 0, . . . , 31 and i = 0, 1, . . .
•Let also $K(i) = (k(i)_{63}, \ldots, k(i)_0) \in V_{64}$, $k(i)_j \in GF(2)$, denote the state of the key register in cycle i for j = 0, . . . , 63 and i = 0, 1, . . .
•Then each cycle of encryption can be described using the following algorithm

$$\textit{Compute the feedback bit: } \varphi = NLF(y_{31}^{(i)}, y_{26}^{(i)}, y_{20}^{(i)}, y_9^{(i)}, y_1^{(i)}) \oplus y_{16}^{(i)} \oplus y_0^{(i)} \oplus k_0^{(i)}$$
$$\textit{Rotate text and insert feedback: } R^{(i+1)} = (\varphi, y_{31}^{(i)}, \ldots, y_1^{(i)})$$
$$\textit{Rotate key: } K^{(i+1)} = (k_0^{(i)}, k_{63}^{(i)}, \ldots, k_1^{(i)}).$$

Keeloq cipher

Keeloq cipher (encryption)



Keeloq cipher

The NLF is a boolean function of 5 variables and is of degree 3. In the specification the NLF is assigned using a table. The vector of outputs is 0x3A5C742E. This corresponds to the following Algebraic Normal Form

$$NLF(x_4, x_3, x_2, x_1, x_0) = x_0 \oplus x_1 \oplus x_0x_1 \oplus x_1x_2 \oplus x_2x_3 \oplus x_0x_4 \oplus x_0x_3 \oplus x_2x_4 \oplus x_0x_1x_4 \oplus x_0x_2x_4 \oplus x_1x_3x_4 \oplus x_2x_3x_4.$$

Keeloq cipher

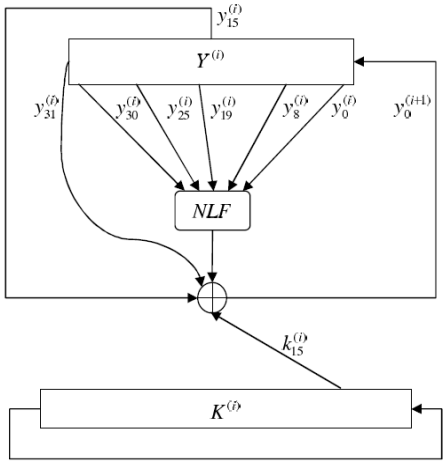
For encryption:
• the key register is filled with the 64 key bits K = (k_{63}, . . . k_0) $\in V_{64}$, $k_j \in GF(2)$, j = 0, . . . , 63, in the straightforward way: K(0) = K.
•If X = (x_{31}, . . . , x_0) $\in V_{32}$, $x_j \in GF(2)$, j = 0, . . . , 31, is a block of plaintext, the initial state of the text register is $Y^{(0)} = (x_{31}, \ldots, x_0)$.
•The output of the algorithm is the ciphertext Z = (z_{31}, . . . , z_0) = $Y^{(528)} \in V_{32}$, $z_j \in GF(2)$, j = 0, . . . , 31.

Keeloq cipher

For decryption:
•The key register is filled in the same way: K(0) = K =(k_{63}, . . . k_0) $\in V_{64}$.
•The decryption procedure complements the encryption.
•One decryption cycle can be defined by the following sequence of operations

$$\textit{Compute the feedback bit: } \varphi = NLF(y_{30}^{(i)}, y_{25}^{(i)}, y_{19}^{(i)}, y_{8}^{(i)}, y_{0}^{(i)}) \oplus y_{15}^{(i)} \oplus y_{31}^{(i)} \oplus k_{15}^{(i)}$$
$$\textit{Rotate text and insert feedback: } R^{(i+1)} = (y_{30}^{(i)}, \ldots, y_{0}^{(i)}, \varphi)$$
$$\textit{Rotate key: } K^{(i+1)} = (k_{62}^{(i)}, \ldots, k_{0}^{(i)}, k_{63}^{(i)}).$$

Keeloq cipher (decryption)



Keeloq attacks

For decryption:

- The key register is filled in the same way: K(0) = K =(k_{63}, . . . k_0) ∈ V_{64}.
- The decryption procedure complements the encryption.
- One decryption cycle can be defined by the following sequence of operations

Compute the feedback bit: $\varphi = NLF(y_{30}^{(i)}, y_{25}^{(i)}, y_{19}^{(i)}, y_{8}^{(i)}, y_{0}^{(i)}) \oplus y_{15}^{(i)} \oplus y_{31}^{(i)} \oplus k_{15}^{(i)}$

Rotate text and insert feedback: $R^{(i+1)} = (y_{30}^{(i)}, \ldots, y_{0}^{(i)}, \varphi)$

Rotate key: $K^{(i+1)} = (k_{62}^{(i)}, \ldots, k_{0}^{(i)}, k_{63}^{(i)})$.

RC5

- Invented by Ron Rivest (Ron's Code 5), and developed by RSA Technology into a number of their products
- A block cipher that uses only XORs, Additions, and Rotations
- Variable length blocks, keys, and number of rounds
- A, B are two halves of text; S_i are key-based
  - $A = ((A \oplus B) <<< B) + S_{2i}$
  - $B = ((A \oplus B) <<< A) + S_{2i+1}$
- With 16+ rounds, it resists differential attack
- Uses low-cycles operations, and is very fast

Other Block Ciphers

- Blowfish (Schneier)
  - Simple: additions, XORs, and table lookups
  - Table lookups may require large memory
  - Variable key length
- CAST
  - The round function differs from one round to next
- Int'l Data Encryption Alg (IDEA), Lai and Masey
  - Plaintext, key, and ciphertext are divided to 4 parts
  - Uses XORs, additions, and multiplications in 8 rounds
  - 128-bit key, 52 16-bit subkeys (can be independent)
  - Resists differential cryptanalysis
  - Used in PGP

AES Requirements

- Private key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- Stronger & faster than Triple-DES
- Active life of 20-30 years (+ archival use)
- Provide full specification & design details
- Both C & Java implementations
- NIST have released all submissions & unclassified analyses

AES Evaluation Criteria

- Initial criteria:
  - security – effort to practically cryptanalyze
  - cost – computational
  - algorithm & implementation characteristics
- Final criteria
  - general security
  - software & hardware implementation ease
  - implementation attacks
  - flexibility (in en/decrypt, keying, other factors)

Advanced Encryption Standard (AES)

- NIST put out the RFP in 1997
- Five finalists:

	MARS	RC6	Rijndael	Serpent	Twofish
General Security	3	2	2	3	3
Implementation of Security	1	1	3	3	2
Software Performance	2	2	3	1	1
Smart Card Performance	1	1	3	3	2
Hardware Performance	1	2	3	3	2
Design Features	2	1	2	1	3

- In October 2000, NIST recommended Rijndael

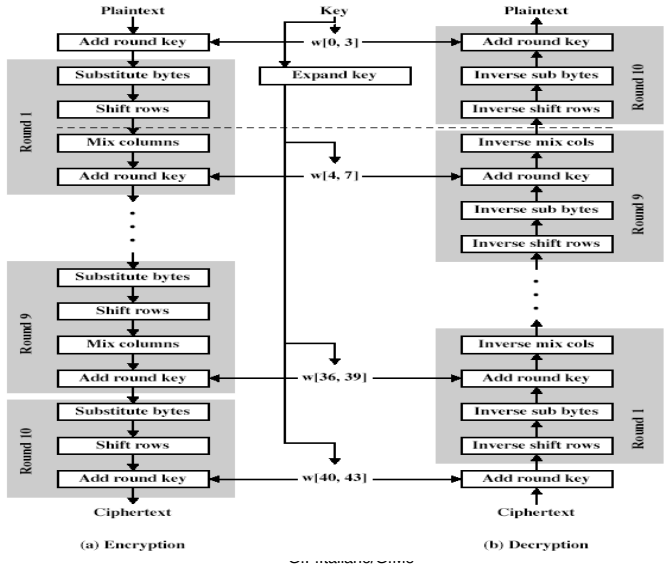
The AES Cipher - Rijndael

- Designed by Rijmen-Daemen in Belgium
- Has 128/192/256 bit keys, 128 bit data
- An **iterative** rather than F**eistel** cipher
  - treats data in 4 groups of 4 bytes
  - operates an entire block in every round
- Designed to be:
  - resistant against known attacks
  - speed and code compactness on many CPUs
  - design simplicity

Rijndael

- Processes data as 4 groups of 4 bytes (state)
- Has 9/11/13 rounds in which state undergoes:
  - byte substitution (1 S-box used on every byte)
  - shift rows (permute bytes between groups/columns)
  - mix columns (subs using matrix multiply of groups)
  - add round key (XOR state with key material)
- Initial XOR key material & incomplete last round
- All operations can be combined into XOR and table lookups - hence very fast & efficient

Rijndael



Byte Substitution

- A simple substitution of each byte
- Uses one table of 16x16 bytes containing a permutation of all 256 8-bit values
- Each byte of state is replaced by byte in row (left 4-bits) & column (right 4-bits)
  - e.g., byte {95} is replaced by row 9 col 5 byte
  - which is the value {2A}
- S-box is constructed using a defined transformation of the values in GF(2^8)
- Designed to be resistant to all known attacks

Shift Rows

- A circular byte shift in each row
  - 1st row is unchanged
  - 2nd row does 1 byte circular shift to left
  - 3rd row does 2 byte circular shift to left
  - 4th row does 3 byte circular shift to left
- Decrypt does shifts to right
- Since state is processed by columns, this step permutes bytes between the columns

Mix Columns

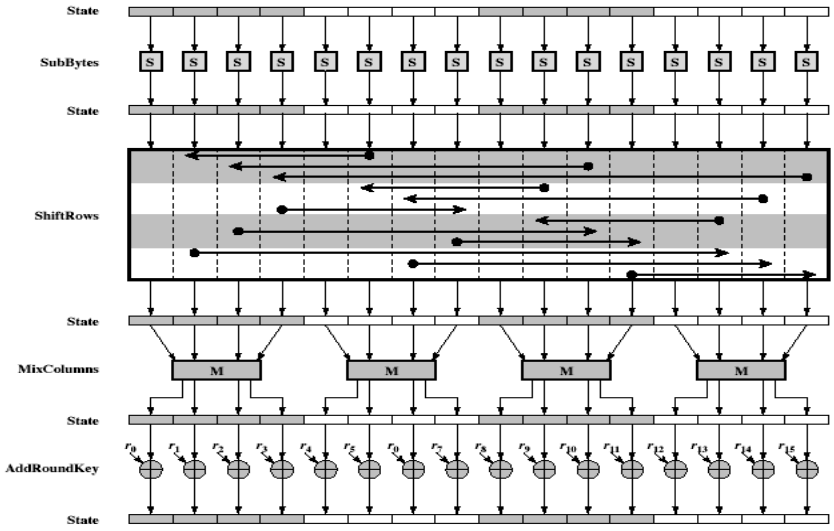
- Each column is processed separately
- Each byte is replaced by a value dependent on all 4 bytes in the column
- Effectively a matrix multiplication in GF(2^8) using prime poly m(x) =$x^8+x^4+x^3+x+1$

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Add Round Key

- XOR state with 128-bits of the round key
- Again processed by column (though effectively a series of byte operations)
- Inverse for decryption is identical since XOR is own inverse, just with correct round key
- Designed to be as simple as possible

AES Round



AES Key Expansion

- Takes 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words
- Start by copying key into first 4 words
- Then loop creating words that depend on values in previous & 4 places back
  - in 3 of 4 cases just XOR these together
  - every 4th has S-box + rotate + XOR constant of previous before XOR together
- Designed to resist known attacks

AES Decryption

- AES decryption is not identical to encryption since sequence of transformations for decryption differs
- But can define an equivalent inverse cipher with steps as for encryption
  - but using inverses of each step
  - with a different key schedule
- Works since result is unchanged when
  - swap byte substitution & shift rows
  - swap mix columns & add (tweaked) round key

Implementation Aspects

- Can efficiently implement on 8-bit CPU
  - byte substitution works on bytes using a table of 256 entries
  - shift rows is simple byte shifting
  - add round key works on byte XORs
  - mix columns requires matrix multiply in GF(2^8) which works on byte values, can be simplified to use a table lookup

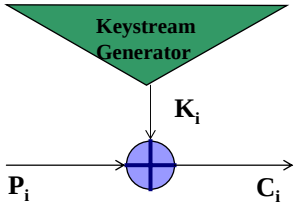
Rijndael Block Cipher

- By Belgians Joan Daemen, and Vincent Rijmen
- Basic operations use bit-coefficient polynomials, in GF(2^8)
- Does not use Feistel structure
- Instead uses 3 types of layers and a state
  - Non-linear layer, using optimized S-boxes
  - Linear mixing layer for diffusion of all bits throughout the rounds
  - Key addition layer, using a simple XOR
- Each round
  - Byte substitution (S-box from state matrix, with index (i,j) based on previous state)
  - Row shift (to the matrix of states)
  - Column mix (also to the matrix of states)
  - Key XOR with the current state

Stream Ciphers and Cipher Block Modes of Operation

Main sources: Network Security Essential / Stallings
Applied Cryptography / Schneier
Handbook of Applied Cryptography / Menezes, van Oorschot, Vanstone

Stream Ciphers



- Stream ciphers work on a stream of blocks (sometimes bits), altering the encryption from one block to the next
  - Keystream may change according to original key, previous encryptions, and block index
    - In synchronous stream ciphers, keystream does not depend on text
  - Other encryption parameters may also change, e.g., S-boxes

RC4

- Invented by Ron Rivest (Ron's Code 4)
- Variable-size stream cipher
- Has an 8 x 8 S-box: S_0, S_1, …, S_{255}
- Entries of S-box are permutation of [0,255]
- Permutation is function of variable-length key

RC4 (cont.)

- Initially, `i = j = 0`.
- Next, generate random byte R:

```
i = (i+1) mod 256;
j = (j+Si) mod 256;
swap (Si,Sj);
t = (Si + Sj) mod 256;
R = St;
```

- Keep in mind `S[S[1]+S[S[1]]]`
- First random byte depends on:

  `S[1], S[S[1]], S[S[1]+S[S[1]]]`

RC4 (cont.)

- Random byte R XORed with plaintext (ciphertext) to produce ciphertext (plaintext)
- Encryption with RC4 fast, i.e., ~10 times faster than DES

RC4 (cont.)

- Initializing S-box:
- First, $S_i = i$, for i = 0,1, … ,255.
- Next, fill a 256-byte array repeating key as necessary: $K_0, K_1, \ldots, K_{255}$:

```
j = 0;
for i = 0 to 255
   j = (j + Si + Ki) mod 256;
   swap(Si, Sj);
```


RC4 (cont.)

- Guarantees pseudo-random key (pad), which is what characterizes stream ciphers
- According to RSA, immune to differential and linear analysis and highly non linear
- Can be in $256! \times 256^2 \sim 2^{1700}$ possible states
- S-box slowly evolves with use:
  - i ensures every element changes,
  - j ensures elements change randomly
- Used in Lotus Notes, Apple's AOCE, Oracle Secure SQL, SSL,…
- Part of Cellular Digital Packet Data Specification, IEEE 802.11.

Summary of RC4

Proprietary algorithm belonging to RSADS Inc.

- Secret key stream cipher.
- Variable length key (up to 2048 bits).
- Fairly fast (1Mbyte/sec on 33MHz processor).
- Claimed to be very strong, sequence period greater than 10100.
- Exportable outside the U.S.
- Designed in 1987. Source code kept secret. Leaked onto the Internet in 1994…

Cipher Block Modes of Operation

- Stream ciphers can be implemented from block cipher building blocks
- Requirements:
  - Should be efficient, without significant overhead
  - Shouldn't allow chosen plaintext attacks to interfere with the encryption
  - Should be fault tolerant, not crashing in case of bit errors
- Note that the secrecy depends on the underlying cipher block algorithm

Electronic Codebook (ECB) Mode

- Simplest form
  - Each block (typically 64 bits) encrypted separately
  - As if there is a codebook of 2^{64} entries (per key)
- Fast, easy to parallelize
- Relatively fault tolerant
- Easy target to known-plaintext attack
  - cryptanalyst can rebuild the code book
  - Also susceptible to stereotypical beginning and ending of messages and statistical attacks
- Also easy target to modification attack
  - E.g., replacing the target-account block in a bank money wiring communication

Cipher Block Chaining (CBC) Mode

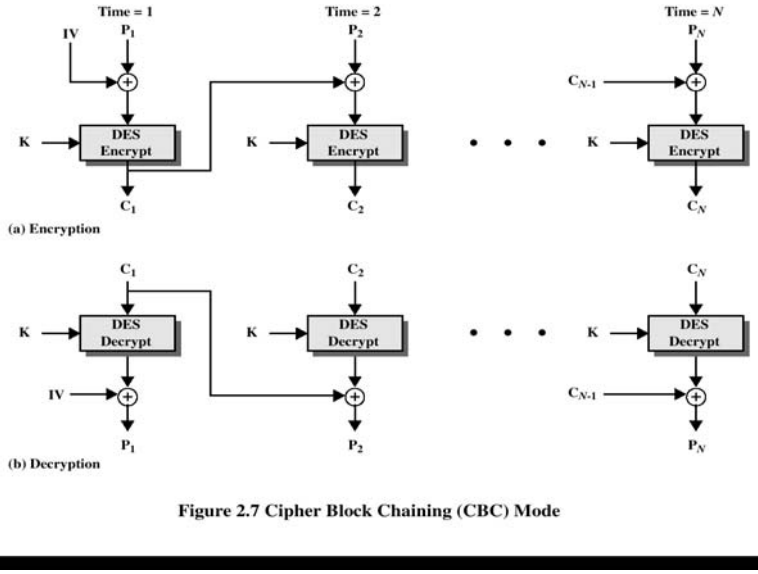
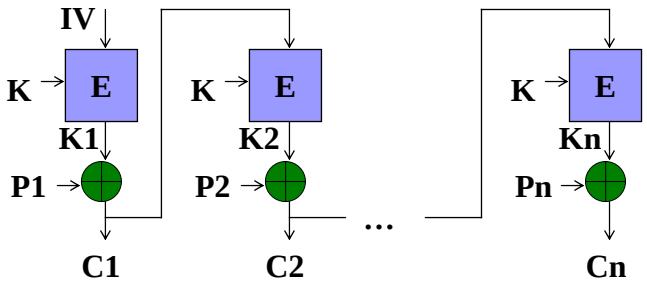


Figure 2.7 Cipher Block Chaining (CBC) Mode

Cipher Block Chaining (CBC) Mode

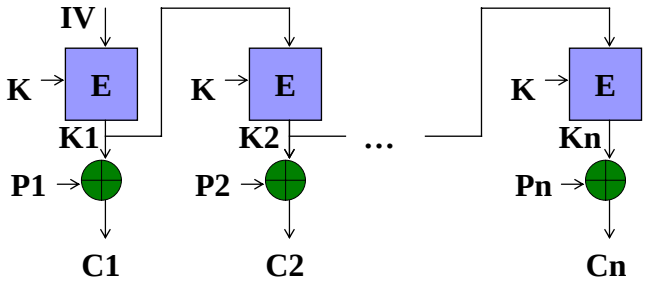
- Encryption
  - $C_i = E_k(P_i \oplus C_{i-1})$
  - $C_0 = IV$
- Decryption
  - $P_i = D_k (C_i) \oplus C_{i-1}$
- Initialization vector modifies encryption of identical blocks
  - Can be chosen by source and sent in the clear
  - Or, encrypt random data in the first block
- Errors
  - A bit of error in the plaintext will not extend the error
  - A bit of error in the ciphertext will garble that block, and will alter same bit in the next block, but then CBC self-recovers completely
- Security
  - A man-in-the-middle can easily append blocks in the end
  - Can change a bit, knowing which bit will be affected in 2nd block

Cipher Feedback Mode (CFB)



- Errors
  - A bit of error in plaintext affects all subsequent blocks but does not extend the error when decrypted
  - A bit of error in ciphertext affects same bit and next block, after which CFB self synchronizes

Output Feedback Mode (OFB)



- Output of Encryption serves as feedback

Key Management for Conventional Cryptography

Main sources: Network Security Essential / Stallings
Applied Cryptography / Schneier
Handbook of Applied Cryptography / Menezes, van Oorschot, Vanstone

Key Generation, Distribution and Management

- The security of any cryptographic system depends on safe and effective key distribution and management
  - frequent changes
  - low computational and communication overhead
- Key Distribution Centers (KDCs) are the single most critical point of failure, and are the toughest to implement
- Key Generation
  - Cryptanalyst may attack the key generation algorithm
- Distribution
  - Opponent may impersonate or attack the communication
- Management
  - Adversary may attack KDC systems, or simply exploit human weaknesses

Key Generation

- Key space should be large enough
- Selection from key space shall be random
  - Humans select poor keys prone to dictionary attack
  - Some algorithms have weak keys that should be avoided (DES has 16 such weak keys)
  - RC4 has weak keys too!
- ANSI X9.17 Key Generation Algorithm
  - Key is generated from previous key, through some encryption process that also takes into account a kept state information
  - Seeds generated from low-order bits of time stamps, time between keystrokes of administrator, etc.

Key Distribution Alternatives

- Physical Delivery
  - Alice can select the key and deliver to Bob
  - Charles, a trusted third-party, can select the key and deliver to both Alice and Bob
- Encrypted direct communication
  - From Alice to Bob using an earlier encrypted session
- Encrypted communication with trusted third-party
  - From Charles to both Alice and Bob

Key Distribution (cont.)

- Encryption location
  - Link encryption
  - End-to-end encryption

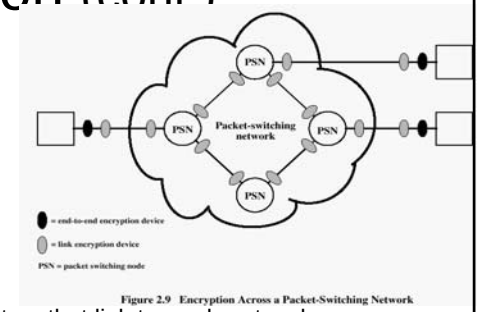


Figure 2.9 Encryption Across a Packet-Switching Network

- Physical delivery
  - best for link encryption, e.g., routers that link two sub-networks
  - hard for end-to-end, esp. ad-hoc / many-to-one communication
- Encrypted direct key-delivery communication
  - Dangerous: an attacker that gets one key, gets them all
- Conclusion:
  - Security of link communication should not be compromised, and shall use manual delivery of keying material (especially key-encryption keys)
  - End-to-end communication can use key-delivery by third party (data keys)

Session Key Distribution by KDC

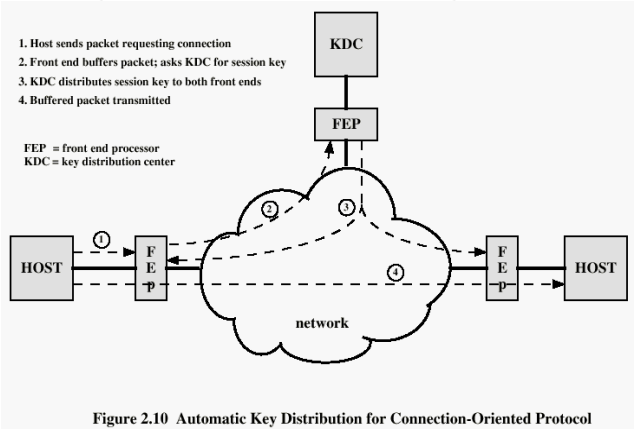


Figure 2.10 Automatic Key Distribution for Connection-Oriented Protocol

- It is safer if KDC-host link encrypted using a physically delivered key
- KDC-host communication shall also be mutually authenticated

Key Management Principles

- To reduce the risk of eavesdropping
  - use different keys for different purposes
  - generate new keys from old ones + hash function
- To reduce the risk of impersonation
  - use mutual authentication when exchanging keys
- To reduce the risk of computer/physical break-in
  - store most keys encrypted using master key
  - save master keys in your memory, smart card, flash key, etc.
  - use tamper-proof hardware encryption, much safer than software
  - destroy media on which keys were stored, even if were encrypted
- Replace keys frequently
- Report compromised keys to KDC with timestamp
- Backup keys shall be broken and spread

KEY MANAGEMENT STD NIST

- Recommendation for Key Management – Part 1: General (Revised)
- NIST Special Publication 800-57 **May, 2006**

Cryptoperiods

- A cryptoperiod is the time span during which a specific key is authorized for use by legitimate entities, or the keys for a given system will remain in effect. A suitably defined cryptoperiod:

1. limits the amount of information protected by a given key that is available for cryptanalysis,
2. limits the amount of exposure if a single key is compromised,
3. limits the use of a particular algorithm to its estimated effective lifetime,
4. limits the time available for attempts to penetrate physical, procedural, and logical access mechanisms that protect a key from unauthorized disclosure

Cryptoperiods

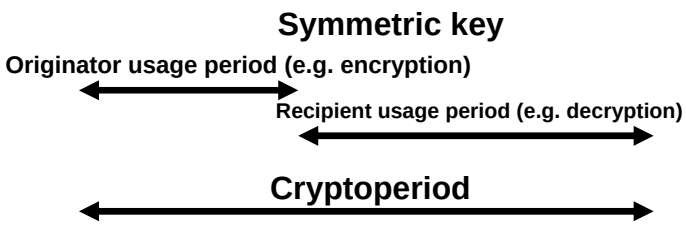
5. limits the period within which information may be compromised by inadvertent disclosure of keying material to unauthorized entities, and
6. limits the time available for computationally intensive cryptanalytic attacks (in applications where long-term key protection is not required).

- Among the factors affecting the risk of exposure are:

1. the strength of the cryptographic mechanisms (e.g., the algorithm, key length, block size, and mode of operation),
2. the embodiment of the mechanisms (e.g., FIPS 140-2 Level 4 implementation, or software implementation on a personal computer),
3. the operating environment (e.g., secure limited access facility, open office environment, or publicly accessible terminal),
4. the volume of information flow or the number of transactions,
5. the security life of the data,
6. the security function (e.g., data encryption, digital signature, key production or derivation, key protection),

Cryptoperiods

7. the re-keying method (e.g., keyboard entry, re-keying using a key loading device where humans have no direct access to key information, remote re-keying within a PKI),
8. the key update or key derivation process,
9. the number of nodes in a network that share a common key,
10. the number of copies of a key and the distribution of those copies, and
11. the threat to the information (e.g., who the information is protected from, and what are their perceived technical capabilities and financial resources to mount an attack).



3. Caso di studio: 802.11 WEP

Wireless

- Wireless 802.11 networks stanno diventando sempre più diffuse.
- Problemi con security di wireless LANs: WEP protocol
- Attacco su cattiva implementazione di crittografia!

Reti Wireless

- Originariamente pensate per servizi mobili stanno guadagnando popolarità grazie ai bassi costi ed alla facilità di realizzazione





- HomeRF – 1.2Mbps (recentemente elevato a 10Mbps)
- Bluetooth – short range (10m), Personal Area Network (PAN), bassisimo voltaggio
- 802.11 – IEEE Standard per le LAN wireless
  - "*Frequency hopping*", utilizza una frequenza a 2.4GHz ISM che non richiede licenza
- 802.11b (WiFi) – Direct Sequencing Spread Spectrum (DSSS), aumenta la velocità fino a 11Mbps
  - Si afferma sul mercato, in termini di costo, accettazione, interoperabilità
- 802.11a – prossimo passo, la velocità prevista è 54Mbps, dovrebbe risolvere i problemi di sicurezza presenti fino ad oggi.

Wireless Security

- Supporto wireless (a livello fisico e di link layer) fa nascere problemi di security.
- Security è difesa di tipo layered! Spesso, la sicurezza fisica fornisce la base.
- Wireless networks spesso "escono" fuori dalle nostre strutture fisiche (muri, recinzioni etc).
- E.g., ognuno può sedere fuori o entrare in un palazzo; ci sono wireless access points ovunque.

IEEE 802.11 Wired Equivalency Privacy (WEP)

- Protocollo scritto dal 802.11 working group.
- Wired Equivalent Privacy
- Obiettivi:
  - Confidentialità: non vogliamo che altri facciano packet sniffing…
  - Controllo degli accessi: non vogliamo che altri accedano ad Internet dalla nostra rete.
  - Integrità dei dati: non vogliamo che altri modifichino i nostri dati; e.g., inseriscano commandi sul nostro shell prompt.

WEP

- Obiettivo: protezione equivalente o migliore di wired networks.
- Usa chiave per autenticare ogni stazione. Una stazione deve possedere la chiave corrente per accedere ai servizi di rete.
- Anche access point richiedono una chiave per essere ammessi nella rete.
- Protocolli di autenticazione e di distribuzione delle chiavi lasciati interamente ai vendors.
- Cifratura dei dati station-to-station opzionale con RC4.

WEP

- 802.11 WEP basato su chiave segreta K condivisa tra utente e base station:
  - Tutte le base station hanno la stessa chiave.
  - Tutti gli utenti nella rete condividono la stessa chiave.
- Innanzitutto, utenti potrebbero rendere pubblica la chiave segreta.
  - Da dipendenti (insiders) arrivano gli attacchi più frequenti
  - Revocation? Ottenere una nuova chiave ed inserirla manualmente nel configuration panel…

WEP e keystream

- WEP usa chiave condivisa K per generare keystream
- Usa RC4 stream cipher.
- Stream cipher generano 1 bit della chiave alla volta e poi XOR con plaintext (1 bit alla volta).
- Producono ciphertext 1 bit alla volta.
- Key bits k_1, k_2, k_3, ... e plaintext bits p_1, p_2, p_3, ... $c_i = p_i \oplus k_i$

Richiami su RC4

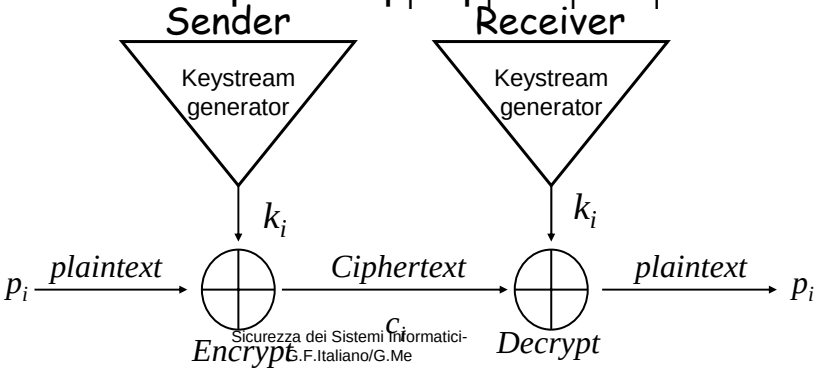
- RC4 usa 8-by-8 S-box con valori $S_0 ... S_{255}$
- Entries sono permutazioni di 0...255, e la permutazione corrente dipende dalla chiave K.
- Algoritmo ha 2 counter, i e j, inizializzati a 0
- Per generare random byte K:

  i = (i+1) mod 256
  j = (j+S_i) mod 256
  swap S_i and S_j
  t = (S_i + S_j) mod 256
  K = S_t

- Ciphertext = Plaintext ⊕ K

Keystreams

- Sender trasmette $c_i = p_i \oplus k_i$
- Receiver recupera plaintext con $p_i = c_i \oplus k_i$
- Funziona perché $p_i = p_i \oplus k_i \oplus k_i$



Basta la chiave?

- Stream cipher (come one time pad): **NON** riciclare mai lo stesso keystream per codificare due msg (differenti).
- Perché no?
- P1 e P2 codificati con stesso keystream KS:

  $C1 = P1 \oplus KS$ e $C2 = P2 \oplus KS$.
- Se due ciphertext codificati con la stessa chiave:

  $C1 \oplus C2 = P1 \oplus KS \oplus P2 \oplus KS = P1 \oplus P2$
- Con il plaintext di uno, si può recuperare l'altro.
- Altrimenti, molte tecniche (e.g., intuire IP header); problema più facile se si hanno n plaintext codificati con la stessa chiave.
- Se ciphertext è dell'attaccante, ha già un plaintext!

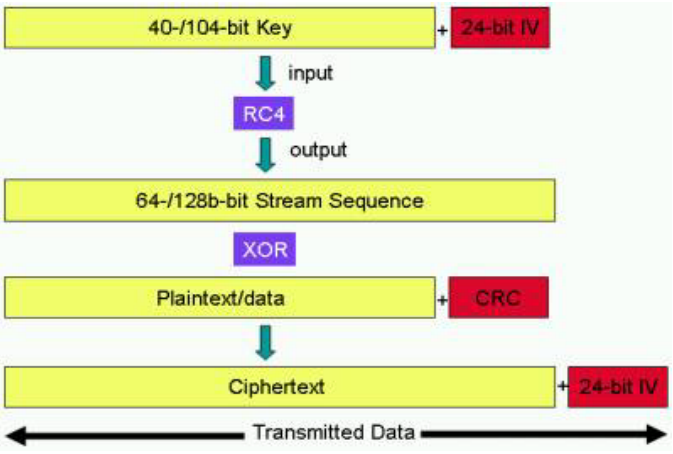
IV – Initialization Vector

- WEP non può usare la stessa chiave due volte!
- Per questo WEP usa IV per ogni pacchetto.
- IV cambia ad ogni pacchetto: se non ripetiamo IV non ripetiamo keystream.
- IV non deve essere riciclato, altrimenti si può recuperare plaintext!

IV (continua)

- IV deve essere cambiato e non ripetuto durante il tempo di vita della chiave.
- Quanto dura una chiave 802.11?
  - Tipicamente qualche anno…
- IV non deve essere riciclato, altrimenti si può recuperare plaintext!
- Molte card 802.11 resettano IV a zero ogni volta che si inserisce card. Poi IV viene semplicemente incrementato…

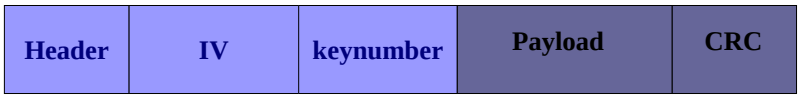
WEP



WEP: Operatività normale

- Sender & receiver (e.g., base station) condividono chiave K.
- Sender S vuole inviare msg M.
- S calcola checksum c(M) con CRC per produrre (M, c(M))
- S sceglie initialization vector (IV) v e genera keystream RC4(v,k) come descritto prima
- S trasmette (M,c(M)) ⊕ RC4(v,k) insieme a v.
- Receiver: prende v, produce keystream, e recupera plaintext con ⊕. Controlla checksum c(M).

WEP Packet Format



- WEP packet include
  - packet header,
  - initialization vector,
  - quale delle 4 chiavi è usata
  - encrypted payload
  - CRC checksum di encrypted payload.

Debolezze di IV

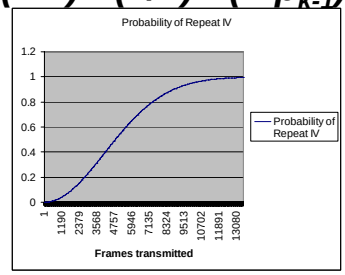
- Si può aspettare che si ripeta lo stesso IV nel tempo di vita di una chiave (i.e., anni!).
- Più banalmente: aspettare la mattina quando utenti inseriscono card 802.11 nel loro PC.
- IV: 24 bit nell'header del pacchetto 802.11.
- Ancora più banalmente: 2^{24} = 16 milioni di valori, e quindi 16 milioni di pacchetti prima di ripetere IV.
- Per un uso medio di 5Mbs e pacchetti di 1500 bytes, basta mezza giornata!

IV's (cont)

- Per un uso medio di 5Mbs e pacchetti di 1500 bytes, basta mezza giornata!
- Questo per una persona. Se più di una persona connessa, la probablità cresce notevolmente
- Paradosso del compleanno: su 23 persone, probabilità di averne due con stesso compleanno (mese, giorno) è circa 50%
- Anche con IV completamente casuale, probabile collisione dopo solo 5,000 pacchetti
- Aumenta al 99% dopo 12,430 frame…

Birthday paradox onWEP

- $p_k = p_{k-1} + (k-1) * (1/n) * (1-p_{k-1})$



- there are 2^{24} possible IVs (n) or about 16 million birthdays.
- So, a 50% chance of a repeated IV occurs after k = 4823 or 212 frames and reaches 99% after 12,430 frames.

Attacco a WEP: Collisioni su IV

- La chiave non cambia mai: basta aspettare che si ripeta lo stesso IV…
- Vademecum del wireless hacker:
  - Trova rete wireless senza sicurezza fisica (semplice)
  - Manda pacchetto da Internet verso una destinazione nella rete wireless. (Attaccante conosce plaintext!)
  - Memorizza IV utilizzato per tuo pacchetto e determina keystream usato. Dato che C1 = P1 ⊕ KS, allora
    C1 ⊕ P1 = P1 ⊕ KS ⊕ P1 = KS.
  - Crea database indicizzato da IV, memorizzando le KS associate: 1500 byte packets * 2^{24} = 24 GB
  - Aspetta che sia trasmesso un pacchetto con stesso IV e poi decripta.

Dictionary attack su WEP

- Database - “dictionary attack”.
- Molte implementazioni resettano IV a zero: il dizionario non occupa neanche troppo spazio!
- Nota 1: NON importa quanto sia grande la chiave RC4.
- Nota 2: NON recuperiamo la chiave della rete wireless, ma soltanto keystream con cui viene cifrato il traffico.

Fluher, Mantin, and Shamir (FMS)

- Collisioni su IV non è l’unico problema!
- Esiste attacco contro implementazione di RC4 che consente di recuperare la chiave.
- Attacco basato su assunzione che attaccante possa intuire primo byte del plaintext usato dalla vittima.
- Basato su RC4 weak key scheduling: si possono intuire chiavi dai primi pacchetti utilizzati.
- Ricordate `S[S[1]+S[S[1]]]` ?

FMS Attack - RC4 Algorithms

KSA(K)

Initialization:

For i = 0 … N - 1

S[i] = i

j = 0

Scrambling:

For i = 0 … N - 1

j = j + S[i] + K[i mod l]

Swap(S[i], S[j])

PRGA(K)

Initialization:

i = 0

j = 0

Generation Loop:

i = i + 1

j = j + S[i]

Swap(S[i], S[j])

Output Z = S[S[i] + S[j]]

FMS Attack - RC4 Algorithms

K = IV . SK

FMS Attack - RC4 Algorithms

→ KSA(K)

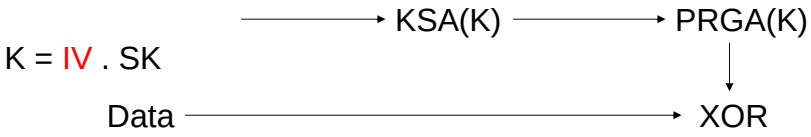
K = IV . SK

FMS Attack - RC4 Algorithms

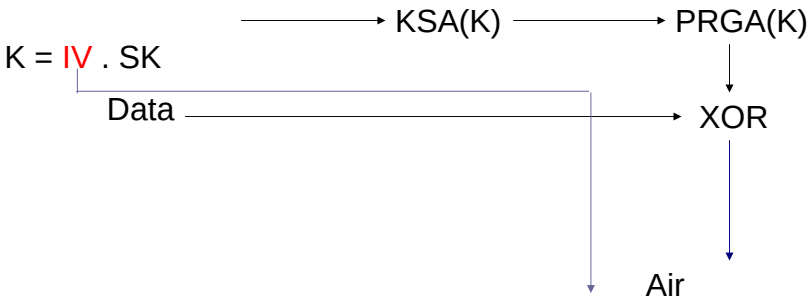
→ KSA(K) → PRGA(K)

K = IV . SK

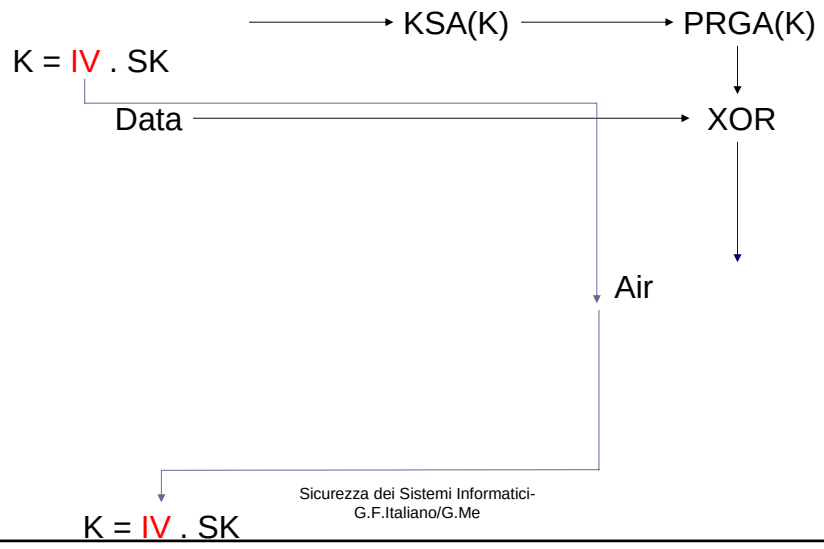
FMS Attack - RC4 Algorithms



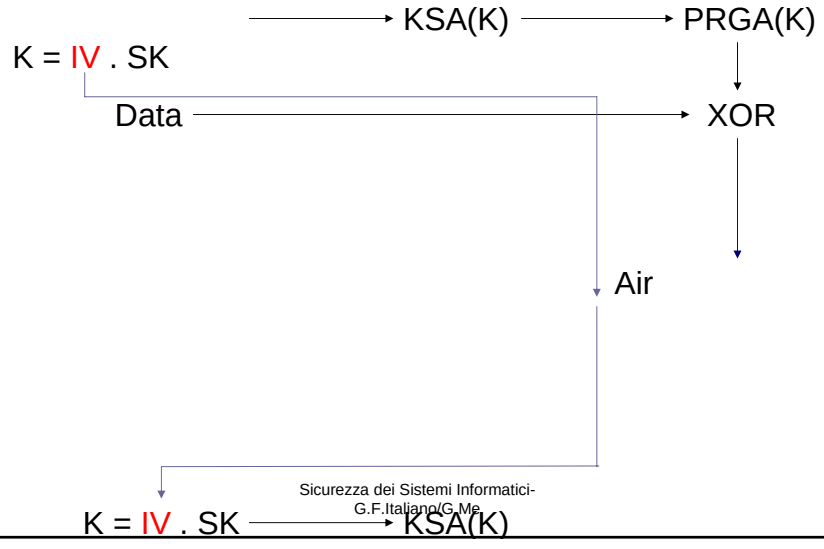
FMS Attack - RC4 Algorithms



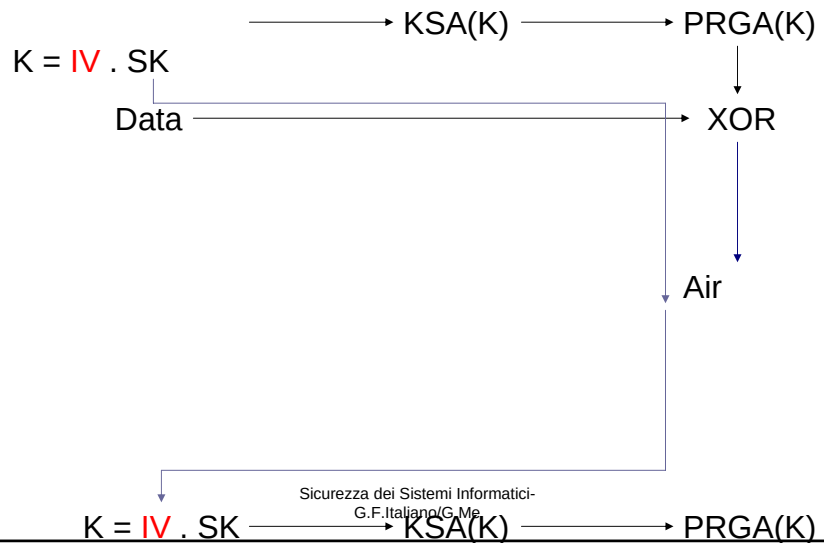
FMS Attack - RC4 Algorithms



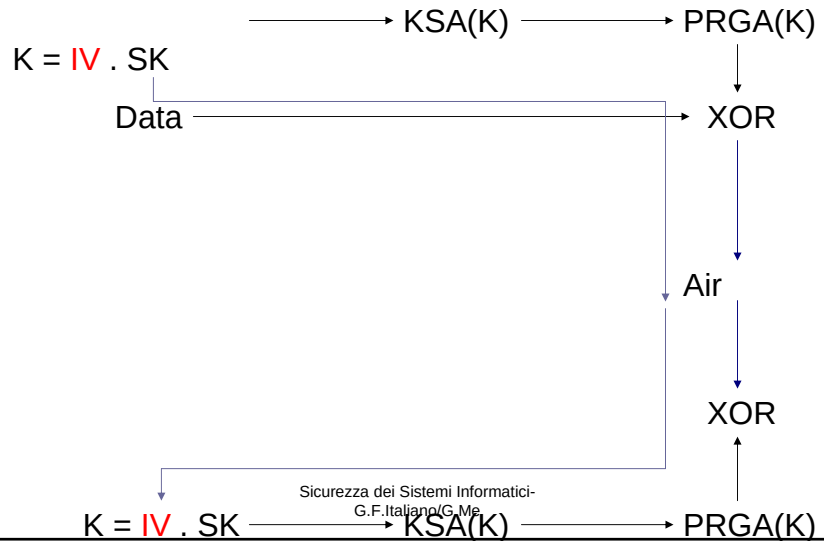
FMS Attack - RC4 Algorithms



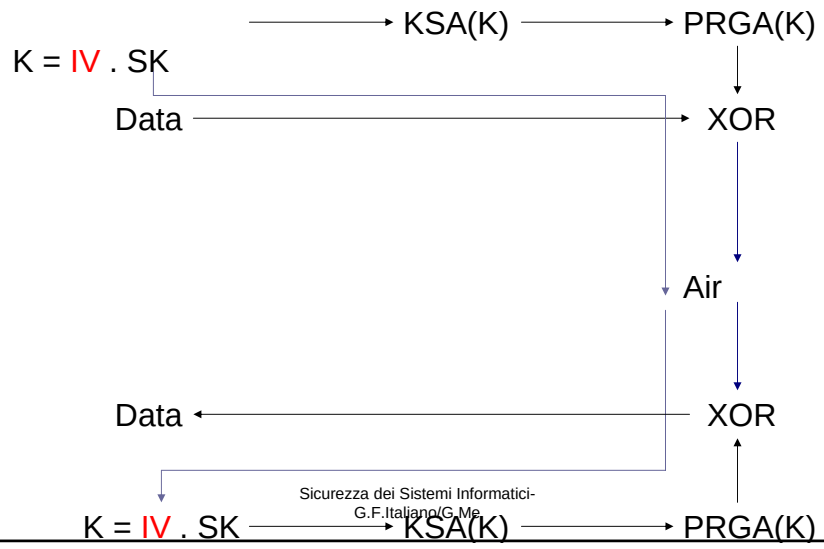
FMS Attack - RC4 Algorithms



FMS Attack - RC4 Algorithms



FMS Attack - RC4 Algorithms



How the FMS Attack Works

- The baseline attack is to look for IVs that conform to:

  (A + 3, N - 1, X)

  Where:
  A = The byte in SK you are cracking
  N = The size of the S-box (256)
  X = Any Number

- Although, it is recommended that you use the following equations right after the KSA:

  $X = S_{B+3}[1] < B+3$

  $X + S_{B+3}[X] = B+3$

- The main dilemma is that this equation is dependent on the previous key bytes, so it has to be applied to the entire sample set for every key byte that is tried

Which Means……

K = IV

Which Means……

K = IV → KSA(K)

Which Means……

K = IV → KSA(K) → $X = S_{B+3}[1] < B+3$, $X + S_{B+3}[X] = B+3$

Which Means……

K = IV → KSA(K) → $X = S_{B+3}[1] < B+3$, $X + S_{B+3}[X] = B+3$

Repeat 6,000,000-8,000,000 times, for each key byte try.

Which Means……

K = IV → KSA(K) → $X = S_{B+3}[1] < B+3$, $X + S_{B+3}[X] = B+3$

Repeat 6,000,000-8,000,000 times, for each key byte try.

I'd go make some dinner and come back in a couple years.

FMS Attack - RC4 Algorithms

KSA(K)
Initialization:
    For i = 0 .. N - 1
        S[i] = i
    j = 0
Scrambling:
    For i = 0 .. N - 1
        j = j + S[i] + K[i mod l]
        Swap(S[i], S[j])

PRGA(K)
Initialization:
    i = 0
    j = 0
Generation Loop:
    i = i + 1
    j = j + S[i]
    Swap(S[i], S[j])
    Output z = S[S[i] + S[j]]

FMS Attack - KSA Initialization

Let N = 16
Let B = 0
Let IV = B + 3, f, 7
Let SK = 1, 2, 3, 4, 5
Let K = IV . SK = 3, f, 7, 1, 2, 3, 4, 5
Let l = the # of elements in K
Assume no elements get swapped when l > B + 3

Initialization

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Initialization

FMS Attack - KSA Scrambling

Let N = 16
Let B = 0
Let IV = B + 3, f, 7
Let SK = 1, 2, 3, 4, 5
Let K = IV . SK = 3, f, 7, 1, 2, 3, 4, 5
Let l = the # of elements in K
Assume no elements get swapped when l > B + 3

KSA(K)
Scrambling:
For i = 0 .. N - 1
j = j + S[i] + K[i mod l]
Swap(S[i], S[j])

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Initialization
3			0													i=0, j=0+0+3=3

FMS Attack - KSA Scrambling

Let N = 16
Let B = 0
Let IV = B + 3, f, 7
Let SK = 1, 2, 3, 4, 5
Let K = IV . SK = 3, f, 7, 1, 2, 3, 4, 5
Let l = the # of elements in K
Assume no elements get swapped when l > B + 3

KSA(K)
Scrambling:
For i = 0 .. N - 1
j = j + S[i] + K[i mod l]
Swap(S[i], S[j])

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Initialization
3			0													i=0, j=0+0+3=3
	0		1													i=1, j=3+1+f=3

FMS Attack - KSA Scrambling

Let N = 16
Let B = 0
Let IV = B + 3, f, 7
Let SK = 1, 2, 3, 4, 5
Let K = IV . SK = 3, f, 7, 1, 2, 3, 4, 5
Let l = the # of elements in K
Assume no elements get swapped when l > B + 3

KSA(K)
Scrambling:
For i = 0 .. N - 1
j = j + S[i] + K[i mod l]
Swap(S[i], S[j])

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Initialization
3			0													i=0, j=0+0+3=3
	0		1													i=1, j=3+1+f=3
		c										2				i=2, j=3+2+7=c

FMS Attack - KSA Scrambling

Let N = 16
Let B = 0
Let IV = B + 3, f, 7
Let SK = 1, 2, 3, 4, 5
Let K = IV . SK = 3, f, 7, 1, 2, 3, 4, 5
Let l = the # of elements in K
Assume no elements get swapped when l > B + 3

KSA(K)
Scrambling:
For i = 0 .. N - 1
j = j + S[i] + K[i mod l]
Swap(S[i], S[j])

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Initialization
3			0													i=0, j=0+0+3=3
	0		1													i=1, j=3+1+f=3
		c										2				i=2, j=3+2+7=c
			5		1											i=3, j=c+1+1=e

Note that S[B+3] contains information relating to SK[B], since SK[B] is used to calculate j

FMS Attack - KSA Scrambling

Let N = 16
Let B = 0
Let IV = B + 3, f, 7
Let SK = 1, 2, 3, 4, 5
Let K = IV . SK = 3, f, 7, 1, 2, 3, 4, 5
Let l = the # of elements in K
Assume no elements get swapped when l > B + 3

KSA(K)
Scrambling:
For i = 0 .. N - 1
j = j + S[i] + K[i mod l]
Swap(S[i], S[j])

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Initialization
3			0													i=0, j=0+0+3=3
	0		1													i=1, j=3+1+f=3
		c										2				i=2, j=3+2+7=c
			e											1		i=3, j=c+1+1=e
3	0	c	e	4	5	6	7	8	9	a	b	2	d	1	f	Done with KSA

FMS Attack - PRGA Initialization

Let N = 16
Let B = 0
Let IV = B + 3, f, 7
Let SK = 1, 2, 3, 4, 5
Let K = IV . SK = 3, f, 7, 1, 2, 3, 4, 5
Let l = the # of elements in K
Assume no elements get swapped when I > B + 3

PRGA(K)
Initialization:
i = 0
j = 0

3	0	c	e	4	5	6	7	8	9	a	b	2	d	1	f	Initialization

FMS Attack - PRGA Generation Loop

Let N = 16
Let B = 0
Let IV = B + 3, f, 7
Let SK = 1, 2, 3, 4, 5
Let K = IV . SK = 3, f, 7, 1, 2, 3, 4, 5
Let l = the # of elements in K
Assume no elements get swapped when I > B + 3

PRGA(K)
Generation Loop:
i = i + 1
j = j + S[i]
Swap(S[i], S[j])
Output z = S[S[i] + S[j]]

3	0	c	e	4	5	6	7	8	9	a	b	2	d	1	f	Initialization
0	3															i=1, j=0+0=0, z=S[3 + 0]=e

e is the output for the first byte

FMS Attack - PRGA Generation Loop

Let N = 16
Let B = 0
Let IV = B + 3, f, 7
Let SK = 1, 2, 3, 4, 5
Let K = IV . SK = 3, f, 7, 1, 2, 3, 4, 5
Let l = the # of elements in K
Assume no elements get swapped when l > B + 3

PRGA(K)
Generation Loop:
i = i + 1
j = j + S[i]
Swap(S[i], S[j])
Output z = S[S[i] + S[j]]

3	**0**	**c**	**e**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**2**	**d**	**1**	**f**	Initialization
0	3															i=1, j=0+0=0, z=S[3 + 0]=e

e is the output for the first byte

e is then xor'ed with the first byte of data, which is always 0xaa on ip networks

FMS Attack - KSA Scrambling

Let N = 16
Let B = 0
Let IV = B + 3, f, 7
Let SK = 1, 2, 3, 4, 5
Let K = IV . SK = 3, f, 7, 1, 2, 3, 4, 5
Let l = the # of elements in K
Assume no elements get swapped when l > B + 3

KSA(K)
Scrambling:
For i = 0 .. N - 1
j = j + S[i] + K[i mod l]
Swap(S[i], S[j])

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**	Initialization
3			**0**													i=0, j=0+0+3=3
	0		**1**													i=1, j=3+1+f=3
		c										**2**				i=2, j=3+2+7=c
			e											**1**		i=3, j=c+1+1=e
3	**0**	**c**	**e**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**2**	**d**	**1**	**f**	Done with KSA

FMS Attack - KSA Scrambling

Let N = 16
Let B = 0
Let IV = B + 3, f, 7
Let SK = 1, 2, 3, 4, 5
Let K = IV . SK = 3, f, 7, 1, 2, 3, 4, 5
Let l = the # of elements in K
Assume no elements get swapped when l > B + 3

KSA(K)
Scrambling:
For i = 0 .. N - 1
j = j + S[i] + K[i mod l]
Swap(S[i], S[j])

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Initialization
3			0													i=0, j=0+0+3=3
	0		1													i=1, j=3+1+f=3
		c										2				i=2, j=3+2+7=c
			e											1		i=3, j=c+1+1=e
3	0	c	e	4	5	6	7	8	9	a	b	2	d	1	f	Done with KSA

FMS Attack - Reversing Output to Key Byte

- $S_{-1}[X]$ is the location of X in S
- $SK[B] = S_{-1}[Out] - j - S[i + 1]$
- $SK[B] = S_{-1}[e] - c - 1 = 1$

FMS-riassunto

- FMS attacco di tipo *known-plaintext*.
- Basato sul fatto che i bit dell'output dipendono sempre dai primi valori del S-box dato che i e j sono inizializzati a 0.

$i = (i+1) \bmod 256$
$j = (j+S_i) \bmod 256$
swap S_i and S_j
$t = (S_i + S_j) \bmod 256$
$K = S_t$

- Primo random byte di RC4 dipende da:

`S[1], S[S[1]], S[S[1]+S[S[1]]]`

FMS

- Attacco statistico che consente il recupero della chiave
- Basato su abilità di derivare informazioni sulla chiave osservando `S[S[1]+S[S[1]]]`
- Per montare l'attacco, cerca IV che pongono l'algoritmo di key set up in uno stato in cui rilascia informazione sulla chiave

FMS

- Casi in cui viene rilasciata informazione su chiave: *casi risolti*
- Non è difficile testare quando un pacchetto produce IV e output byte che risultano risolti.
- Ogni pacchetto risolto rilascia informazione solo su un byte della chiave
- Bisogna intuire correttamente ogni byte della chiave prima che un pacchetto dia informazione su byte successivo della chiave

FMS

- Bisogna intuire ogni key byte - attacco è di natura statistica: ogni pacchetto risolto dà probabilità del 5% di intuizione corretta di un byte

FMS Attack

- Attack
  - Certain initialization vectors setup RC4 so it reveals key information in its output bytes
  - Invariance weakness allows you to use output bytes to derive the most possible key bytes
  - 802.11b specification's SNAP header makes the first few output bytes predictable

FMS Attack - Analysis of Results

- Results are correct ~ e^{-3} = 5% of the time
- In cases where results rely on only 2 elements in the array not being swapped, it is correct ~ e^{-2} = 13% of the time
- The correct key values will become more evident as you collect more data
- Once the first key byte is derived, it is used to crack the next byte, and so on

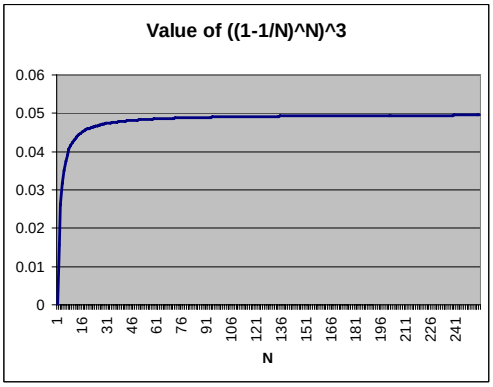
Why 5%?

- This number comes from aggregating the probability that a byte is unchanged over each step over the three bytes.
  P(1 byte is unchanged after one random swap) = (1 – 1/N)
  N is the length of the resulting keystream.
  P(1 byte is unchanged after N random swaps) = $(1 – 1/N)^N$
  P(3 bytes are unchanged after N random swaps) = $((1 – 1/N)^N)^3$
  P(3 bytes are unchanged after N random swaps) = $(e^{-1})^3 = e^{-3}$

$$\exp(x) = \lim_{n \to \infty} \left(1 + \frac{x}{n}\right)^n$$

- The expression, $((1 – 1/N)^N)^3$, can be modeled as (e^{-3}) because as N grows to be of any applicable length, the value of the expression asymptotically heads for 0.05.
- In the end, the value of N is irrelevant as the value is always just below 5%

Why 5%?



FMS

- Esaminando gran numero di casi risolti, possiamo attenderci di vedere bias verso byte della chiave.
- Attacco statistico che consente recupero della chiave dopo 60 IV differenti (stessa chiave): stima di 4,000,000 pacchetti.
- Attacco di Klein (Esercizio)

Attacchi alla integrità: rottura del CRC

Iniezione di messaggi

- Violare il controllo d'accesso con l'iniezione di messaggi
  - Con un keystram recuperato, può essere costruito un valido testo cifrato : C' = P ⊗ RC4(*v,k*)
  - I messaggi nel testo cifrato costruito possono essere iniettati nella rete senza scatenare alcun allarme poichè gli IV sono riutilizzabili .

Attacco del tipo "Bit-Flipping"

1. l'attaccante monitora il traffico di rete tramite uno sniffer
2. l'attaccante cattura un frame e modifica in maniera casuale alcuni bit del payload del frame
3. successivamente l'attaccante modifica, conseguentemente, l'ICV del frame
4. a questo punto l'attaccante trasmette il frame modificato verso la destinazione
5. l'Access Point (od un ricevitore wireless, se la WLAN è stata realizzata senza AP) riceve il frame modificato e ne calcola l'ICV basandosi sul contenuto dei dati trasportati nel frame stesso

Attacco del tipo “Bit-Flipping”

6. l’Access Point (od un ricevitore wireless, se la WLAN è stata realizzata senza AP) confronta l’ICV calcolato con quello trasportato dal frame

7. poiché i due ICV sono uguali, l’Access Point (od un ricevitore wireless, se la WLAN è stata realizzata senza AP) accetta il frame in chiaro in quanto ritenuto integro

8. il ricevitore wireless a cui è destinato il traffico decapsula il contenuto del payload del frame ricevuto (eventualmente per il tramite di un AP), ricostruisce e processa i pacchetti di livello 3

9. poiché i bit del payload del frame sono stati modificati dal attaccante, i controlli sui CRC dei pacchetti di livello 3 da esso trasportati falliscono

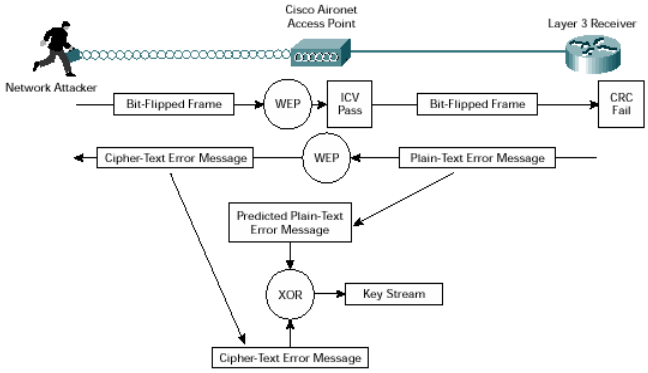
Attacco del tipo “Bit-Flipping”

10. lo stack IP del ricevitore genera dunque un errore, che causa la trasmissione di messaggi di errore dal ricevitore verso il mittente

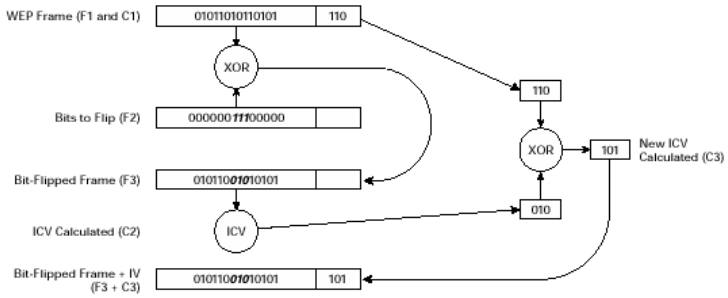
11. a questo punto l'attaccante, avvalendosi di uno sniffer, rintraccia i frame cifrati provenienti dal ricevitore e che trasportano, nel payload, i messaggi di errore di livello 3 diretti verso la sorgente del frame precedentemente ricevuto

12. una volta rintracciati tali frame, l'attaccante è in grado di derivare il key stream allo stesso modo di quanto avviene in un attacco di tipo IV replay

Attacco del tipo “Bit-Flipping”



Calcolo dell’ICV da parte dell’hacker



1. un determinato frame (F1 nella figura) ha associato un certo ICV (C1)
2. l'attaccante genera un nuovo frame (F2) della stessa lunghezza del frame F1 ma con alcuni bit del payload modificati; ad esso sarà associato un certo ICV (C2) calcolato dall’ attaccante
3. un terzo frame (F3) viene generato mediante il calcolo della funzione XOR, bit a bit, del frame F1 e di quello F2
4. successivamente viene calcolato l’ICV associato al frame F3 (C3) applicando la funzione XOR, bit a bit, di C1 e C2

Calcolo dell'ICV da parte dell'hacker

$$c(x \oplus y) = c(x) \oplus c(y)$$

$$C = RC4(v,k) \oplus (M, c(M))$$

$$C' = C \oplus (D, c(D))$$

$$= RC4(v,k) \oplus (M, c(M) \oplus (D, c(D))$$

$$= RC4(v,k) \oplus (M \oplus D, c(M) \oplus c(D)$$

$$= RC4(v,k) \oplus (M', c(M \oplus D))$$

$$= RC4(v,k) \oplus (M', c(M'))$$

Rompere WEP

- Numerosi tool sw (disponibili nel pubblico dominio) per fare sniffing su reti wireless.
- Rompono WEP dopo aver osservato un numero sufficiente di pacchetti
- In pratica basta osservare qualche milione di pacchetti (mezza giornata…)

Suggerimenti per Wireless

- Assumere che link layer non offra alcuna sicurezza.
- Usare meccanismi di sicurezza a livello più alto (IPsec, SSH,…) senza affidarsi a WEP.
- Trattare tutti i sistemi connessi via 802.11 come esterni (e.g., mettere access point fuori da firewall!)
- Assumere che ognuno dentro il range fisico possa comunicare come utente valido nella rete.
- Considerare che si possono utilizzare antenne (sofisticate) che arrivano a range molto più elevati delle 802.11 card.

Tipologie di attacchi WEP

- Attacchi passivi basati su analisi statistiche.
- Attacchi attivi che iniettano traffico da stazioni mobili non autorizzate (known plaintext).
- Attacchi attivi che decriptano il traffico.
- Attacchi basati su dizionario che, dopo una giornata di traffico consentono in real-time decryption automatico di tutto il traffico.

WEP usa RC4:

Problema è implementazione di RC4 in 802.11 (IV)

WEP: a che punto siamo?

- War-driving
  - Nuovo hobby (Web page)
  - Hacker organizzano gite per cercare reti wireless
- IEEE sta avendo tempi lunghi per risolvere problemi di standard.
- Implementazioni attuali ancora vulnerabili a molti livelli…

Lezione da WEP?

- Problemi comuni anche in altri protocolli…
- Design di protocolli di security compito estremamente difficile
- Affetto da molte complicazioni
- Richiede expertise speciali oltre l'ingegnerizzazione di protocolli
- Dal punto di vista ingegneristico uso di CRC-32 e di RC4 giustificato da velocità e facilità di implementazione
- Ma proprio l'implementazione ha reso WEP estremamente vulnerabile ad attacchi attivi, passivi, basati su dizionario…

Lezione da WEP (cont.)

- Interazioni molto sottili tra ingegnerizzazione e security
- Protocolli stateless e liberali nell'accettare input sono buoni dal punto di vista dell'ingegnerizzazione
- Dal punto di vista della security estremamente pericoloso: fornisce all'attaccante più libertà di operare (traffic injection)
- Security è una proprietà dell'intero sistema: ogni decisione va esaminata tenendo presente security! (quanti lo fanno?)

Stubblefield, Ioannidis, and Rubin

Simply implemented the attack using inexpensive hw

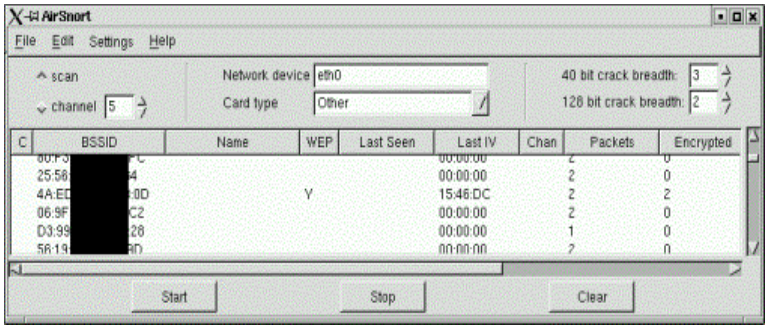
Identified other weaknesses in WEP:

- WEP keys are ASCII (no hashing is done!): it limits possible key space to ASCII equivalents to letters.
- WEP is link layer protocol: encrypts network layer data.
  - First byte going to be the IP packet.
  - Worse, 802.11, in order to be compatible with IP as well as IPX and other network protocols, uses the 802.2 logical link layer encapsulation.
  - This just means that all packets always start with the same 802.2 header.
  - Guessing the first byte is trivial!

Breaking WEP

- Building on those attacks there are several software tools, widely and freely available over Internet, to sniff wireless transmissions.
- Some have ability to break WEP if provided with sufficient number of encrypted packets:
  - Airsnort (airsnort.shmoo.com),
  - WEPcrack (wepcrack.sourceforge.net),
  - ethereal (www.ethereal.com) - used by Stubblefield et al.

Breaking WEP



- AirSnort works by setting wireless NIC (Network Interface Card) into promiscuous mode.
- Capable of capturing SSIDs (Service Set ID), whether WEP enabled, last IV transmitted, number of packets sent, encrypted packets, and so on.

Conclusions of Stubblefield et al.

- These improvements brought down number of packets required by an attacker from 5 million to about 1 million.
- Conclusions of Stubblefield et al:
  - Assume that the link layer offers no security.
  - Use higher-level security mechanisms such as IPsec and SSH for security, instead of relying on WEP.
  - Treat all systems that are connected via 802.11 as external. Place all access points outside the firewall.
  - Assume that anyone within physical range can communicate on the network as a valid user.
  - Keep in mind that an adversary may utilize a sophisticated antenna with much longer range than found on a typical 802.11 PC card.

Summary of WEP problems

- Passive attacks based on statistical analysis.
- Active attack to inject new traffic from unauthorized mobile stations, based on known plaintext.
- Active attacks to decrypt traffic, based on tricking the access point.
- Dictionary-building attack that, after analysis of about a day's worth of traffic, allows real-time automated decryption of all traffic.
- Uses RC4: the problem is implementation of RC4 in 802.11 (IV), not encryption!

Summary of WEP problems (cont.)

- War-driving
  - New hobby (Web pages)
  - Hackers pack into a car and cruise around looking for networks that allow entrance
- IEEE process is taking a long time to fix the standard.
- Current implementations are still vulnerable at the many levels…

Note on RC4

- RC4 is efficient stream cipher that can be used securely
- Implementation of RC4 in other protocols (e.g., SSL) not affected by this attack
  - SSL preprocesses encryption key and IV by hashing with both MD5 and SHA1 (different sessions have different keys)
- Standard RSA recommendation: discard first 256 bytes of RC4 output (expensive for small packets)

Symmetric Key Cryptography

Gianluigi Me
Me@disp.uniroma2.it
Anno Accademico 2008/09

Main sources: Network Security Essential / Stallings
Applied Cryptography / Schneier
Handbook of Applied Cryptography / Menezes, van Oorschot, Vanstone